GOLDILOCKS 22c.1 User Manual (en)

SUNJESOFT Inc.

Table of Contents

Part I.

Getting Started

1.

Preface

1.1 Overview

This user manual is intended for the user who is configuring, managing and operating GOLDILOCKS. The purpose of this manual is to convey the basic concepts required for installation and management of GOL DILOCKS. This manual also describes cautions when using GOLDILOCKS system.

- The description which is presented in this document can be changed according to the intstallation en vironment and specific usage.
- This document is based on GOLDILOCKS 22c.1 version.
- This document is based on RedHat linux - based platform.

Target Reader

The target readers are as follows.

- Programmers who need basic knowledge about how to manage GOLDILOCKS database
- Administrators and performance managers of GOLDILOCKS database
- Administrators and performance managers of GOLDILOCKS cluster system

1.2 Summary

This chapter describes the basic structure and characteristics of GOLDILOCKS for the novice user. A user c an select either a standalone or a cluster system architecture to use GOLDILOCKS. Differences of each arc hitecture and their usages are described.

GOLDILOCKS Database Management System

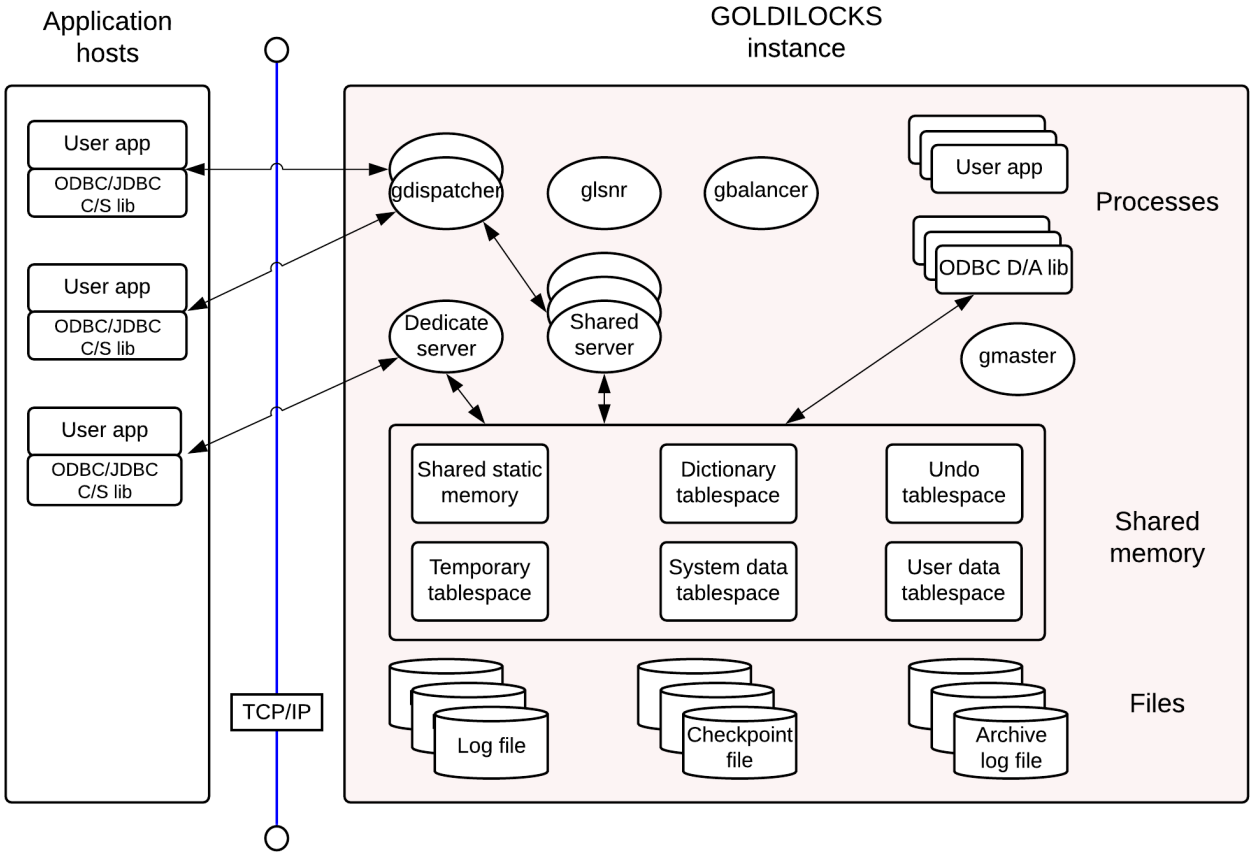
GOLDILOCKS database system consists of the following parts.

- User-installed GOLDILOCKS software binaries
- Database which is a set of tablespaces, implemented with one or more shared memory
- Various files for persistence support of database
  - The data files created with the same size as shared memory per shared memory
  - Redo log files which support the recovery of the database in the event of a failure
  - Configuration files for database settings
  - Trace log files which record information such as events during database operations
- gmaster processes which manage database, and many system threads within it

GOLDILOCKS Architecture

To prevent spreading the application process failure over the entire database system, GOLDILOCKS datab ase is a multi-process architecture based on shared memory, instead of a multi-thread architecture. The o verall architecture of GOLDILOCKS database is as shown in figure 1. Data are loaded onto a shared mem ory and gmaster process is a management daemon which manages database such as boot-up, log flush, aging. Also, it stores redo log files and data files on a disk file to ensure the permanence of data. Applicat ions using GOLDILOCKS database will use one of the following two accessing models.

Figure 1 GOLDILOCKS architecture



- Direct access (D/A) model
  - A user may use D/A model when user application is operated on the same equipment as GOLDIL OCKS database.
  - User applications should be used by linking with GOLDILOCKS ODBC/JDBC libraries for D/A.
  - GOLDILOCKS ODBC/JDBC libraries for D/A include query processing module and storage manage ment module inside and they process user request by directly attaching shared memory configuri ng that database.
  - It is suitable to implement a few works which require low-latency because the communication lo ad between application process and database process module is removed.
- Client/ Server (C/S) model
  - A user may use C/S model when user application is operated on the same equipment as GOLDIL OCKS database or on a different equipment.
  - User applications should be used by linking with GOLDILOCKS ODBC/JDBC libraries for C/S.
  - The GOLDILOCKS development library for C/S processes user's requests through TCP communica tion with the server (gserver) which serves the database specified in the connect string.
  - The response speed of single application of C/S is inferior to that of D/A. However, C/S does not depend on the location of the application, and the relatively stable operation is possible even wh en an error occurs in the application.
  - C/S model may be operated in shared mode or dedicated mode. In dedicated mode, a single serv er (gserver) process is performed on a single client. In shared mode, it responds to multiple client

s because dispatcher (gdispatcher) and shared-server (gserver) are always running.
- Dedicated mode is suitable for large amount of data, and shared mode is suitable for many client s, small amount of data.
- For more information about setting dedicated mode or shared mode, refer to **odbc.ini File** and **Lis tener Configuration**.

In D/A model, an application directly accesses to database and manipulates it, and thus the data base instance becomes unstable because many errors occur at an early development stages. The refore, it would be efficient to develop it in C/S model at an early development stage, and then, to switch it to D/A model at the final development stage.

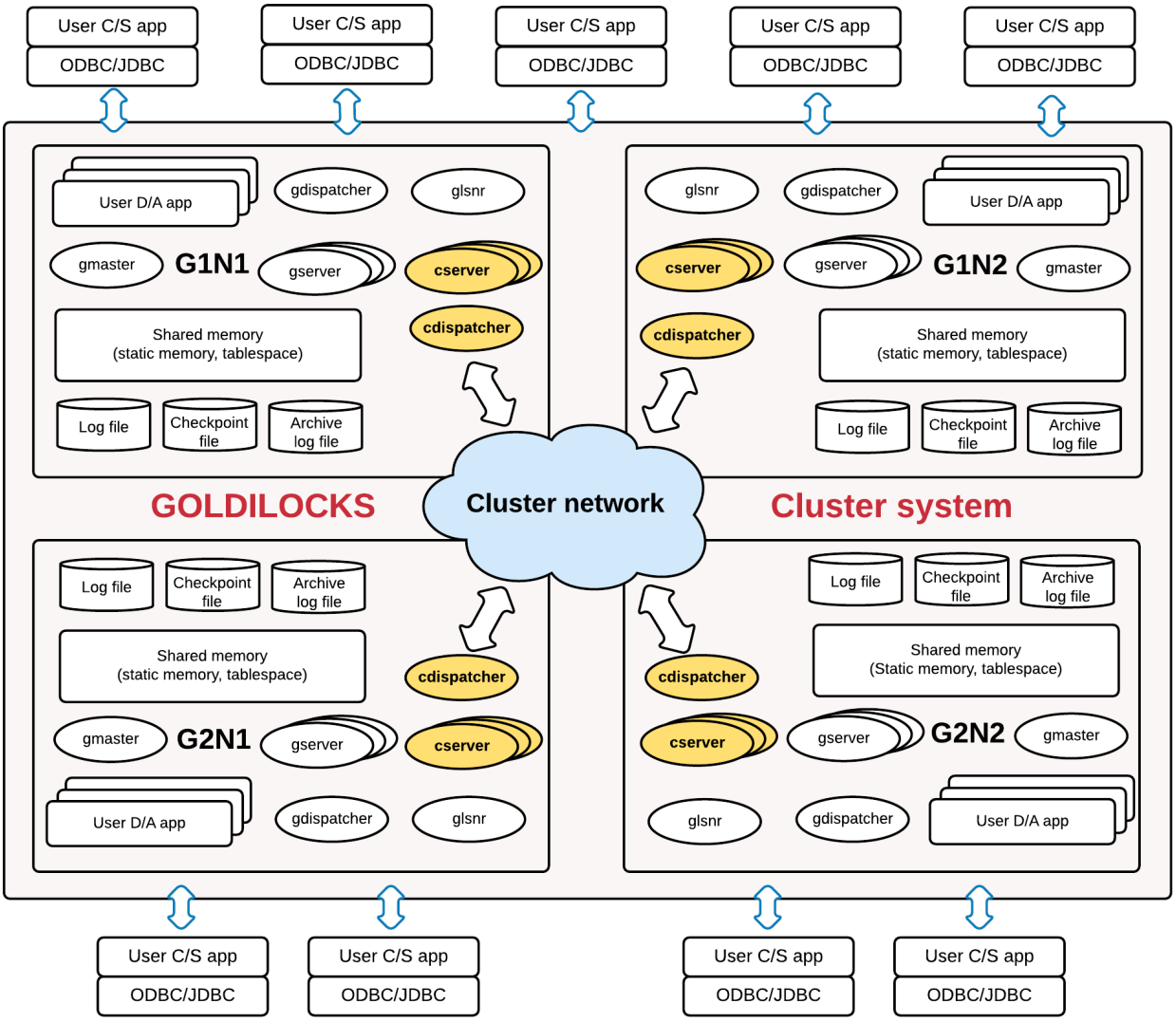
GOLDILOCKS Cluster System Architecture

GOLDILOCKS can be used by configuring a standalone database, or binding multiple databases into a sin gle cluster and managing the database in cluster unit. In other words, a user can distribute and store tabl e data into multiple nodes according to the desired sharding strategy. This guarantees high availability an d improves the throughput due to the parallel processing.

GOLDILOCKS cluster system guarantees ACID of transaction which is clister-widely performed. There fore, it provides the data reliability as same as that of the transaction performed on a standalone server when a ny node belonging to the cluster system is connected to perform the transaction.

Each database belonging to GOLDILOCKS cluster system has a structure for multi-process structure and d ata loading method, which is as same as the structure of the standalone database. However, cdispatcher process and cluster server (cserver) process are added. cdispatcher is a process for efficient communiation between member nodes in a cluster, and cluster server (cserver) process is for the data storage and mana gement on the cluster member node. Also, tablespaces and management areas for transaction managem ent of cluster system are added to the shared memory.

Figure 2 GOLDILOCKS cluster system architecture



1.3 Characteristics of GOLDILOCKS Cluster

Features of Cluster

GOLDILOCKS cluster is a cluster system of shared nothing structure and it overcomes limitations for trans action performance and storage of an existing standalone system.

- High throughput
  - GOLDILOCKS cluster does not limit creating groups, and the linear performance can be improved by creating groups.
  - It overcomes the limitation for the storage space of the existing memory-based standalone syste m.
- High availability
  - If a group consists of multiple members and at least one member is running in a group, it does n ot affect the availability.
  - Even when not every member in a group is available, other groups except for that group normall y provides service.
- Online expansion and online recovery
  - creating groups or members are possible even when the service is in progress, and it does not aff ect the service in progress.
  - Even the member of the suspended service due to an error can participate in a cluster online.
- Providing the perfect transaction
  - It perfectly provides the following properties of which a transaction should comply with.
    - Atomicity
    - Consistency
    - Isolation
    - Durability
- Providing the perfect MVCC (multi-version concurrency control)
  - GOLDILOCKS cluster provides a global statement level consistency as standalone system does.
  - The SQL starting at a specific point can access a desired version among the various versions when accessing to any node.
- Providing the standard SQL and the standard DBC
  - It provides SQL which is equivalent to SQL 92.
  - It provides standard DBCs such as JDBC, ODBC.
- Application compatibility
  - The application source or SQL developed in the existing standalone system can be used in GOLDI LOCKS cluster without modifying it.

Constraint of Cluster

All SQL statements in GOLDILOCKS cluster can be used same as those in standalone system except for th e following constraints.

PRIMARY KEY for the sharded table, UNIQUE constraint and the UNIQUE INDEX should include a sharding key.

The following is an example of a failure because the constraint of UNIQUE (name) does not include an id column which is a sharding key.

```
gSQL>
CREATE TABLE t1
(
    id   INTEGER PRIMARY KEY,
    name VARCHAR(128),
    UNIQUE (name)
)
SHARDING BY HASH(id);
ERR-HYC00(16380): UNIQUE or PRIMARY KEY must include all sharding key columns for cluster
system
```

The constraint should be generated including sharding key as like UNIQUE (id, name) or UNIQUE (name, id) in the following example.

```
gSQL>
CREATE TABLE t1
(
    id   INTEGER PRIMARY KEY,
    name VARCHAR(128),
    UNIQUE( id, name )
)
SHARDING BY HASH (id);
Table created.
```

Non-deterministic statements should have the global secondary index to distinguish the same ro ws among the cluster members.

The following is an example of error which occurs when creating a table omitting the global secondary in

dex by force.

```
gSQL> CREATE TABLE t1 ( c1 INTEGER ) WITHOUT GLOBAL SECONDARY INDEX;
Table created.
gSQL> INSERT INTO t1 VALUES (1), (2), (3), (4), (5);
5 rows created.
gSQL> COMMIT;
Commit complete.
```

The following is an example of deleting three rows and it does not guarantee that the cluster members d elete the same rows.

```
gSQL> DELETE FROM t1 FETCH 3;
ERR-42000(16423): does not support non-deterministic DML in the cluster system : global
secondary index expected
```

The following example does not guarantee that the cluster members update the same rows to the same value by using RANDOM(1, 100).

```
gSQL> UPDATE t1 SET c1 = RANDOM(1, 100);
ERR-42000(16423): does not support non-deterministic DML in the cluster system : global
secondary index expected
```

The following is an example of updating the row at the current position by using updatable cursor, and it requires the global secondary index to distinguish the same rows among cluster members.

```
gSQL> \var v1 INTEGER
gSQL> DECLARE cur1 CURSOR FOR SELECT c1 FROM t1 FOR UPDATE;
Cursor declared.
gSQL> OPEN cur1;
Cursor is open.
gSQL> FETCH cur1 INTO :v1;
V1
--
 1
1 row fetched.
gSQL> UPDATE t1 SET c1 = 1 WHERE CURRENT OF cur1;
ERR-42000(16423): does not support non-deterministic DML in the cluster system : global
secondary index expected
```


It does not support the deferrable constraint.

```
gSQL> ALTER TABLE t1 ADD CONSTRAINT t1_uk UNIQUE(id) DEFERRABLE;
ERR-HYC00(16388): does not support deferrable constraints in the cluster system :
ALTER TABLE t1 ADD CONSTRAINT t1_uk UNIQUE(id) DEFERRABLE
                                    *
ERROR at line 1:
```


It does not guarantee the sequence when using the same sequence in different servers.

- It is executed in g1n1 server.

```
gSQL> SELECT seq1.NEXTVAL FROM dual;
NEXTVAL
-------
      1
1 row selected.
gSQL> SELECT seq1.NEXTVAL FROM dual;
NEXTVAL
-------
      2
1 row selected.
```

- It is executed in g2n1 server.

```
gSQL> SELECT seq1.NEXTVAL FROM dual;
NEXTVAL
-------
     21
1 row selected.
```

- It is executed again in g1n1 server.

```
gSQL> SELECT seq1.NEXTVAL FROM dual;
NEXTVAL
-------
      3
1 row selected.
```

2.

Tutorial

2.1 Managing GOLDILOCKS Instance

This chapter describes the basic knowledge of managing the GOLDILOCKS instance.

Overview

GOLDILOCKS database system consists of database and instance. Database is a collection of various files which are necessary for driving database such as dictionary data on memory, user data, data file for dictio nary data and user data, online redo files.

Database instance consists of two parts. One is the memory portions containing the run-time information for operating GOLDILOCKS database, and the other is the background process which is used to operate a nd manage it. Each database instance is identified as shared memory key value and "GOLDILOCKS _DATA " environment variable value, both are used to configure shared memory.

Property Setting

GOLDILOCKS properties are listed in *$GOLDILOCKS _DATA/conf/goldilocks.properties.conf* file. A user m ay install GOLDILOCKS using the basic properties. This chapter describes the main properties such as TBS (Tablespace), LOG, CONTROL FILE.

The followings describe main property items of when installing GOLDILOCKS.

Table 2-1 Main property items

Property	Description	Default value
SYSTEM_TABLESPACE_DIR	It is the directory path of installing the following system TBS. • DICTIONARY_TBS • MEM_DATA_TBS • MEM_UNDO_TBS • MEM_TEMP_TBS • MEM_TRANS_TBS	'<GOLDILOCKS_DATA>/db'
SYSTEM_MEMORY_DICT_TABLESPACE_SIZE	It is the dictionary tablespace size.	128M
SYSTEM_MEMORY_DATA_TABLESPACE_SIZE	It is the memory data tablespace size.	200M
SYSTEM_DISK_DATA_TABLESPACE_SIZE	It is the disk data tablespace size.	200M
SYSTEM_MEMORY_UN	It is the undo tablespace size.	32M

Property	Description	Default value
DO_TABLESPACE_SIZE		
LOG_DIR	It is the default log directory path.	'<GOLDILOCKS_DATA>/wal'
SYSTEM_LOGGER_DIR	It is the system log directory path.	'<GOLDILOCKS_DATA>/trc'
CONTROL_FILE_COUNT	It is the number of control files.	2
CONTROL_FILE_0	It is the first control file path.	'<GOLDILOCKS_DATA>/wal/control_0.ctl'
CONTROL_FILE_1	It is the second control file path.	'<GOLDILOCKS_DATA>/wal/control_1.ctl'
BUFFER_CACHE_SIZE	It is the size of the buffer cache which is used to cache the table and the index page created in the disk tablespace.	64M

- DICTIONARY_TBS: It is the tablespace which stores dictionary tables of GOLDILOCKS.
- MEM_DATA_TBS: It is the memory tablespace in which the user table/index is generated.
- DISK_DATA_TBS: It is the disk tablespace in which the user table/index is generated.
- MEM_UNDO_TBS: It is the tablespace which contains the undo (rollback) information of the transaction.
- MEM_TRANS_TBS (Cluster only): It is the tablespace used for the global transaction recovery in cluster system. The size is automatically calculated based on the transaction table size and the number of cluster node.

A user can change the text property file ($GOLDILOCKS_DATA/conf/goldilocks.properties.conf), or define a new variable in the form of GOLDILOCKS_<property_name> to change the database settings or instance settings. In the priority, the property file takes precedence over the environment variable.

Background Process

GOLDILOCKS has a background process (gmaster) for managing instance. gmaster consists of multiple system thread internally, and the contents are as follows.

Table 2-2 System threads

Thread	Description
Main thread	It starts or ends gmaster process.
Log archiving thread	It copies the previous redo log file to a specified location when switching online redo file, and stores it.
Ager thread	It cleans up the resources being used by dropped schema objects.
Page flusher thread	It distributes the task to the IO slave and controls them in order to store the dirty pages in the disk at checkpoint.
Log flusher thread	It periodically collects the log records accumulated in the redo log buffer, and then stores them into online redo log file at run-time.
Checkpoint thread	It downloads dirty pages to the data file on the disk and the online redo log file when swi tching redo log file.
Cleanup thread	It cleans up resources used by abnormally terminated clients, and then rolls back the trans actions.
IO slave thread	It performs all disk IO related to data file such as checkpoint and data file loading.
Process monitor thread	It monitors after executing the processes such as balancer (gbalancer), dispatcher (gdispa tcher), shared-server (gserver), then reexecutes when abnormal termination is detected.
Cluster Recover Thread (Cluster only)	It recovers a global transaction in cluster system.
Failover Thread (Cluster only)	It deals with the failover through reselecting offiline and coordinator for the members wh en an error occurs on a specific node or in a network in cluster system.

Client Process

Client/ Server Model

The Client/ Server (C/S) model application is connected to listener (glsnr) which is waiting for access requ est. Then it creates a new database service process (gserver), in dedicated mode, and it handles user's req uest by using TCP communication. All these operations are carried out through inter process communicat ion, so any signal generated in the application process does not affect on the state of the database. More over, cleanup thread regularly checks and returns all the resources used by abnormally terminated applica tion.

Direct Access Model

GOLDILOCKS supports a direct access (D/A) model as well as Client/ Server (C/S) model. All applications u sing D/A model are linked to the server library supported by GOLDILOCKS, and then directly access datab ase and instance. Therefore, no other special service process exists but only the application processes doe s exist.

When D/A model application process is interrupted abnormally by the signal generated during operation, all resources in use will be cleaned up by the signal handler function which the library set during connecti on. The function cleans up the resources according to the two following steps.

1. The signal handler marks an abnormal termination on the session object and terminates the process.
2. The cleanup thread of gmaster return resources to the database in the same way as C/S model after a certain period of time.

Application processes directly access the database area in D/A model. Therefore, comply with the followi ng precautions.

- When the D/A model application process is terminated by a fatal signal such as SEGV, the signal han dler which was registered by the library at connection to the database should stop using shared resou rces. If a specific signal handler should be installed in the application it should be declared before con necting to the database.
- When forcibly shut down the application process, do not use SIGKILL (kill-9) because the process can not detect the generation of the signal. A user should use SIGTERM, SIGQUIT or SIGUSR2.

Memory Architecture of Instance

The memory size used by the database instance is determined by the relevant properties in the property fi le. Shared memory used by instance can be divided into static area and tablespace area. Static area includ es basic information about public instance, each session, statement, transaction, redo log buffer, dictiona ry cache and several other operation. Tablespace area includes page frame of each tablespace and Page Control Header (PCH) for controlling them.

The application process memory includes instance memory attached at connection. Additionally, it includ es process basis sharing ODBC environment, several ODBC handles, heap memory area with bind informa tion.

Startup and Shutdown Instance

To startup the GOLDILOCKS instance, set the SHARED_MEMORY_STATIC_KEY property differently from other instances. After that, a user can startup the GOLDILOCKS instance by using gsql. Execute it to take sysdba role as follows.

A user should run **listener** before startup or shutdown the GOLDILOCKS instance in dedicated mode of C/ S model.
A user can not startup or shutdown the GOLDILOCKS instance in shared mode of C/S model.

```
% gsql sys gliese --as sysdba
Connected to GOLDILOCKS Database.
gSQL>
```

Startup phrase in the GOLDILOCKS instance has several phases as follows.

- NOMOUNT
  - It starts up gmaster process which is a managing daemon of the GOLDILOCKS instance.
- MOUNT
  - It loads properties and recovery control file by using $GOLDILOCKS _DATA environment variable.
- OPEN
  - After loading the tablespace contents from data file, it recovers by using redo log, rebuilds no-logging indexes, creates dictionary cache, and then waits for the user's service connection.

A user can start up the GOLDILOCKS instance by using gsql as follows.

```
gSQL> \startup nomount
Startup success
gSQL> alter system mount database;
System altered.
gSQL> alter system open database;
System altered.
```

To directly enter into OPEN phase, do as follows.

```
gSQL> \startup open
Startup success
```

If GOLDILOCKS instance is shut down, gmaster (the daemon process for management) would be terminated. Then, connection and database operation is no longer possible.

There are four ways to shutdown the GOLDILOCKS instance as follows.

- NORMAL
  - After blocking the access of a new session and waiting until the end of all connected sessions, a user perform a checkpoint and shuts down the instance.
- TRANSACTIONAL
  - After blocking the start of a new transaction and waiting until the end of all running transactions, a user performs a checkpoint and shuts down the instance.
- IMMEDIATE
  - After blocking the execution of a new unit operation (Connection unit with GOLDILOCKS database. e.g. FETCH or EXECUTE, etc.) and waiting until the end of all unit operations, a user rolls back all transactions, performs a checkpoint and shuts down the instance.

- ABORT
  - Regardless of any connected session's status, a user terminates gmaster and shuts down the inst ance.

To shutdown an instance, use gsql with sysdba role and perform `\shutdown`, as follows.

```
% gsql sys gliese --as sysdba
Connected to GOLDILOCKS Database.
gSQL> \shutdown normal
Shutdown success
gSQL>
```

Start and End of Listener

A user should run the listener to provide the service in the client/ server environment.

A user can start the listener as follows.

```
% glsnr --start

Listener is started successfully.
%
```

A user can end the listener as follows.

```
% glsnr --stop

Listener is stopped.
%
```

For more information about listener control, refer to **glsnr**.
For more information about how to start or end cluster system refer to **Start and End of Cluster System**.

2.2 Installing GOLDILOCKS and Creating Database

This chapter describes how to install the GOLDILOCKS software and create a database.

Overview

GOLDILOCKS software is a compressed file with the name such as *goldilocks-<version_no>-<os_type>-<cpu_type>.tar.gz*. After decompressing the file, software binaries, various samples, fundamental database directory structures are created at the corresponding location, and then installation is completed. After the installation, the directory is created, and the directory is named after the package. Then directories whose names are *goldilocks_home* and *goldilocks_data* are created under it.

- goldilocks_home directory
  - Binary home of GOLDILOCKS product
  - Defined as GOLDILOCKS _HOME environment variable
  - Operation binaries, libraries for the client, header files, licenses, etc. are located.
- goldilocks_data directory
  - Home of the user database (instance) which GOLDILOCKS creates.
  - Defined as GOLDILOCKS _DATA environment variable
  - A default location which has data files, log files, property files

After that, use a utility called gcreatedb in *$GOLDILOCKS_HOME/bin* directory to create a database in *$GOLDILOCKS _DATA* directory.

Release Platform

GOLDILOCKS is available in the following release platform.

Table 2-3 Release platform

Platform	Platform name	OS	CPU	Remarks
Server platform	linux-x86_64	linux	x86_64	>= linux kernel 2.6 >= glibc 2.1[1] >= gcc 4.1.2 >= java 1.6 <= java 1.8
				>= linux kernel 2.6 >= glibc 2.1[1]

Platform	Platform name	OS	CPU	Remarks
	linux-powerpc-64	linux	powerpc	>= gcc 4.1.2 >= java 1.6 <= java 1.8
	hpux11.31-itanium-64	HP-UX 11.31	itanium	>= java 1.6 <= java 1.8
	aix7-powerpc-64	AIX 6.1	powerpc	>= java 1.6 <= java 1.8
Client platform	linux-x86_64	linux	x86_64	>= linux kernel 2.6 >= glibc 2.1[1] >= gcc 4.1.2 >= java 1.6 <= java 1.8
	linux-x86_32	linux	x86_32	>= Linux kernel 2.6 >= glibc 2.1[1] >= gcc 4.1.2 >= java 1.6 <= java 1.8
	hpux11.31-itanium-64	HP-UX 11.31	itanium	>= java 1.6 <= java 1.8
	hpux11.31-itanium-32	HP-UX 11.31	itanium	>= java 1.6 <= java 1.8
	aix7-powerpc-64	AIX 7.2	powerpc	>= java 1.6 <= java 1.8
	linux-powerpc-64	linux	powerpc	>= Linux kernel 2.6 >= glibc 2.1[1] >= gcc 4.1.2 >= java 1.6 <= java 1.8
	windows-x86-64	Windows	PENTINUM x86	>= java 1.6 <= java 1.8
	windows-x86-32	Windows	PENTINUM x86	>= java 1.6 <= java 1.8

[1]The user should install libnsl separately in CentOS 8, RHEL 8 or higher.

System Requirements

Check the following requirements before installing GOLDILOCKS.

- At least 2 G should be secured at the physical memory and disk space.

- A sufficient amount of paging (swap) area is required.
- The correct package version which fits into the platform to be installed is required.

GOLDILOCKS Package Configuration

This chapter describes the directory configuration when installing GOLDILOCKS.

Package Directory Configuration

Table 2-4 Parent directory configuration

Directory	Server	Client	Description
GOLDILOCKS_HOME	O	O	Binaries and libraries are installed, overwriting-enabled group when updating
GOLDILOCKS_DATA	O	X	The data storing path, overwriting-unabled group

Table 2-5 Package directory configuration

Parent directory	Package directory	Description
GOLDILOCKS_HOME	admin	Required schema script to create database
	bin	Execution files
	lib	Library files
	include	Header files such as ODBC, XA, Embedded SQL, etc.
	license	License files
	sample	Sample files
	msg	Error message files
	script	Script file for ease of use (It will be supported in future)
	app_dev	Application development
GOLDILOCKS_DATA	conf	Configuration files
	db	Database files
	wal	Log files, control files
	archive_log	Archive log files
	backup	Back up files
	trc	Trace log files, warning message files
	journal	Journal file used at cluster rebalance
	data	Required TxFile to operate clustone

Package File List

The followings are description of files in a directory, and whether it is included in server package or client package.

Table 2-6 admin directory

File name	Server	Client	Description
README	O	X	Read me
DictionarySchema.sql	O	X	Dictionary schema creating script
InformationSchema.sql	O	X	Information schema creating script
PerformanceViewSchema.sql	O	X	Performanceview schema creating script

Table 2-7 admin/ cluster directory

File name	Server	Client	Description
README	O	X	read me
DictionarySchema.sql	O	X	Dictionary schema creating script
InformationSchema.sql	O	X	Information schema creating script
PerformanceViewSchema.sql	O	X	Performanceview schema creating script

The script created in admin/standalone directory is used when using GOLDILOCKS in standalone. On the other hand, the script created in admin/cluster directory is used when using GOLDILOCKS by configuring cluster system.

Table 2-8 bin directory (Unix)

File name	Server	Client	Description
README	O	O	Read me
gmaster	O	X	GOLDILOCKS master
gcreatedb	O	X	Database creating tool
glsnr	O	X	Listener control tool
gbalancer	O	X	Loads balancer for C/S shared
gdispatcher	O	X	Manages multiple connections for C/S shared
gserver	O	X	Instance manager for C/S
gsql	O	X	Interactive SQL tool
gsqlnet	O	O	Interactive SQL tool for C/S
gpec	O	O	Embedded SQL precompiler
logmirror	O	X	Redo log replication tool
cyclone	O	X	CDC replication tool
gloader	O	X	Import/ export tool
gloadernet	O	O	Import/ export tool for C/S
cymon	O	X	CDC monitoring Tool
gdump	O	X	Control/ log/ data/ binary property file viewer

File name	Server	Client	Description
gsyncher	O	X	Synchronization utility for shared memory log and disk log files
tablediff.jar	O	X	Table comparison tool
cdispatcher	O	X	Manages cluster connections and distributes protocols in cluster system
cserver	O	X	Instance manager for cluster system
gtrclogger	O	X	Trace log manager for cluster system
gmon	O	X	Process monitoring tool
galocator	O	X	Location management tool
gagent	O	X	Location provider tool
gloctl	O	O	Interactive location editing tool
cyfile	O	X	CDC exporting file tool
clustone	O	X	CDC replication tool for cluster
clustone_sender	O	X	CDC replication tool for cluster (sender tool)

Table 2-9 bin directory (Windows client)

File name	Server	Client	Description
README	X	O	read me
gloadernet.exe	X	O	Import/export tool for C/S
gpec.exe	X	O	Embedded SQL precompiler
gsqlnet.exe	X	O	Interactive SQL tool for C/S
gloctl.exe	X	O	-

Table 2-10 lib directory (Unix)

File name	Server	Client (64 bit)	Client (32 bit)	Description
README	O	O	O	Read me
libstib.so	O	X	X	Shared library for infiniband
libgoldilocks.a	O	X	X	D/A and C/S-inclusive static library for ODBC
libgoldilocksa.a	O	X	X	D/A-only static library for ODBC
libgoldilocksas.so	O	X	X	D/A-only shared library for ODBC
libgoldilocksc.a	O	O	O	C/S-only static library for ODBC
libgoldilockscs-ul32.so	O	O	X	64 bit-C/S-only shared library for ODBC (SQLLEN = 4 byte)
libgoldilockscs-ul64.so	O	O	X	64 bit-C/S-only shared library for ODBC (SQLLEN = 8 byte)
libgoldilockscs.so	X	X	O	32 bit-C/S-only shared library for ODBC
libgoldilockscvtGB18030_32.so	X	X	O	32 bit GB18030 character set conversion library
libgoldilockscvtGB18030				

File name	Server	Client (64 bit)	Client (32 bit)	Description
_64.so	O	O	X	64 bit GB18030 character set conversion library
libgoldilockscvtUHC_32.so	X	X	O	32 bit UHC character set conversion library
libgoldilockscvtUHC_64.so	O	O	X	64 bit UHC character set conversion library
libgoldilocksesql.a	O	O	O	Static library for embedded SQL
libgoldilocksesqls.so	O	O	O	Shared library for embedded SQL
libgoldilockss.so	O	X	X	D/A and C/S-inclusive shared library for ODBC
goldilocks6.jar	O	O	O	C/S-only JDBC library (java 1.6)
goldilocks7.jar	O	O	O	C/S-only JDBC library (java 1.7)
goldilocks8.jar	O	O	O	C/S-only JDBC library (java 1.8)
libgoldilocksjni.so	O	X	X	D/A-only JDBC shared library
libgoldilocksjnigc.so	O	O	O	JDBC shared library for global connection
libgdlc.a	O	O	O	C/S-only static library for ODBC
libgdlcs.so	O	O	O	C/S-only shared library for ODBC

Table 2-11 lib directory (Windows client)

File name	Server	Client (64 bit)	Client (32 bit)	Description
goldilocksc.lib	X	O	O	C/S-only static library for ODBC
goldilockscs.dll	X	X	O	32 bit-C/S-only shared library for ODBC
goldilockscs-ul64.dll	X	O	X	64 bit-C/S-only shared library for ODBC (SQLLEN = 8byte)
goldilockssetup32.dll	X	X	O	32 bit setup library for ODBC
goldilockssetup64.dll	X	O	X	64 bit setup library for ODBC
goldilocksesql.lib	X	O	O	Static library for embedded SQL
goldilocksesqls.dll	X	O	O	Shared library for embedded SQL
goldilockscvtGB18030_32.dll	X	X	O	32 bit GB18030 character set conversion library
goldilockscvtGB18030_64.dll	X	O	X	64 bit GB18030 character set conversion library
goldilockscvtUHC_32.dll	X	X	O	32 bit UHC character set conversion library
goldilockscvtUHC_64.dll	X	O	X	64 bit UHC18030 character set conversion library
goldilocks6.jar	X	O	O	C/S-only JDBC library (for java 1.6)
goldilocks7.jar	X	O	O	C/S-only JDBC library (for java 1.7)
goldilocks8.jar	X	O	O	C/S-only JDBC library (for java 1.8)
gdlc.lib	X	O	O	C/S-only static library for ODBC

File name	Server	Client (64 bit)	Client (32 bit)	Description
gdlcs.lib	X	O	O	C/S-only shared library for ODBC
gdlcs.dll	X	O	O	C/S-only shared library for ODBC
README	X	O	O	read me
goldilocksjnigc.dll	X	O	O	GOLDILOCKS application development library (JDBC D/A mode)

Table 2-12 include directory (Unix)

File name	Server	Client	Description
README	O	O	Read me
sql.h	O	O	ODBC header file
sqlca.h	O	O	ODBC header file
sqlext.h	O	O	ODBC header file
sqltypes.h	O	O	ODBC header file
sqlucode.h	O	O	ODBC header file
goldilocks.h	O	O	Header file for GOLDILOCKS ODBC application development
goldilockstypes.h	O	O	GOLDILOCKS ODBC data type specification file
xa.h	O	O	Standard XA header file
goldilocksxa.h	O	O	GOLDILOCKS XA header file
goldilocksesql.h	O	O	Embedded SQL header file

Table 2-13 include directory (Windows client)

File name	Server	Client	Description
README	X	O	Read me
goldilocks.h	X	O	Header file for GOLDILOCKS ODBC application development
goldilockstypes.h	X	O	GOLDILOCKS ODBC data type specification file
goldilocksxa.h	X	O	GOLDILOCKS XA header file
goldilocksesql.h	X	O	Embedded SQL header file
sqlca.h	X	O	ODBC header file

Table 2-14 license directory

File name	Server	Client	Description
README	O	X	Read me

Table 2-15 msg directory

File name	Server	Client	Description
README	O	O	Read me
goldilocks_error.msg	O	O	Error message file

Table 2-16 conf directory

File name	Description
README	Read me
goldilocks.property.conf	Database operation property text file
goldilocks.listener.conf	Listener property file
goldilocks.invited.conf	Client management file for database connection invited
goldilocks.excluded.conf	Client management file for database connection excluded
goldilocks.gagent.conf	gagent-only configuration file
tablediff.conf	Tablediff configuration file
cyclone.master.conf	Cyclone master only file
cyclone.slave.conf	Cyclone slave only file
logmirror.master.conf	LogMirror master only file
logmirror.slave.conf	LogMirror slave only File
odbc.ini	Template for ODBC configuration
gsql.ini	Template for gsql configuration
glogin.sql	Execution statement list when driving gsql
goldilocks.glocator.conf	gLocator-only configuration file
cyfile.conf	cyfile-only configuration file
clustone.master.conf	clustone master-only default file
clustone.slave.conf	clustone slave-only default file

Table 2-17 db directory

File name	Description
README	Read me

Table 2-18 wal directory

File name	Description
README	Read me

Table 2-19 archive_log directory

File name	Description
README	Read me

Table 2-20 backup directory

File name	Description
README	Read me

Table 2-21 trc directory

File name	Description
README	Read me

Table 2-22 data directory

File name	Description
README	read me

Installing GOLDILOCKS Software

This chapter describes the operating system and the environment setting before installing GOLDILOCKS.

Kernel Parameters

Shared Memory

Shared memory is a type of Inter Process Communication (IPC). It is a memory which is used for sharing data in multiple programs. GOLDILOCKS uses shared memory with user programs using gsql, gloader, ODBC for Client/ Server (C/S) environment. Because all tablespaces for operation are created in shared memory, the precise parameter setting is required.

The followings are parameters and the recommended values required for the shared memory which is used to install GOLDILOCKS.

Table 2-23 Kernal properties for shared memory

Parameter name	Description	Recommended value	Remarks
shmmax	The maximum size of single shared memory segment	The value should be bigger than the size of the biggest datafile.	The value should be set bigger than the size of the biggest datafile belonging to the desired tablespace.
shmmni	The maximum number of shared memory segment available in system	The value should be bigger than the value of which the number of all datafile + 1.	The value should be set bigger than the value of which the number of all datafile + 1 (shared memory segment for SSA).
shmall	The total sum of all shared memory segment (The number of pages)	The value should bebigger than the total sum of tablespace configuration.	It is the total sum of pages in shared memory available in system. Generally, it is used for 8 GB or bigger shared memory. If the total sum of tablespaces in GOLDILOCKS is 32 GB, shmall should be set bigger than it.

The following is an example to set shmall when the total size of tablespaces is 32 GB.

```
kernel.shmmax = 34359738368
kernel.shmmni=4096
kernel.shmall = 8388609
```

- It is assumed that the shmmax is 32 GB and PAGE_SIZE is 4096 bytes.

```
8388609 = (34359738368 / 4096) + 1
```

- In this case, the value of shmall should be bigger than 8388609.

Semaphore

Semaphore is a kind of IPC, like as shared memory, and it is a technology to control multiple processes' behavior using the resources from the operating system. Depending on semaphore setting, multiple processes can simultaneously refer to a relevant resource, and when any process is in use, the other process may wait until it stops using the resource.

GOLDILOCKS uses semaphore to control the access sequence to the shared memory. For example, if multiple GOLDILOCKS client programs request a change to the same data, it should be controlled properly. The semaphore parameter value should be set to an appropriate value according to semaphore operation of GOLDILOCKS. A general Linux value is recommended.

The followings are recommended semaphore values to install GOLDILOCKS.

Table 2-24 Recommended kernel parameter value for semaphore

Kernel parameter	Description	Recommended value
semmsl	The number of semaphores per single semaphore set	250
semmni	The number of semaphore sets	128
semmns	The total sum of semaphore sets (semmni * semmsl)	32000
semopm	The maximum number of semaphores per system call	100

In Linux based system such as Redhat, Ubuntu, if a user creating IPC resource logs out the session list managed by systemd, then the corresponding IPC resource is automatically deleted. Therefore, the system should be set as follows to prevent deleting the semaphore. (kernel 3.0.0 and higher)

```
# cp -i /etc/systemd/logind.conf /etc/systemd/logind.conf_prev
# cat /etc/systemd/logind.conf
[Login]
#NAutoVTs=6
#ReserveVT=6
```

```
...
RemoveIPC=no
```

- Modify it, then execute it.

```
# systemctl restart systemd-logind
```

Network

The backlog means the sockets' queue length waiting to be accepted during the TCP socket listen. Set gls nr's backlog in GOLDILOCKS as glsnr config file's BACKLOG. If the backlog's maximum value in system is bigger than somaxconn, then it sets to somaxconn. In this case, somaxconn should be extended.

GOLDILOCKS uses Unix Domain Socket (UDS) queue when it operates in C/S shared mode. The queue le ngth is set to max_dgram_qlen. If the value is small when clients access the network simultaneously, then it leads to bottleneck state of communication among glsnr, gbalancer and gdispatcher.

The followings are recommended network values to install GOLDILOCKS.

Table 2-25 Recommended kernel parameter value for network

Kernel parameter	Description	Recommended value
somaxconn	The maximum value of listen backlog	1024
max_dgram_qlen	Unix domain socket queue size	256

Applying Parameters

For a one-time execution, a user can do as follows. (It is required to reapply when the user restarts the sys tem).

```
[SHELL]> echo   34359738368   >  /proc/sys/kernel/shmmax
[SHELL]> echo   8388608 >   /proc/sys/kernel/shmall
[SHELL]> echo   4096   >  /proc/sys/kernel/shmmni
[SHELL]> echo   250 32000 100 128  /proc/sys/kernel/sem
[SHELL]> echo   1024   >  /proc/sys/net/core/somaxconn
[SHELL]> echo   256   >  /proc/sys/net/unix/max_dgram_qlen
```

If a user wants to apply it automatically even when the user restarts the system, the user can do as follow s in /etc/sysctl.conf.

```
# shared memory
kernel.shmmax = 34359738368
kernel.shmall = 8388608
kernel.shmmni = 4096
```

```
# semaphore
kernel.sem = 250 32000 100 128
# network
net.core.somaxconn = 1024
net.unix.max_dgram_qlen = 256
```

Use the following command to apply the changes given above.

```
[SHELL]> sysctl  -p
```

Checking Parameters

The described parameters can be checked by using the following commands.

```
[SHELL]> ipcs -l
------ Shared Memory Limits --------
max number of segments = 4096
max seg size (kbytes) = 33554432
max total shared memory (kbytes) = 33554432
min seg size (bytes) = 1
------ Semaphore Limits --------
max number of arrays = 128
max semaphores per array = 250
max semaphores system wide = 32000
max ops per semop call = 100
semaphore max value = 32767
```

Decompressing GOLDILOCKS

GOLDILOCKS package is supplied in a compressed form. The basic installation completes by decompressi on.

The followings are simple examples of how to install the GOLDILOCKS package.

```
##  $GOLDILOCKS_HOME=/home/GOLDILOCKS/goldilocks-mercury.2.1.0-linux-x86_64/
[SHELL]> gzip  - d goldilocks-mercury.2.1.0-linux-x86_64.tar.gz
[SHELL]> tar -xvf goldilocks-server-mercury.2.1.0-linux-x86_64.tar
goldilocks-server-mercury.2.1.0-linux-x86_64/goldilocks_home/include/sqlext.h
goldilocks-server-mercury.2.1.0-linux-x86_64/goldilocks_home/include/goldilocks.h
goldilocks-server-mercury.2.1.0-linux-x86_64/goldilocks_home/include/sqlca.h
...
```

When the decompression completes, a user can change the directory name <package_file_name> on the user's taste, and accordingly the user should change the environment variables of $GOLDILOCKS _HOME and $GOLDILOCKS _DATA.

For more information about directory created by decompression, refer to **GOLDILOCKS Package Configuration**

Setting Enviromment Variables

After decompression of GOLDILOCKS package, bin and lib path will be created under $GOLDILOCKS _HOME directory. Then as given below, a user should add bin and lib path under PATH and LD__LIBRARY_PATH to execute GOLDILOCKS software and develop applications. (When developing GOLDILOCKS client application, a user should always insert $GOLDILOCKS_HOME/include to include file directory of compile option.)

```
export PATH=$GOLDILOCKS _HOME/bin:$PATH
export LD_LIBRARY_PATH=$GOLDILOCKS _HOME/lib:$LD_LIBRARY_PATH
```

Environment variables are required to use GOLDILOCKS. A user should set them prior to installation because some variables are referenced to even during installation.

Table 2-26 OS environment variables for GOLDILOCKS installation

Environment variables	Description	Remarks
GOLDILOCKS_HOME	Directory path to install GOLDILOCKS binaries	This variable is referenced during GOLDILOCKS operation, and the directory to install GOLDILOCKS should be set as an environment variable in advance.
GOLDILOCKS_DATA	The location to create GOLDILOCKS database instance	This variable is referenced during GOLDILOCKS database creation and operation.
PATH	Directory path of GOLDILOCKS executable file	This variable should be set to execute various GOLDILOCKS binaries without an absolute path.
LANG	Character set of terminal	• If the character set is different from the original character set which is created during GOLDILOCKS database creation, characters (Except alphabets, numbers and special characters) may not be displayed properly. Or the string related functions may not be executed correctly. • A user should set locale corresponding to GB18030, SQL_ASCII, UHC, UTF8. • e.g. export LANG=ko_KR.utf8

```
##  $GOLDILOCKS_HOME=/home/GOLDILOCKS/goldilocks-mercury.2.1.0-linux-x86_64/
[SHELL]> gzip  -d goldilocks-mercury.2.1.0-linux-x86_64.tar.gz
```

```
[SHELL]> tar -xvf goldilocks-server-mercury.2.1.0-linux-x86_64.tar
goldilocks-server-mercury.2.1.0-linux-x86_64/goldilocks_home/include/sqlext.h
goldilocks-server-mercury.2.1.0-linux-x86_64/goldilocks_home/include/goldilocks.h
goldilocks-server-mercury.2.1.0-linux-x86_64/goldilocks_home/include/sqlca.h
...
```

Deleting Database

Delete datafile, control file, redo log file and archive log file (when using the archive log file) should be de leted when deleting the existing database to recreate the GOLDILOCKS DATABASE.

The following is an example of deleting GOLDILOCKS DATABASE.

```
## $GOLDILOCKS_BASE=/home/GOLDILOCKS/goldilocks-mercury.2.1.0-linux-x86_64/
## $GOLDILOCKS_DATA=/home/GOLDILOCKS/goldilocks-mercury.2.1.0-linux-x86_64/
                    goldilocks_data/
## $GOLDILOCKS_HOME=/home/GOLDILOCKS/goldilocks-mercury.2.1.0-linux-x86_64/
                    goldilocks_home/
[SHELL]> rm -rf $GOLDILOCKS_DATA/db/*.dbf
[SHELL]> rm -rf $GOLDILOCKS_DATA/wal/*.ctl
[SHELL]> rm -rf $GOLDILOCKS_DATA/wal/*.log
[SHELL]> rm -rf $GOLDILOCKS_DATA/archive_log/*.log
```

The location of the data file and archive file may vary depending on the settings by a user.

Deletion

GOLDILOCKS package is not provided in compressed file format, so a specific deletion rule is not required. Delete the installed directory after terminating DATABASE.

The following is an example of deleting GOLDILOCKS package.

```
## $GOLDILOCKS_BASE=/home/GOLDILOCKS/goldilocks-mercury.2.1.0-linux-x86_64/
## $GOLDILOCKS_DATA=/home/GOLDILOCKS/goldilocks-mercury.2.1.0-linux-x86_64/
                    goldilocks_data/
## $GOLDILOCKS_HOME=/home/GOLDILOCKS/goldilocks-mercury.2.1.0-linux-x86_64/
                    goldilocks_home/
[SHELL]> rm -rf $GOLDILOCKS_DATA
[SHELL]> rm -rf $GOLDILOCKS_HOME
[SHELL]> rm -rf $GOLDILOCKS_BASE
## ~/.bash_profile
$GOLDILOCKS_BASE=/home/GOLDILOCKS/goldilocks-mercury.2.1.0-linux-x86_64/ ❶ Delete
```

```
$GOLDILOCKS_DATA=/home/GOLDILOCKS/goldilocks-mercury.2.1.0-linux-x86_64/
                    goldilocks_data/ ❷ Delete
$GOLDILOCKS_HOME=/home/GOLDILOCKS/goldilocks-mercury.2.1.0-linux-x86_64/
                    goldilocks_home/ ❸ Delete
```

Creating Database

Create database after the completion of property creation. A user can create database by using $GOLDILOCKS_HOME/bin/gcreatedb. The followings are how to use the gcreatedb commands.

```
[SHELL]> gcreatedb --help
Usage
    gcreatedb [options]
Options:
    --cluster        cluster system (if not specified, stand-alone system)
    --db_name        database name
    --db_comment     database comment
    --timezone       timezone ( {+/-}{TZH:TZM} )
    --character_set  character set
                       SQL_ASCII
                       UTF8
                       UHC
                       GB18030
    --char_length_units  char length units
                          OCTETS
                          CHARACTERS
    --member         local cluster member name
    --host           cluster ip address or host name
    --port           cluster port number
    --silent         suppresses the display of the result message
    --help           print help message
examples:
    gcreatedb --db_name="goldilocks" --db_comment="goldilocks database" --timezone="+09:00"
--character_set="UTF8" --char_length_units="OCTETS" --silent
```

$GOLDILOCKS_HOME/conf/goldilocks.properties.conf is referenced when creating database. Then, table space files are created in *SYSTEM_TABLESPACE_DIR* path in *goldilocks.properties.conf* with the value of ****_TABLESPACE_SIZE*.

--cluster option should be specified when creating the database which is to participate in cluster system.

The followings are execution arguments of gcreatedb.

Table 2-27 Execution arguments of gcreatedb

Argument	Description
--cluster	It is the database to be used in cluster system. If it is omitted, standalone database is created.
--db_name	It is the database name. If it is omitted, it is set as goldilocks.
--db_comment	It is the database description. If it is omitted, it is set as goldilocks database.
--timezone	It is the timezone. If it is omitted, it is set as TIMEZONE property.
--character_set	It is the database character set. GOLDILOCKS supports four types of character sets. • GB18030: Simplified Chinese • SQL_ASCII: Character set supporting ASCII • UHC: Unified Hangul Code • UTF8: Unicode Transformation Format - 8 If it is omitted, it is set as CHARACTER_SET property.
--char_length_units	It is the unit of character length. • OCTETS: It identifies 1 byte as 1 character • CHARACTERS: It identifies 1 character (n byte) as 1 character. If it is omitted, it is set as CHAR_LENGTH_UNITS property.
--member	It is the member name local database to be used in cluster system. If it is omitted, it uses the value set in LOCAL_CLUSTER_MEMBER property.
--host	It is the host name or the IP address of local member to be used for communication between cluster system members. If the host name is used, then it uses the first IPv4 address of the system. If it is omitted, it uses the value set in LOCAL_CLUSTER_MEMBER_HOST property.
--port	It is the TCP listen port of local member to be used for communication between cluster system members. If it is omitted, it uses the value set in LOCAL_CLUSTER_MEMBER_PORT property.
--silent	It hides display messages.
--help	It displays help messages.

A user should consider the followings when creating database.

- Setting kernel parameter shared memory
  - If the size specified in shared memory setting is smaller than the size described in $GOLDILOCKS_HOME/conf/goldilocks.properties.conf, a user can not create the database.
- Tablespace size
  - Tablespace files are created when the database is created, and the tablespace is used after being allocated to the memory as big as the tablespace size when driving GOLDILOCKS. Therefore, it us

es memory even when the actual user data does not exist in the data tablespace. Therefore, a user should create database with the sufficient memory considering the actually available memory as well as shared memory on GOLDILOCKS startup machine when writing goldilocks.properties.conf.

- Whether to use cluster system
  - It should be specified that whether the database will participatte as a member of cluster system when creating the database. In other words, --cluster option should be omitted when executing gcreatedb using the database as standalone, but --cluster option should be specified when using the database as cluster system.

When database is created successfully, a user can check the following tablespace file in the path described in SYSTEM_TABLESPACE_DIR of goldilocks.properties.conf.

```
[SHELL]> gcreatedb
Database created
[SHELL]> gcreatedb --db_name="TEST_DB"                   \
                   --db_commnet="test database comment"  \
                   --timezone="+09:00"                   \
                   --character_set="UHC"                 \
                   --char_length_units="OCTETS"
Database created
[SHELL]> ls $GOLDILOCKS_DATA/db
system_data.dbf  system_dict.dbf  system_undo.dbf
```

The following is an example of creating database to be used in cluster system.

```
[SHELL]> gcreatedb --cluster
Database created
[SHELL]> gcreatedb --cluster --member=G1N1
Database created
[SHELL]> gcreatedb --cluster --member=G1N1 --host=127.0.0.1 --port 10101
Database created
[SHELL]> gcreatedb --cluster                          \
                   --db_name="TEST_DB"                \
                   --home=$GOLDILOCKS_DATA            \
                   --host=127.0.0.1                   \
                   --port=10101                       \
                   --db_comment="g1n1 db comment"     \
                   --timezone="+09:00"                \
                   --character_set="UHC"              \
                   --char_length_units="OCTETS"
Database created
```

```
[SHELL]> ls $GOLDILOCKS_DATA/db
system_data.dbf  system_dict.dbf  system_trans.dbf system_undo.dbf
```

Building Dictionary Schema Information

Create the following schema to get the system and object information.

A user should build the following schema after database creation. Otherwise, there is a possibility of malfunction in Catalog API of ODBC, JDBC for obtaining the object's structure information (for example, SQLTables() function). If so, it will not interlock with third party tools.

- DICTIONARY_SCHEMA: It consists of views and tables to get object information such as DBA_*, ALL_*, USER_* .
- INFORMATION_SCHEMA: It consists of views and tables included in the SQL standard INFORMATION_SCHEMA.
- PERFORMANCE_VIEW_SCHEMA: It consists of views to get system information by combining the fixed tables' information.

Views and tables included in each schemas provides convenience to get system information.

After driving GOLDILOCKS instance on OPEN phase, a user should execute the sql files as follows. It should be done at least once after the first database creation.

The scripts to build dictionary schema information are divided into the script for standalone system and the script for cluster system. Therefore, a user should use the appropriate script for the purpose to build the information.

The following describes how to build the information by using the script for standalone database.

```
% gsql --as sysdba --import $GOLDILOCKS_HOME/admin/standalone/DictionarySchema.sql
% gsql --as sysdba --import $GOLDILOCKS_HOME/admin/standalone/InformationSchema.sql
% gsql --as sysdba --import $GOLDILOCKS_HOME/admin/standalone/PerformanceViewSchema.sql
```

The following describes how to build the information by using the script for cluster system.

```
% gsql --as sysdba --import $GOLDILOCKS_HOME/admin/cluster/DictionarySchema.sql
% gsql --as sysdba --import $GOLDILOCKS_HOME/admin/cluster/InformationSchema.sql
% gsql --as sysdba --import $GOLDILOCKS_HOME/admin/cluster/PerformanceViewSchema.sql
```

2.3 Managing Database Memory Structure

This chapter describes the database components which compose GOLDILOCKS instance.

Database Memory Structure

GOLDILOCKS database is divided into memory area and disk area. Memory area is a collection of tablespa ce consisting of one or more shared memories. GOLDILOCKS database never goes down to the disk by re place operation.

Disk area consists of data files, control file, property file, online redo log files. Data file exists one per shar ed memory of each tablespace, and control file contains instance configuration information. property file stores instance environment settings, and online redo log file is used to recover database.

Control File

Control file records the physically stored information on the disk of database, and determines the status o f database by firstly reading at the beginning of Instance startup. The control file records the following inf ormation.

- Instance state at the time of the previous checkpoint
- Transaction durability mode (CDS/TDS) at the time of the previous startup
- Online redo log file information
- Data file information of each tablespace

Online Redo Log File

Online redo log files store all changes made to the database by transactions in instances. It is used to reco ver unwritten changes on the data file when restarting instance after database's abnormal termination. F our online redo log files are generated by default in the size specified in LOG_FILE_SIZE property when cr eating database. A user can add more if the user need. The redo log files are reused in circulation manner.

Checkpoint are generated when online redo log file switches to the next file, and then some dirty pages move down to an appropriate data file. If the checkpoint operation is delayed and the updated page doe s not move down (ACTIVE state), all transactions will be suspended until checkpoint completion. Therefor e, creating a suitable size online redo log file according to the application's characteristic is helpful to imp rove the database system performance.

Undo Segment

Undo segments records images in advance of changing operation to use when transactions partially or to tally rollback. A single undo segment is assigned to a single transaction during update operation. Undo se gments are stored in MEM_UNDO_TBS tablespace. It is recommended to secure enough undo tablespace, in preparation for multiple update transactions or a single transaction with large amount of update opera tion (bulk delete).

Data File

Data file includes the contents of tables/indexes stored in tablespace.

Data file consists of the followings.

- Page
  - The minimum unit of database I/O. The current size is 8 Kbytes.
- Extent
  - A certain number of continuous pages' collection. The minimum unit of which the segment is all ocated space from the tablespace.
  - Extent size of each tablespace could be different.
- Segment
  - A specific type of data structure's collection. A segment consists of extent sets.

Exceptionally, a temporary tablespace, such as MEM_TEMP_TBS tablespace, does not execute redo loggi ng, nor does data file create.

Tablespace

Database is divided into tablespaces which is a logical structure containing tables and indexes. GOLDILOC KS tablespace is classified into the memory tablespace and the disk tablespace. In the memory tablespace, a separate disk I/O does not occur when the shared memory is created per each data file in the tablespac e and accesses to the page. However, in the disk tablespace, the buffer cache of the system is used to acc ess the page. The information about the tablespace existing in the current database can be output by retr ieving V$TABLESPACE table.

GOLDILOCKS supports the following tablespaces by default.

Table 2-28 Tablespaces of GOLDILOCKS

Owner	Name	Description
	DICTIONARY_TBS	Default dictionary tables are stored in this tabespace to operate database.
	MEM_UNDO_TBS	Undo segments and transaction information are stored in this table space.
	MEM_DATA_TBS	If a user does not specify a tablespace when creating schema object, the data ta ble is stored in this tablespace by default.

Owner	Name	Description
SYSTEM	DISK_DATA_TBS	It is the disk tablespace which the user uses by default.
	MEM_TEMP_TBS	Indexes which does not specified tablespace name, and temporary tables which are used by queries are created in the tablespace. Indexes are rebuilt when restarting instance because logging does not occur.
	MEM_TRANS_TBS	It is used to recover global transaction in cluster system. It is created only when it is configured as cluster system.
USER	User-defined	User defines this tablespace to collect specific tables to a specific tablespace and manage them.

Tablespace Types

There are five types of tablespace as follows.

- DICT
  - Tablespace type for storing dictionary tables and indexes.
- DATA
  - Tablespace type for storing general schema objects such as tables and indexes.
  - It becomes the subject of redo logging and page flushing.
- UNDO
  - Tablespace type for storing undo segments.
  - It becomes the subject of redo logging and page flushing.
- TEMPORARY
  - Tablespace type for storing the temporary tables and no-logging indexes which are created during SELECT query.
  - It is not the subject of redo logging and page flushing.
- TRANSACTION (Cluster only)
  - Tablespace type which is used to recover global transaction in cluster system

Checking Information of Database Storage Structure

This chapter describes a method to check the information about multiple database storage structure which are mentioned above.

Control File Information

Use gdump utility to check the contents because control file is stored in binary format.

```
[SHELL]> gdump CONTROL control_0.ctl
```

Online Redo Log File Information

Retrieve the control file by using gdump tool, then the name and current state of each online redo file wil l be displayed.

Data File Information

Retrieve the control file by using gdump tool, then the data files in each tablespaces and its states will be displayed. Also, viewing the V$DATAFILE table, their current states will be displayed.

Tablespace Information

Retrieve the control file by using gdump tool, then the name and state of each tablespace in the current database will be displayed. Also, a user can enquire the V$TABLESPACE table by using SQL.

Property Information

Open the text file $GOLDILOCKS_Data/conf/goldilocks.properties.conf, then the property information wil l be displayed. When working online, retrieve V$PROPERTY, V$DB_PROPERTY tables, then the informatio n about currently applied property values will be displayed.

General Operation of Data Storage

A tablespace stores data, and its operation is as follows.

Creating Tablespace

The memory and the disk USER DATA tablespaces are created as follows.

```
gSQL> CREATE TABLESPACE TEST_TBS DATAFILE 'TEST_TBS.dbf' SIZE 10M;
Tablespace created.
gSQL> CREATE MEMORY TABLESPACE TEST_TBS DATAFILE 'TEST_MEM_TBS.dbf' SIZE 10M;
Tablespace created.
gSQL> CREATE DISK TABLESPACE TEST_TBS DATAFILE 'TEST_DISK_TBS.dbf' SIZE 10M;
Tablespace altered.
gSQL> CREATE DISK TABLESPACE DISK_TBS DATAFILE 'TEST_DISK_TBS.dbf' SIZE 10M AUTOEXTEND OFF;
Tablespace altered.
```

TEMPORARY tablespace does not include data file, so it is created as follows.

```
gSQL> CREATE TEMPORARY TABLESPACE TEST_TEMP_TBS MEMORY 'TEST_TEMP_TBS' SIZE 10M;
Tablespace created.
```

Retrieving Tablespace Usage State

A tablespaces space is allocated or deallocated in the unit of one extent consisting of one or more consec utive pages. A user can retrieve the size of one extent (BYTE) in a specific tablespace as follows.

```
gSQL> SELECT EXTENT_SIZE FROM V$TABLESPACE WHERE TBS_NAME = 'TEST_TBS';
EXTENT_SIZE
-----------
     262144
1 row selected.
```

A user can view the state of all extents in tablespaces by using D$TABLESPACE_EXTENT table. When an e xtent is in use, the STATE column is 'U'. When an extent is in free state, the STATE column is 'F'. Therefor e, the remaining size of space in the current tablespace (the number of extents) can be calculated as follo ws.

```
gSQL> SELECT COUNT(*) FROM D$TABLESPACE_EXTENT('TEST_TBS') WHERE STATE = 'F';
COUNT(*)
--------
      38
1 row selected.
```

Seeing the result above, the empty space in TEST_TBS is 38 * 262144 = 9437184 Byte.

Altering Tablespace

A user can alter the tablespaces by using Add/Remove Data File (Memory in case of temporary tablespace s), and Online/Offline.

Add/ Drop Data File (or Memory)

If a user wants to add spaces to tablespaces while operating the database, the DATA tablespaces allocate the additional space by using the following syntax.

```
gSQL> ALTER TABLESPACE TEST_TBS ADD DATAFILE 'TEST_TBS2.dbf' SIZE 10M;
Tablespace altered.
```

TEMPORARY tablespace adds spaces as follows. Unlike DATA tablespace, a name should be given, and th e name should be a unique memory name in database.

```
gSQL> ALTER TABLESPACE TEST_TEMP_TBS ADD MEMORY 'TEST_TEMP_TBS2' SIZE 10M;
Tablespace altered.
```

A user can drop the space in DATA tablespace by using the following syntax. However, if any part of the area is used, the user can not drop it.

```
gSQL> ALTER TABLESPACE TEST_TBS DROP DATAFILE 'TEST_TBS2.dbf';
Tablespace altered.
```

Similarly, a user can withdraw the space in TEMPORARY tablespace by using the following syntax.

```
gSQL> ALTER TABLESPACE TEST_TEMP_TBS DROP MEMORY 'TEST_TEMP_TBS2';
Tablespace altered.
```

Offline Tablespace

Switch the tablespace to offline mode if a user wants to move the location of data file in the tablespace. Use the following syntax.

```
gSQL> ALTER TABLESPACE TEST_TBS OFFLINE;
Tablespace altered.
```

A user can switch the tablespace to online mode again by using the following syntax.

```
gSQL> ALTER TABLESPACE TEST_TBS ONLINE;
Tablespace altered.
```

Altering Automatic Data File Expand Property in Disk Tablespace

The data file in the disk tablespace is created in the initial size, and it is automatically extended to the max imum size when it is needed. The automatic expand property can be on or off as follows. The automatic expand size and the maximum size of the data file can be altered when altering the automatic expand pr operty.

```
gSQL> ALTER DATABASE DATAFILE 'TEST_DISK_TBS.dbf' AUTOEXTEND ON;
Database altered.
gSQL> ALTER DATABASE DATAFILE 'TEST_DISK_TBS.dbf' AUTOEXTEND OFF;
Database altered.
gSQL> ALTER DATABASE DATAFILE 'TEST_DISK_TBS.dbf' AUTOEXTEND ON NEXT 10M MAXSIZE 20M;
Database altered.
```

Rename

Rename the tablespace by using the following syntax.

```
gSQL> ALTER TABLESPACE TEST_TBS RENAME TO TEST_TBS2;
Tablespace altered.
```

Switch the tablespace to offiline mode, if a user wants to change the location of data file in the tablespac e. Then, move the data file by using OS command, rename it by using ALTER TABLESPACE statement, the n switch the tablespace to online state.

```
gSQL> ALTER TABLESPACE TEST_TBS OFFLINE;
Tablespace altered.
gSQL> ALTER TABLESPACE TEST_TBS RENAME DATAFILE 'TEST_TBS.dbf' TO 'TEST_TBS_1.dbf';
ERR-42000(16164): file does not exist :
ALTER TABLESPACE TEST_TBS RENAME DATAFILE 'TEST_TBS.dbf' TO 'TEST_TBS_1.dbf'
                                                                        *
ERROR at line 1:
```

- The following procedure describes how to rename the file by using OS command.

```
gSQL> ALTER TABLESPACE TEST_TBS RENAME DATAFILE 'TEST_TBS.dbf' TO 'TEST_TBS_1.dbf';
Tablespace altered.
gSQL> ALTER TABLESPACE TEST_TBS ONLINE;
Tablespace altered.
```

Dropping Tablespace

Drop an unnecessary tablespace by using the following syntax. The statement after INCLUDING is optiona l, but if the statement is given then it will delete all the content (schema object) and data file in the tables paces.

```
gSQL> DROP TABLESPACE TEST_TBS INCLUDING CONTENTS AND DATAFILES;
Tablespace dropped.
```

Store Mode

GOLDILOCKS uses the store mode as an instance unit to maximize performance under certain circumstan ce. Store mode defines which part of ACID property to give up to improve the operation performance in t he transaction.

GOLDILOCKS supports two types of store modes.

- Transactional Data Store (TDS) mode
  - TDS mode is default store mode of GOLDILOCKS database.
  - Transactional Data Store (TDS) mode is general DBMS store mode. In this mode, all transactions i n the instance write both undo log and redo log. Therefore, a user can rollback the transactions a nd recover using data file and online redo log file even when an instance is terminated abnormall y, because periodical checkpoint are performed.
- Concurrent Data Store (CDS) mode
  - All running transactions in the instance write undo logs, but do not write redo logs. Therefore, th e transactions can deal with all run-time errors, and can rollback. But if an instance is terminated abnormally or terminated using shutdown abort statement, all updated data would be lost. (It do es not provide recover facility). It is because checkpoint action does not occur. CDS Mode control s concurrency among transactions, so it ensures normal operations of transactions even when dif ferent transactions access the same object at the same time.
  - CDC Mode is mainly used in run-time information oriented database in which query/update oper ations are frequently performed but does not require durability such as cache server.

Store mode is set through the property setting when a user starts up the instance. Transactions can not b e executed in different store modes. The user should carefully set the store mode when the user starts up the instance because the user can not change the instance in online mode.

2.4 Managing Schema Object

Schema Object

Schema object is a set of logical structure created by a user. GOLDILOCKS supports the following schema objects, which are table, index, synonym, view, sequence, constraint and stored procedure.

Schema Object Management Privileges

Currently, GOLDILOCKS supports user and his privilege. Therefore, not all the users share all the created o bjects, so the privilege should be given to a user.

Managing Table

This chapter describes table overview, methods of retrieving the table information, creating/ altering tabl e and loading/ dropping data.

Table

A table is the most basic unit of storage containing user data. A table consists of columns and rows.

Table Type

Currently, GOLDILOCKS supports general heap table whose data saving order is irrelevant to sort order of a particular column. However, GOLDILOCKS does not support clustered table, partitioned table.

- It supports basic heap table only.
- It supports primary key/unique/not null constraint.
- It supports the datatypes as follows.
  - BOOLEAN
  - SMALLINT, INTEGER, BIGINT, REAL, DOUBLE, NUERMIC, FLOAT
  - CHAR (MAX 2000), VARCHAR (MAX 4000), BINARY, VARBINARY, LONG VARCHAR, LONG VA RBINARY
  - DATE, TIME, TIMESTAMP, INTERVAL (It supports WITH/WITHOUT TIMEZONE of TIME/TIMEST AMP.)
  - It does not support BLOB type.

- The number of columns, indexes and constraints are not limited.
- A user can retrieve the entire table information using a query *SELECT * FROM TABLES;*.

If a huge number of rows are stored in a particular table, and then the table size becomes big, even bulk delete operation does not return the table's empty space to the tablespace. But TRUNCATE operation can return all existing space to the tablespace.

Table data can be stored and retrieved being distributed to multiple nodes according to the user's desirin g distribution policy when using GOLDILOCKS configuring it as cluster system. For more information, refe r to **Managing Table**

Managing Index

This chapter describes index overview and creating/deleting index.

Overview

Index is a subsidiary schema object which is linked to tables. A user can easily find the location of specific ally conditioned row using the index. A user can also retrieve the row's column value if the column is the key column of the index.

GOLDILOCKS can create as many indexes in need to tables. However, too many indexes burden the exec ution of inserting/ changing/ deleting operation of the table, then it may lower the performance.

Primary key or unique constraint automatically creates an index on that column.

Index Property

- It supports B-link tree form index.
  - GOLDILOCKS provides B-link tree form index by default.
- The size of a single index node is 8 Kbytes.
- The maximum number of key columns is 32, and the maximum key length is 2000 bytes.
- It supports unique index.
- It supports Ascending/Descending, NULLS FIRST and NULLS LAST.
  - A user can define whether to sort index key columns in ascending order (ASC) or in descending (DESC) order.
  - A user can define whether to list NULL value of the index key column at first (NULLS FIRST) or at last (NULLS LAST).
- A user can retrieve the whole index information by using the query *SELECT * FROM INDEXES;*.
- A user can calculate the index size in the similar way of calculating table because index is implemente d by segment in the same way of implementing table.

Sequence

Sequence is a schema object which generates a unique number.

A user can generate the sequence as follows.

```
gSQL> CREATE SEQUENCE customers_seq START WITH 1000 INCREMENT BY 1 NOCACHE NOCYCLE;
Sequence created.
```

A user can use the sequence by using NEXTVAL.

```
gSQL> SELECT customers_seq.NEXTVAL FROM dual;
```

A user can drop the sequence as follows.

```
gSQL> DROP SEQUENCE customers_seq;
```

Global sequence object is automatically created when user creates sequence by using GOLDILOCKS confi guring cluster system. This object manages the global pool of the sequence value which is used being sha red by all member nodes in cluster system. Each member node is allocated the sequence value as many as specified from the global object when calling NEXTVAL and uses them.
For more information, refer to **Global Sequence**.

2.5 Managing User

Creating User

Only the SYS user and the user with the CREATE USER ON DATABASE privilege can create a user for GOL DILOCKS database. The CREATE SESSION ON DATABASE privilege is required to connect to the newly cre ated user.

The followings are the syntax to create a user.

```
<user definition> ::=
    CREATE USER user_identifier IDENTIFIED BY password
    [ DEFAULT TABLESPACE tablespace_name ]
    [ TEMPORARY TABLESPACE tablespace_name ]
```

The followings are the syntax rules and parameters to create a user.

- user_identifier
  - It is a username to be created.
  - The username and the role name should be unique.
  - The length of a user_identifier should be smaller than 128 bytes.
- password
  - The password is stored encrypted.
  - The length of a password should be smaller than 128 bytes.
  - Password is case-sensitive.
- DEFAULT TABLESPACE tablespace_name
  - It specifies the default tablespace which stores the objects such as tables, indexes (LOGGING) cre ated by a user which is to be created.
  - If DEFAULT TABLESPACE clause is omitted, it specifies as the default data tablespace (MEM_D ATA_TBS) which is defined when creating DATABASE (<database definition>).
- TEMPORARY TABLESPACE tablespace_name
  - It specifies the TABLESPACE to store user-created temporary tables, index (NO LOGGING), and q uery processing-generated intermediate results.
  - If TEMPORARY TABLESPACE clause is omitted, it is specified as the default temporary tablespace (MEM_TEMP_TBS) which is defined when creating DATABASE (<database definition>).
- INDEX TABLESPACE tablespace_name
  - It specifies the default tablespaces to store an index object created by a user to be created.
  - If INDEX TABLESPACE clause is omitted, the default value is INDEX TABLESPACE NULL.

Dropping User

It drops the created user of GOLDILOCKS database. An access privilege and user range for dropping user are as same as those for creating user.

The following is the syntax to drop a user.

```
<drop user statement> ::=
    DROP USER [ IF EXISTS ] user_identifier [ <drop behavior> ]
    ;
<drop behavior> ::=
      RESTRICT
    | CASCADE
```

The followings are the syntax rules and parameters for dropping a user.

- IF EXISTS
  - Even when the user does not exist, an error does not occur.
- user_identifier
  - It is a database username to be dropped.
  - A user can not drop a user, such as SYS, which is automatically generated when the database is created.
  - It does not drop objects, such as tablespace, which is not the owner but created by user_identifier.
- drop behavior
  - If drop behavior is omitted, the default value is RESTRICT.

Altering User

It alters the definition of the GOLDILOCKS database user. ALTER USER privilege is required for an ordinary user. However, if a user is the user_identifier user, then the user can alter the definition without the privilege.

The followings are the syntax to alter the user definition.

```
<alter user statement> ::=
      ALTER USER user_identifier <alter user action>
    | ALTER USER PUBLIC <alter schema path>
    ;
<alter user action> ::=
```

```
      <alter password>
    ¦ <alter profile>
    ¦ <alter default tablespace>
    ¦ <alter temporary tablespace>
    ¦ <alter index tablespace>
    ¦ <alter schema path>
<alter password> ::=
    IDENTIFIED BY new_password [ REPLACE old_password ]
<alter profile> ::=
    PROFILE { profile_name ¦ DEFAULT ¦ NULL }
<password expire> ::=
    PASSWORD EXPIRE
<account lock> ::=
    ACCOUNT { LOCK ¦ UNLOCK }
<alter default tablespace> ::=
    DEFAULT TABLESPACE tablespace_name
<alter temporary tablespace> ::=
    TEMPORARY TABLESPACE tablespace_name
<alter index tablespace> ::=
    INDEX TABLESPACE { tablespace_name ¦ NULL }
<alter schema path> ::=
    SCHEMA PATH ( { schema_name ¦ CURRENT PATH } [, ...] )
```

The followings are the syntax rule and parameters for altering user.

- user_identifier
  - It is a username to be altered.
- <alter password>
  - It alters user password.
  - IDENTIFIED BY new_password
    - It changes the current password, and the new password is stored encrypted.
    - The length of a password should be smaller than 128 bytes.
    - Password is case-sensitive.
  - REPLACE old_password
    - If a user have ALTER USER ON DATABASE privilege, the user can omit it.
    - If a user do not have ALTER USER ON DATABASE privilege, the user can not omit it.
    - The user and user_identifier should be same.
- <alter profile>
  - It alters the profile for password management policies.
  - PROFILE profile_name
    - It allocates profile_name created by a user.

    - PROFILE DEFAULT
        - It allocates "DEFAULT" which is the default profile.
    - PROFILE NULL
        - It does not allocate the profile.
- 〈password expire〉
    - It expires the user password.
- 〈account lock〉
    - ACCOUNT LOCK
        - It locks the user account.
    - ACCOUNT UNLOCK
        - It unlocks the account lock.
- 〈alter default tablespace〉
    - It alters user's default tablespace.
    - tablespace_name should be data tablespace.
- 〈alter temporary tablespace〉
    - It alters user's temporary tablespace.
    - tablespace_name should be temporary tablespace.
- 〈alter index tablespace〉
    - It alters the index tablespace of a user.
    - It assigns INDEX TABLESPACE tablespace_name
        - When assigning the data tablespace, it becomes the LOGGING index.
        - When assigning the temporary tablespace, it becomes the NOLOGGING index.
    - INDEX TABLESPACE NULL
        - It does not assign the index tablespace.

2.6 GOLDILOCKS Property

GOLDILOCKS property is classified as the property which is applied when creating database and the property which can be updated at online/offline. A user can change the tablespace path and the redo log file only at MOUNT phase of startup.

Properties When Creating Database

Table 2-29 Properties when creating database

Name	Description
SYSTEM_MEMORY_DICT_TABLESPACE_SIZE	Initial size of dictionary tablespace
SYSTEM_MEMORY_DATA_TABLESPACE_SIZE	Initial size of system data tablespace
SYSTEM_MEMORY_UNDO_TABLESPACE_SIZE	Initial size of system undo tablespace
SYSTEM_MEMORY_TEMP_TABLESPACE_SIZE	Initial size of system temporary tablespace
LOG_BLOCK_SIZE	Block size of redo log file
LOG_FILE_SIZE	Initial size of redo log file
LOG_GROUP_COUNT	The number of redo log files
CHARACTER_SET	Character set
TIMEZONE	Time zone
CHAR_LENGTH_UNITS	Character length unit

Properties When Driving Database

There are more than 100 properties in GOLDILOCKS database. The followings are frequently used properties among them.

Table 2-30 Frequently used properties when driving database

Name	Description
SHARED_MEMORY_STATIC_KEY	Key value to create the shared memory
SHARED_MEMORY_STATIC_SIZE	Size of the shared memory
DATA_STORE_MODE	Storage mode of GOLDILOCKS instance
LOG_BUFFER_SIZE	Log buffer size
LOG_DIR	Directory path of redo log
PRIVATE_STATIC_AREA_SIZE	Static area size per session
CLIENT_MAX_COUNT	The maximum number of accessible session
PROCESS_MAX_COUNT	The maximum number of process

Name	Description
NET_BUFFER_SIZE	Network buffer size per session

2.7 GOLDILOCKS Utility

gcreatedb

The gcreatedb utility initializes GOLDILOCKS database and gets ready for the service. The gcreatedb gene rates data files and log files in the GOLDILOCKS_DATA environment variable location according to given properties. The following is a syntax.

```
[SHELL]> gcreatedb --help
Usage
    gcreatedb [options]
Options:
    --cluster               cluster system (if not specified, stand-alone system)
    --db_name               database name
    --db_comment            database comment
    --timezone              timezone ( {+/-}{TZH:TZM} )
    --character_set         character set
                              SQL_ASCII
                              UTF8
                              UHC
                              GB18030
    --char_length_units     char length units
                              OCTETS
                              CHARACTERS
    --member                local cluster member name
    --host                  cluster ip address or host name
    --port                  cluster port number
    --silent                suppresses the display of the result message
    --help                  print help message
examples:
    gcreatedb --db_name="goldilocks" --db_comment="goldilocks database" --timezone="+09:00"
--character_set="UTF8" --char_length_units="OCTETS" --silent
```

Tablespace files are created in *SYSTEM_TABLESPACE_DIR* of goldilocks.properties.conf as each value of * **_TABLESPACE_SIZE by referring to $GOLDILOCKS_HOME/conf/goldilocks.properties.conf when creati ng the database.

The followings are execution arguments of gcreatedb command.

Table 2-31 Execution arguments of gcreatedb

Argument	Description
--cluster	It is the database to be used in cluster system. If it is omitted, standalone database is created.
--db_name	It is the database name. If it is omitted, it is set as *goldilocks*.
--db_comment	It is the database description. If it is omitted, it is set as *goldilocks database*.
--timezone	It is the timezone. If it is omitted, it is set as TIMEZONE property.
--character_set	It is the database character set. GOLDILOCKS supports four types of character sets. • GB18030: Simplified Chinese • SQL_ASCII: Character set supporting ASCII • UHC: Unified Hangul Code • UTF8: Unicode Transformation Format - 8 If it is omitted, it is set as CHARACTER_SET property.
--char_length_units	It is the unit of character length. • OCTETS: It identifies 1 byte as 1 character. • CHARACTERS: It identifies 1 character (n byte) as 1 character. If it is omitted, it is set as CHAR_LENGTH_UNITS property.
--home	It is the database home directory. It searches for a property file, and is referenced to as a location of creating and storing various DB files. If it is omitted, it uses the value set in GOLDILOCKS_DATA environment variable.
--member	It is the member name of local database to be used in cluster system. If it is omitted, it uses the value set in LOCAL_CLUSTER_MEMBER property.
--host	It is the host name or the IP address of local member to be used for communication between cluster system members. If the host name is used, then it uses the first IPv4 address of the system. If it is omitted, it uses the value set in LOCAL_CLUSTER_MEMBER_HOST property.
--port	It is the TCP listen port of local member to be used for communication between cluster system members. If it is omitted, it uses the value set in LOCAL_CLUSTER_MEMBER_PORT property.
--silent	It hides display messages.
--help	It displays help messages.

gsql (GOLDILOCKS Interactive SQL Tool)

gsql is an interactive command line utility to execute SQL statements for managing GOLDILOCKS databas e. DBA creates initial table schema by using gsql, or checks the current database state.

The following is the gsql syntax.

```
[SHELL]> gsql <userid> <passwd>
gSQL> CREATE TABLE T1 ( COL1 INTEGER );
create success
gSQL> \q
[SHELL]>
```

gloader (GOLDILOCKS Data Upload/download Tool)

The gloader utility downloads existing data in database to a file in text format, or it uploads an existing d ata in text format to a new database. The text data file format of gloader is Comma-Separated Value (CS V).

The following is how to use gloader.

```
gloader [export¦import] userid/passwd control='control_file_name' \
data='data_file_name' log='log_file_name' bad='bad_file_name'
```

Table 2-32 Arguments of gloader

Argument	Description
export	import	It declares whether to downlad the contents of existing table to data_file_name, or to upload existing data in data_file_name to a specified tabe in control_file_name.
userid	It specifies user ID.
passwd	It specifies the password of the userid.
control	It specifies the file path in which detailed settings are written during export/import op eration.
data	It specifies target data file to export, or data file to import.
log	It specifies log file path in which the progress and the elapsed time of import/export o peration.
bad	It specifies bad file path which records data records failed in insertion at importing du e to various errors.

The following is an example of a control file.

```
TABLE  TEST_TBL
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
```

3.

Cluster Tutorial

3.1 Managing GOLDILOCKS Cluster System

This chapter describes the basic information for configuring and managing the cluster system by using multiple GOLDILOCKS databases. Only the parts added to the cluster system or distinguishing the cluster system comparing to the standalone database are described in this chapter, so it is prerequisite to read the standalone database tutorial before read this chapter.

Overview

GOLDILOCKS can be used by configuring a standalone database as described in the previous chapter. Or, a user can also bind multiple databases into a single cluster, then select an appropriate solution for data distribution. In other words, a user can customize how to distribute massive data to multiple servers when using GOLDILOCKS cluster system. This guarantees high availability and improves the throughput due to the parallel processing.

GOLDILOCKS cluster system consists of one or more cluster groups, and a cluster group consists of one or more cluster members. It does not require an extra application server or a meta server. Applications are operated accessing to cluster members corresponding to the data server. Cluster members belonging to the same cluster group keeps the same data replication.

A user should determine whether to use GOLDILOCKS database as standalone system or as cluster system, when creating database of each node. To use it as cluster system, a user should add options related to cluster when creating database of each node.

Property Setting

Properties to build cluster system are described in *$GOLDILOCKS_DATA/conf/goldilocks.property.conf* file of each server as like when using standalone database. Main properties for TBS (tablespace), LOG, CONTROL FILE can be set as same as setting in standalone system even when using cluster system. However, there should not be the same path or port of files among cluster members when creating multiple databases to configure cluster system in a single server.

The followings describe main property items of when configuring cluster system.

Table 3-1 Main property items

Property	Description	Default value
	It is the directory path of installing the following system TBS.	

Property	Description	Default value
SYSTEM_TABLESPACE_DIR	• DICTIONARY_TBS • MEM_DATA_TBS • MEM_UNDO_TBS • MEM_TEMP_TBS • MEM_TRANS_TBS	'<GOLDILOCKS_DATA>/db'
SYSTEM_MEMORY_DICT_TABLESPACE_SIZE	It is the dictionary tablespace size.	256M
SYSTEM_MEMORY_DATA_TABLESPACE_SIZE	It is the data tablespace size.	200M
SYSTEM_MEMORY_UNDO_TABLESPACE_SIZE	It is the undo tablespace size.	32M
LOG_DIR	It is the default log directory path.	'<GOLDILOCKS_DATA>/wal'
SYSTEM_LOGGER_DIR	It is the system log directory path.	'<GOLDILOCKS_DATA>/trc'
CONTROL_FILE_COUNT	It is the number of control files.	2
CONTROL_FILE_0	It is the first control file path.	'<GOLDILOCKS_DATA>/wal/control_0.ctl'
CONTROL_FILE_1	It is the second control file path.	'<GOLDILOCKS_DATA>/wal/control_1.ctl'
LOCAL_CLUSTER_MEMBER	It is the name of cluster member of which local server uses in cluster system.	'G1N1'
LOCAL_CLUSTER_MEMBER_HOST	It is the host name of local server.	'127.0.0.1'
LOCAL_CLUSTER_MEMBER_PORT	It is TCP listen port of which local server uses for the communication in cluster system.	10101

The name of cluster member and host-port combination should be unique in cluster system. To omit the property setting above, provide information by using *--member, --host, --port* options when creating database using gcreatedb. The member information provided by property or gcreatedb option is stored and managed in *$GOLDILOCKS_DATA/wal/location.ctl* file.

To modify the properties, update the text property file (*$GOLDILOCKS_DATA/conf/goldilocks.properties.conf*), or define a new variable in a form of *GOLDILOCKS_<property_name>* in an environment variable. The property file is prior to the environment variable.

Background Process

GOLDILOCKS cluster system has the background process (gmaster) to manage instances per each member node. gmaster of each node consists of multiple system threads internally. Most of them are as same as those in standalone system, but the following system threads are added to manage cluster system.

The following threads are performed only when starting the database created as cluster mode.

- Cluster recover thread: It recovers global transaction in cluster system.
- Failover thread: It deals with the failover through reselecting offiline and coordinator for the member s when an error occurs on a specific node or in a network in cluster system.

Two following processes are additionally driven when using GOLDILOCKS as cluster system.

- cdispatcher: It transmit/receive cluster packet and manages sessions.
- cserver: It performs operation of modifying and querying for the database to which the member nod e belongs.

GOLDILOCKS cluster system requires complex cluster protocol communication among member nodes, an d cluster dispatcher (cdispatcher) is a process to efficiently perform the management of network commu nication context and packet distribution mechanism. In addition, it performs monitoring continuously the validity of cluster session through heartbeat.

In cluster system, SQL performing in a specific node (driver node) needs to store data on the remote mem ber node by referring to the sharding strategy of the target table or enquires the data stored in the corres ponding node. In this case, a process is required to process this request and return the result in each me mber node, and that process is cserver.

Client Process

A user can use both client server (C/S) model and direct access (D/A) model in cluster system as like stand alone system. However, each member node of cluster system has its own listener, so the client program s hould know the listen port of the cluster member to access beforehand (in C/S mode). A user can process various transactions when accessing to any node of cluster system as like standalone database.

Signal handling, cleanup for connection, releasing shared resources in cluster system are processes as sa me as those in standalone system.

Memory Structure of Instance

GOLDILOCKS cluster system is used to bind multiple shared-nothing databases to a management unit wh ich is a single cluster system. Therefore, a method of using memory of cluster member nodes is similar to that of standalone. The memory size of which each member node uses are determined by properties set i n its own property. Static area contains instance basic information for database management, informatio n for each session, statement, transaction, redo log buffer, dictionary cache, and other operational infor mation as like standalone system. Additionally, information for management of cluster session such as inf

ormation for location and cluster session are also stored.

Tablespace area consists of page frames and page control header (PCH). The page frame contains the co ntents of each tablespace and PCH controls those page frames.

The application process memory contains instance memories attached when connecting, ODBC environm ent shared in process unit, various ODBC handles and heap memory area containing other information su ch as bind information.

Start and End of Cluster System

To start GOLDILOCKS system, create instance on each member node beforehand (gcreatedb), and then r egister cluster group and cluster member. Later, a user can start or end the system by using sysdba role th rough gsql and gsqlnet.

listener should be in operation if a user wants to start or end GOLDILOCKS cluster system in dedicated m ode of C/S model(to use gsqlnet).

GOLDILOCKS cluster system can not be started or ended in the shared mode of C/S model.

```
% gsql --as sysdba
Enter user-name: sys
Enter password:
Connected to an idle instance.
gSQL>
```

GOLDILOCKS cluster system has the following startup phases. OPEN phase is subdivided into LOCAL OPE N and GLOBAL OPEN unlike the standalone system.

- NOMOUNT
  - It launches gmaster which is a demon to manage GOLDILOCKS instance.
- MOUNT
  - It reads properties, and the control file for the recovery by using $GOLDILOCKS_DATA environme nt variable.
- LOCAL OPEN
  - It loads tablespace content from data files, then performs recovery by using the redo log files, an d recovers in-doubt global transaction, newly build no-logging indexes, and creates dictionary ca ches.
- OPEN (GLOBAL OPEN)
  - It connects cluster sessions of each cluster member, arranges shard map, selects a global manage r (global coordinator) and a group manager (group coordinator), then waits for the user to acces s the service.

To start or end the GOLDILOCKS cluster system the cluster system environment should be configured perf orming the following preliminary works. If the member name, host address, port number are uniquely giv en when creating each member database by using gcreatedb, the property setting process for each node can be omitted.

- Setting property: Update property files to be used on each member node.
- Creating database: Create the database through gcreatedb on each member node.
- Creating cluster group and adding a member: Create a group and a member configuring cluster syste m.
- Driving listener: The listner on each node should be driven beforehand to use gsqlnet.

Use the following syntaxes to create a cluster group or a member.

- **ALTER CLUSTER GROUP name ADD MEMBER**
- **CREATE CLUSTER GROUP**

• Creating database: It is performed on each node.

```
% gcreatedb --cluster --db_name='goldilocks' --member='g1n1' \
    --host='192.168.0.11' --port 10110
% gcreatedb --cluster --db_name='goldilocks' --member='g1n2' \
    --host='192.168.0.12' --port 10120
```

• Creating a cluster group and a member: It is performed on a single node. The startup phase of the clus ter member to be created or added should be GLOBAL OPEN.

```
gSQL> create cluster group g1 cluster member g1n1
        host '192.168.0.11' port 10110;
gSQL> alter cluster group g1 add cluster member g1n2
        host '192.168.0.12' port 10120;
```

If the configuration of GOLDILOCKS cluster system is completed as above, the entire cluster system can b e started or ended by using the following two methods.

- Accessing to each member node to start or end nodes one by one
  - Use \startup and \shutdown commands.
- Start or end the entire member node at once in a single member node
  - Connect through gsqlnet, and then use \cstartup and \cshutdown commands.

The following describes the first method above which is how to access each member node then drive clus ter system. \startup command is used to directly enter into LOCAL OPEN phase without the intermediate phase. This can be operated being subdividing into three phases, which are \startup nomount, alter syste m mount database, and alter system open local database.

Drive up to LOCAL OPEN phase on each node through \startup, and then access a single node and drive up to GLOBAL OPEN phase.

- Startup up to LOCAL OPEN: It is performed on each member node.

```
% gsql sys gliese --as sysdba
  gSQL> \startup
  Startup success.
```

- Startup up to OPEN: It is performed on a single node.

```
% gsql sys gliese --as sysdba
  gSQL> alter system open global database;
  System altered.
```

Starting up with the first method can be a burden to an operator if many nodes are included in cluster system. It is because the entire process from end of cluster system to LOCAL OPEN should be performed everytime on every member node. The following is a simple method to startup the entire member for the ease of operation.

- Startup to GLOBAL OPEN: It is performed on a single node.

```
% gsqlnet sys gliese --as sysdba
gSQL> \cstartup
Startup success.
```

- \cstartup and \cshutdown commands can be performed only in gsqlnet, and it is not supported in gsql. Also, listener should be driven beforehand on every member node to be driven.

- To build GOLDILOCKS cluster system containing multiple members on a single physical node, home directory name, member name and the cluster port information should not be duplicated when creating each database.

When GOLDILOCKS system ends, gmaster, the management daemon process, ends on each member node, so it does not allow any more connection or other database operations.

ABORT is the only shutdown mode available when terminating only the single node in GOLDILOCKS cluster system. ABORT mode immediately terminates gmaster regardless of the state of connected sessions, and unloads the instance.

If \shutdown command is performed by using gsql as follows, only the connected member node is termin ated.

```
% gsql sys gliese --as sysdba
Connected to GOLDILOCKS Database.
gSQL> \shutdown abort
Shutdown success
gSQL>
```

To terminate the entire member belonging to cluster system at once, use \cshutdown command of gsqln et as follows. The normal option can be omitted.

```
% gsqlnet sys gliese --as sysdba
Connected to GOLDILOCKS Database.
gSQL> \cshutdown normal
Shutdown success
gSQL>
```

Using abort option when performing \cshutdown, it terminates the entire member node by forc e. Therefore, some member nodes may fail to join cluster system when restarting it by using \cs tartup. Although the failed nodes can join through the following join commands and rebalance process, it is recommended to terminate it by using \cshutdown normal which is a safe method.

To terminate and start a member node or some member nodes belonging to cluster system, follow the pr ocess below. The following describes how to restart only the G1N2 node among cluster member nodes t hen make it join cluster system.

```
% gsql sys gliese --as sysdba --dsn=g1n2
Connected to GOLDILOCKS Database.
gSQL> \shutdown abort
Shutdown success
gSQL> \startup
Startup success
gSQL> alter system join database;
System altered.
```

To restart a member node and make it rejoin cluster system, rebalancing operation for the altered table may be required if the corresponding node is terminated when a transaction occurs. It the rebalancing op eration is not performed, the transaction performing on a driver node may fail to alter the table on which the rebalaning is not performed.

A rebalancing operation is a process which synchronizes data distribution policy and property information of table among member nodes, and dividedly stores table data again in each member nodes according to the data distribution policy.

```
% gsql sys gliese --as sysdba
Connected to GOLDILOCKS Database.
gSQL> \shutdown abort
Shutdown success
gSQL> \startup
Startup success
gSQL> alter system join database;
ERR-42000(16405): some tables in the database need to be rebalanced
System altered.
gSQL> alter database rebalance;
Database altered.
```

3.2 Installing GOLDILOCKS and Creating Database

Installing and creating member database which is to be included in GOLDILOCKS cluster system is almost as same as those in standalone system. This chapter describes only the unique feature and method used f or installing and creating database of cluster system comparing to standalone system.

Configuring GOLDILOCKS Package

The package used to configure GOLDILOCKS cluster system is as same as those of standalone database. However, the scripts to build dictionary and performance view are divided into the script for standalone s ystem and the script for cluster system. Therefore, a user should use the appropriate script for the purpos e to build the information after creating the database.

The script for standalone database is located below the *$GOLDILOCKS_HOME/admin/standalone* director y and the script for cluster system is located below the *$GOLDILOCKS_HOME/admin/cluster* directory.

Table 3-2 admin/ standalone directory

File name	Description
README	Read me
DictionarySchema.sql	It is the dictionary schema creation script.
InformationSchema.sql	It is the information schema creation script.
PerformanceViewSchema.sql	It is the PerformanceView schema creation script.

Table 3-3 admin/ cluster directory

File name	Description
README	Read me
DictionarySchema.sql	It is the dictionary schema creation script.
InformationSchema.sql	It is the information schema creation script.
PerformanceViewSchema.sql	It is the PerformanceView schema creation script.

Installing GOLDILOCKS Software

GOLDILOCKS software should be installed on every member node belonging to cluster system, and the in stalling method is as same as that of standalone system. The method for setting and checking kernel para meter, setting environment variable are as same as those of standalone system, so refer to the correspon ding tutorial.

In this case, be cautious that setting properties for each cluster member node, database name, database version, character set, time zone should be same for the normal operation of cluster system.

Creating Database

Use gcreatedb utility to create database on each member node configuring cluster system as like standalone system.

The following options of gcreatedb are used only when creating cluster database. Other options are as same as standalone system.

Table 3-4 Execution arguments of gcreatedb

Argument	Description
--cluster	It represents that it is cluster database. When it is omitted, standalone database is created.
--member	The member name of local database to be used in cluster system. When it is omitted, it uses the value set in LOCAL_CLUSTER_MEMBER property.
--host	IP address of local member to be used for communication between cluster system members. If it is omitted, it uses the value set in LOCAL_CLUSTER_MEMBER_HOST property.
--port	TCP listen port of local member to be used for communication between cluster system members If it is omitted, it uses the value set in LOCAL_CLUSTER_MEMBER_PORT property.

The following is an example of creating database to be used in cluster system.

```
[SHELL]> gcreatedb --cluster
Database created
[SHELL]> gcreatedb --cluster --member=G1N1
Database created
[SHELL]> gcreatedb --cluster --member=G1N1 --host=127.0.0.1 --port 10101
Database created
[SHELL]> gcreatedb --cluster                        \
                   --db_name="TEST_DB"              \
                   --home=$GOLDILOCKS_DATA          \
                   --host=127.0.0.1                 \
                   --port=10101                     \
                   --db_comment="g1n1 db comment"   \
                   --timezone="+09:00"              \
                   --character_set="UHC"            \
                   --char_length_units="OCTETS"
```

```
Database created
[SHELL]> ls $GOLDILOCKS_DATA/db
system_data.dbf  system_dict.dbf  system_trans.dbf system_undo.dbf
```

Building Dictionary Schema Information

To normally use cluster system, the dictionary schema information should be built as like standalone syste m. If the following schema is not built, the catalog API (e.g. SQLTables() function) of ODBC, JDBC obtaini ng object's structure information malfunctions, then it can not interwork with the third party tools. In con clusion, it should be built after creating database.

It is recommended to build schema in cluster system after completing the operation of creating a cluster group and a member. It is because GOLDILOCKS automatically creates schema on every member node w hen performing creation script by connecting to a member node after completing cluster system configur ation.

- DICTIONARY_SCHEMA: It consists of tables and views inquiring object information such as DBA_*, A LL_*, USER_*.
- INFORMATION_SCHEMA: It consists of tables and views included in the SQL standard INFORMATION _SCHEMA.
- PERFORMANCE_VIEW_SCHEMA: It consists of views inquiring system information combining inform ation of fixed tables.

The followings describe how to build schema by using the script for cluster system. Perform the operation by accessing to a single member node in GLOBAL OPEN phase as described above.

```
% gsql sys gliese --as sysdba --import $GOLDILOCKS_HOME/admin/cluster/DictionarySchema.sql
% gsql sys gliese --as sysdba --import $GOLDILOCKS_HOME/admin/cluster/InformationSchema.sql
% gsql sys gliese --as sysdba --import
$GOLDILOCKS_HOME/admin/cluster/PerformanceViewSchema.sql
```

Various structure information can be viewed by using the schema information built above. What is differe nt from standalone system is that a user can extract only the desired node information by giving group na me and member node after the object when inquiring the information.

```
% gsql test test
Connected to GOLDILOCKS Database.
gSQL> select origin_member_name, stat_name, stat_value
    2   from gv$system_mem_stat
    3  where stat_name = 'PLAN_CACHE_TOTAL_SIZE';
ORIGIN_MEMBER_NAME STAT_NAME             STAT_VALUE
```

```
------------------ ---------------------- ----------
G1N1               PLAN_CACHE_TOTAL_SIZE   17844952
G2N2               PLAN_CACHE_TOTAL_SIZE   16796288
G2N1               PLAN_CACHE_TOTAL_SIZE   16796288
G1N2               PLAN_CACHE_TOTAL_SIZE   16796288
G3N1               PLAN_CACHE_TOTAL_SIZE   16796288
G3N2               PLAN_CACHE_TOTAL_SIZE   16796288
6 rows selected.
gSQL> select origin_member_name, stat_name, stat_value
    2   from gv$system_mem_stat@g1n2
    3  where stat_name = 'PLAN_CACHE_TOTAL_SIZE';
ORIGIN_MEMBER_NAME STAT_NAME             STAT_VALUE
------------------ ---------------------- ----------
G1N2               PLAN_CACHE_TOTAL_SIZE   16796288
1 row selected.
gSQL> select origin_member_name, stat_name, stat_value
    2   from gv$system_mem_stat@g1
    3  where stat_name = 'PLAN_CACHE_TOTAL_SIZE';
ORIGIN_MEMBER_NAME STAT_NAME             STAT_VALUE
------------------ ---------------------- ----------
G1N1               PLAN_CACHE_TOTAL_SIZE   17844952
G1N2               PLAN_CACHE_TOTAL_SIZE   16796288
2 row selected.
```

3.3 Managing Schema Object

The schema object is a logical structure created by a user. GOLDILOCKS cluster system supports the schema objects such as table, index and global sequence. This chapter describes features of each schema object in cluster system comparing to standalone system. How to efficiently manage them is also described.

Managing Table

A user can specifies one of four following sharding strategies as an option when creating a table in GOLDILOCKS cluster system. Sharding strategy is how to distribute and store table data to each cluster group in cluster system. This option can be specified only in cluster system, and it can not be used when database is created as standalone.

- Cloned strategy
  - It equally copies entire table data.
- Hash sharding strategy
  - It distributes table data based on the hash value of sharding key.
- Range sharding strategy
  - It distributes table data based on the range value of sharding key.
- List sharding strategy
  - It distributes table data based on the list value of sharding key.

For more information about creating table, refer to **CREATE TABLE** and **Cluster Table and Shard**.

When omitting the sharding strategy option, it is defined by **DEFAULT_SHARDING** property. The default value of DEFAULT_SHARDING is 0, and it creates cloned table.

```
CREATE TABLE t1
(
    id   INTEGER,
    name VARCHAR(128)
);
```

If a table is created without giving the sharding strategy by as user as above, then GOLDILOCKS cluster system internally creates a table by using the following syntax.

```
CREATE TABLE t1
(
    id   INTEGER,
```

```
    name VARCHAR(128)
)
    CLONED
    AT CLUSTER WIDE
;
```

Hash/Range/List sharded table should comply with the followings when creating constraints.
- PRIMARY KEY, UNIQUE constraint should include the sharding key.

Cloned Strategy

It does not distribute the table data based on a specific condition, but it copies all data. The target of cop ying data can be all nodes in cluster system, or it can be a specific cluster group. In other words, it arrang es clones in cluster members of a defined cluster group.

- AT CLUSTER WIDE
  - It arranges clones in all cluster members of all cluster group in cluster system.
  - When adding a cluster group or cluster member, a user can rearranges clones by using **ALTER TA BLE name REBALANCE** statement.
- AT CLUSTER GROUP group_list
  - It arranges clones in all cluster members of a defines cluster group.
  - When adding a cluster member to a defined cluster group, a user can rearranges clones by using **ALTER TABLE name REBALANCE** statement.
  - Adding cluster group does not affect the rearrangement of clones.

When the option is omitted, the default value is automatically set to *AT CLUSTER WIDE*.

```
CREATE TABLE t1
(
    id   INTEGER,
    name VARCHAR(128)
)
CLONED
AT CLUSTER WIDE;
```

```
CREATE TABLE t1
(
    id   INTEGER,
    name VARCHAR(128)
)
CLONED
```

```
AT CLUSTER GROUP G1, G2;
```

Hash Sharding Strategy

It distributes the table data based on the hash value of a column defined as a sharding key.

To use hash sharding strategy, sharing key should be defined complying with the following conditions.

- Up to 32 columns can be listed.
- The duplicated column can not be used.
- The column of LONG VARCHAR or LONG VARBINARY type can not be used.

```
CREATE TABLE t1 ( id INTEGER, name VARCHAR(128) )
    SHARDING BY HASH(id);
```

When hash sharding related options are omitted as above, GOLDILOCKS system interprets it as follows.

```
CREATE TABLE t1 ( id INTEGER, name VARCHAR(128) )
    SHARDING BY HASH(id)
    SHARD COUNT 24
    AT CLUSTER WIDE;
```

Table rows are distributes to one of 24 shards based on hash value of id column, and 24 shards are equall y arranged over the entire cluster system. For more information, refer to **Cluster Table and Shard**.

Range Sharding Strategy

It distributes the table data based on the range value of the column defined as a sharding key. The shards are classified based on each range value, and they can be arranged by defining a specific group, or arrang ed as CLUSTER WIDE.

The following is a syntax of defining six range shards based on the range value of the sharding key colum n, and creating a table to distribute them as CLUSTER WIDE. If a cluster group is created after creating a t able, the rearrangement of shard containing the created group can be performed by using REBALANCE f eature.

- Create a range sharded table.
- Arrange six shards in existing cluster groups (g1, g2, g3).

```
CREATE TABLE t1
(
   id   INTEGER,
   name VARCHAR(32)
```

```
)
SHARDING BY RANGE (id)
    AT CLUSTER WIDE
    SHARD s1 VALUES LESS THAN ( 200000 ),
    SHARD s2 VALUES LESS THAN ( 400000 ),
    SHARD s3 VALUES LESS THAN ( 500000 ),
    SHARD s4 VALUES LESS THAN ( 600000 ),
    SHARD s5 VALUES LESS THAN ( 800000 ),
    SHARD s6 VALUES LESS THAN ( MAXVALUE )
;
```

- Add a cluster group.

```
CREATE CLUSTER GROUP g4
       CLUSTER MEMBER g4n1 HOST '192.168.0.41' PORT 10401
;
```

- Rearrange the range shard.
- Rearrange six shards in cluster groups (g1, g2, g3, g4) in which the added group is included.

```
ALTER TABLE t1 REBALANCE;
```

The following is a syntax of defining range shards, which are defined based on the range of sharding key column value, only to be arranged in a specific cluster group. In other words, the shard s1 whose range v alue is smaller than 200000 is allocated in cluster group g1, s2 is allocated in cluster group g2, and shard s3 is allocated in g3. Like as shards can not be rearranged by using REBALANCE feature even when a clus ter group is added later in a table created by defining cluster group.

- Create a range sharded table.
- Allocate each range shards to defined cluster groups.

```
CREATE TABLE t1
(
   id   INTEGER,
   name VARCHAR(32)
)
SHARDING BY RANGE (id)
    SHARD s1 VALUES LESS THAN ( 200000 )   AT CLUSTER GROUP g1,
    SHARD s2 VALUES LESS THAN ( 400000 )   AT CLUSTER GROUP g2,
    SHARD s3 VALUES LESS THAN ( 500000 )   AT CLUSTER GROUP g3,
    SHARD s4 VALUES LESS THAN ( 600000 )   AT CLUSTER GROUP g2,
    SHARD s5 VALUES LESS THAN ( 800000 )   AT CLUSTER GROUP g3,
```

```
    SHARD s6 VALUES LESS THAN ( MAXVALUE ) AT CLUSTER GROUP g1
;
```

- Add a cluster group.

```
CREATE CLUSTER GROUP g4
       CLUSTER MEMBER g4n1 HOST '192.168.0.41' PORT 10401
;
```

- Rearrange the range shard.
- A shard is not allocated to the newly created cluster group (g4).

```
ALTER TABLE t1 REBALANCE;
```

The following conditions should be considered to specify the range sharding key.

- Up to 32 columns can be listed.
- The duplicated column can not be used.
- The column of LONG VARCHAR or LONG VARBINARY type can not be used.

List Sharding Strategy

It distributes the table data based on the list value of the column defined as a sharding key. Like as the ra nge sharding strategy, each shard can be arranged by defining a specific group, or arranged as CLUSTER WIDE.

The followings are conditions to define the shard key in list sharding strategy.

- Only a single column can be used.
- The column of LONG VARCHAR or LONG VARBINARY type can not be used.

The following is a syntax of creating a table of which created list shards are arranged as CLUSTER WIDE. Use REBALANCE feature to rearrange shards including a cluster group created after creating a table.

- Create a list sharded table.
- Arrange five shards in existing cluster groups (g1, g2, g3).

```
CREATE TABLE city
(
   id   INTEGER,
   name VARCHAR(32)
)
SHARDING BY LIST (name)
```

```
    AT CLUSTER WIDE
    SHARD s1 VALUES IN ( 'SEOUL' ),
    SHARD s2 VALUES IN ( 'PUSAN', 'ULSAN', 'DAEGU' ),
    SHARD s3 VALUES IN ( 'DAEJEON', 'GWANGJU' ),
    SHARD s4 VALUES IN ( 'ANSAN', 'GOYANG' ),
    SHARD s5 VALUES IN ( DEFAULT )
;
```

- Add a cluster group.

```
CREATE CLUSTER GROUP g4
       CLUSTER MEMBER g4n1 HOST '192.168.0.41' PORT 10401
;
```

- Rearrange the list shard.
- Rearrange five shards in cluster groups (g1, g2, g3, g4) in which the created group is included.

```
ALTER TABLE city REBALANCE;
```

The following is a syntax of defining list shards, which are defined based on the list value of sharding key column value, only to be arranged in a specific cluster group. Shards can not be rearranged by using REB ALANCE feature even when a cluster group is created.

- Create a list sharded table.
- Allocate each range shards to defined cluster groups.

```
CREATE TABLE city
(
   id   INTEGER,
   name VARCHAR(32)
)
SHARDING BY LIST (name)
    SHARD s1 VALUES IN ( 'SEOUL' )                  AT CLUSTER GROUP g1,
    SHARD s2 VALUES IN ( 'PUSAN', 'ULSAN', 'DAEGU' ) AT CLUSTER GROUP g2,
    SHARD s3 VALUES IN ( 'DAEJEON', 'GWANGJU' )     AT CLUSTER GROUP g3,
    SHARD s4 VALUES IN ( 'ANSAN', 'GOYANG' )        AT CLUSTER GROUP g2,
    SHARD s5 VALUES IN ( DEFAULT )                  AT CLUSTER GROUP g1
;
```

- Create a cluster group.

```
CREATE CLUSTER GROUP g4
       CLUSTER MEMBER g4n1 HOST '192.168.0.41' PORT 10401
;
```

- Rearrange the list shard.
- A shard is not allocated to the newly created cluster group (g4).

```
ALTER TABLE city REBALANCE;
```

Managing Index

Global Secondary Index

In cluster system, multiple member nodes exist, and table records are dividedly stored or duplicated based on the shard strategy. Standalone system guarantees the uniqueness of a record by storing unique value (Row Identifier: RID) in the database when the record is stored. However, in cluster system, each node ca n have the duplicated value, so the uniqueness can not be guaranteed.

Therefore, it is required to guarantee the uniqueness of a record in cluster system. This is a reason why gl obal RID(GRID) is added. The GRID value of a record is not updated even when the record is updated, the n it guarantees the uniqueness of a specific record in cluster system.

Global secondary index is a B-tree index which consists of keys to search the GRID value of the records qu ickly in cluster system.

A use can select whether to create global secondary index through the following properties when creatin g a table. The user also can delete or recreate the global secondary index after table creation is complete d. Only one global secondary index can be created per table.
For more information, refer to **DEFAULT_GLOBAL_SECONDARY_INDEX_CREATION**.

A global secondary index is necessary to perform the non-deterministic query for a table. If a global secon dary index does not exist in a table, then a non-deterministic query fails as follows.

```
gSQL> DELETE FROM T1 LIMIT 1;
ERR-42000(16423): does not support non-deterministic DML in the cluster system : global
secondary index expected
```

Enquire USER_GSI_PLACE DICTIONARY, or use ALL_GSI_PLACE and DBA_GSI_PLACE dictionary to check if the global secondary index of a table is created.

```
gSQL> CREATE TABLE T1( I1 INTEGER );
Table created.
gSQL> COMMIT;
Commit complete.
gSQL> SELECT *
    2   FROM USER_GSI_PLACE@LOCAL
    3  WHERE TABLE_NAME = 'T1';
TABLE_SCHEMA TABLE_NAME GROUP_ID GROUP_NAME MEMBER_ID MEMBER_NAME MEMBER_OFFLINE
------------ ---------- -------- ---------- --------- ----------- --------------
BLOCKS
------
PUBLIC       T1                1 G1                 1 G1N1        FALSE
    64
PUBLIC       T1                1 G1                 2 G1N2        FALSE
  null
2 rows selected.
gSQL> DROP TABLE T1;
Table dropped.
gSQL> COMMIT;
Commit complete.
gSQL> SELECT *
    2   FROM USER_GSI_PLACE@LOCAL
    3  WHERE TABLE_NAME = 'T1';
no rows selected.
```

Global Sequence

GOLDILOCKS cluster system provides global sequence object which are an expansion of the existing sequence for multiple member nodes to share and use the set of sequence value fitting into user defined conditions. In other words, the global sequence object is internally and automatically created when a user creates a sequence in cluster system, then the sequence values in a specific range are allocated and used when calling NEXTVAL on each member node. The following is a syntax of creating and using the global sequence, which are as same as those of sequence in standalone.

```
gSQL> CREATE SEQUENCE global_user_seq START WITH 1000 INCREMENT BY 1 NOCACHE NOCYCLE;
Sequence created.
gSQL> SELECT global_user_seq.NEXTVAL FROM dual;
NEXTVAL
-------
```

```
      1
1 row selected.
gSQL> DROP SEQUENCE global_user_seq;
Sequence dropped.
```

Like as the sequence used in standalone, the global sequence object can define a cache size as a creating option, and this means acquiring several sequence value from the global sequence object and loading at l ocal cache. The followings are local cache status of each member node and return values of when calling NEXTVAL, this is when creating the global sequence object by setting cache size to 5.

```
gSQL> CREATE SEQUENCE seq START WITH 1 CACHE 5;
Sequence created.
```

Member name	NEXTVAL result	The number of remaining local cache sequence		Description
		G1N1	G1N2	
G1N1	1	4	0	From Global Object (Alloc 1 ~ 5)
G1N1	2	3	0	From Local Cache
G1N1	3	2	0	From Local Cache
G1N2	6	3	4	From Global Object (Alloc 6 ~ 10)
G1N2	7	3	3	From Local Cache
G1N2	8	3	2	From Local Cache
G1N2	9	3	1	From Local Cache
G1N2	10	3	0	From Local Cache
G1N2	11	3	4	From Global Object (Alloc 11 ~ 15)
G1N2	12	3	3	From Local Cache
G1N1	4	1	3	From Local Cache
G1N1	5	0	3	From Local Cache
G1N1	16	4	3	From Global Object (Alloc 16 ~ 20)

In the table above, The sequence values are allocated to G1N1 and G1N2 two times each (four times in t otal) from the global sequence object. The cache option value is set to 5 when creating a sequence, so fiv e sequence values are allocated each from the global object. Those five sequences allocated to a member node is stored in its own local cache, then it is returned one by one whenever calling NEXTVAL.

- G1N1
  - The values of five sequences (1~5) are allocated from the global sequence object when calling th e first NEXTVAL.

    - A single sequence is returned as a result of NEXTVAL and the values of other four sequences are l oaded in its own local cache.
    - NEXTVAL called later returns in turn the values of the sequences loaded in the local cache.
- G1N2
    - The values of sequences (6~10) which are after the sequences allocated on G1N1 are allocated when calling the first NEXTVAL.
    - 6 is returned as a result of NEXTVAL, and the values of other four sequences (7~10) are loaded i n its own local cache.
    - NEXTVAL called later returns the sequence values loaded in the local cache.
    - Local cache is run out, so values of other five sequences (11~15) are allocated again from the glo bal sequence object.
- G1N1
    - Local cache is run out, so the values of five sequences (16~20) after the sequences allocated last on G1N2 are allocated.

The sequence as big as the CACHE size are allocated to all member nodes when NEXTVAL has never bee n called on its own node or when all allocated nodes are run out. Therefore, if a system needs to acquire the sequence value quickly, it is required to set the appropriate cache size when creating a sequence to pr event too much frequent allocation. It is because, unlike standalone database, allocating sequence from t he global sequence object is accompanied by the network communication cost.

The sequence values loaded in local cache can not be reused when database restarts due to a system erro r or operational work. The sequence values are allocated again from the global sequence object when cal ling NEXTVAL for the first time since the restart. Therefore, the CACHE size allocated when creating the s equence also means the range value of sequence which is possible to be lost when an error occurs, so th e size should be set appropriately considering not only the corresponding feature but also loss range.

The result value of calling NEXTVAL from a specific node may not be sequential in cluster system. In the e xample above, the return value is not sequential when G1N1 node calls NEXTVAL. After the first allocate d value(1~5) is run out, 16~20 is allocated second, so 16 is returned to a user as the next sequence value of 5. It is because a single global sequence pool is allocated competitively to multiple nodes.

The followings are features and constraints of global sequence object, comparing to the sequence for exi sting standalone database.

- The sign can not be modified by using INCREMENT BY option in ALTER SEQUENCE statement. (The si ze can be modified.)
- The duplicated value among member nodes can be returned depending on the pool size of entire seq uence when using CYCLE option. Therefore, create the sequence pool big enough, considering INCR EMENT BY, CACHE SIZE, and the number of cluster member nodes when CYCLE option is in need.
- The return value from a specific node may not be sequential even when an error does not occur. Of c ourse, if only one member node calls NEXTVAL, then sequential sequence value is obtained.
- The CACHE SIZE is 1 for NOCACHE, so the additional sequence values are not loaded in local cache. I n this case, only one sequence is allocated from the global sequence object whenever calling NEXTV

AL, and it increases the network cost and degrades the performance.

- Creating, updating and deleting sequences are operated as AUTO COMMIT.
- All sequence values loaded in local cache of all nodes are reset when the size of CACHE and INCREM ENT are updated by ALTER statement. In other words, a new sequence set should be allocated from t he global sequence object when calling NEXTVAL afterwards.

3.4 GOLDILOCKS Property

The followings are main properties used in GOLDILOCKS cluster system.

Table 3-5 Main properties of GOLDILOCKS

Name	Description
LOCAL_CLUSTER_MEMBER	Member name
LOCAL_CLUSTER_MEMBER_HOST	Host address for connecting to cluster session
LOCAL_CLUSTER_MEMBER_PORT	Port for connecting to cluster session
CDISPATCHER_THREADS	The number of cluster dispatcher threads
CSERVERS	The number of cluster server process
CLUSTER_DATA_SYNC_SERVERS	The number of cluster server process to synchronize replica data

Each property has the following two alterable scopes in GOLDILOCKS cluster system.

- LOCAL: It can alter property values not only for entire member but also for a specific member only.
- GLOBAL: It can not alter the property value only for a specific member, but it should alter the propert y value for all members.

For example, PRIVATE_STATIC_AREA_SIZE properties can be altered up to the LOCAL range (IS_GLOBAL column = FALSE) as follows, so the property value can be altered by using *alter system set* statement not only for entire member but also for a specific member only. If AT clause is not added to after *alter system set* statement, the altered properties are applied to all members of cluster system.

```
% gsql test test
Connected to GOLDILOCKS Database.
gSQL> select origin_member_name, property_name, property_value, is_global
    2   from gv$property
    3  where property_name = 'PRIVATE_STATIC_AREA_SIZE';
ORIGIN_MEMBER_NAME PROPERTY_NAME            PROPERTY_VALUE IS_GLOBAL
------------------ ------------------------ -------------- ---------
G1N1               PRIVATE_STATIC_AREA_SIZE 104857600      FALSE
G1N2               PRIVATE_STATIC_AREA_SIZE 104857600      FALSE
2 rows selected.
gSQL> alter system set private_static_area_size = 200000000 at g1n2;
System altered.
gSQL> select origin_member_name, property_name, property_value, is_global
    2   from gv$property
    3  where property_name = 'PRIVATE_STATIC_AREA_SIZE';
ORIGIN_MEMBER_NAME PROPERTY_NAME            PROPERTY_VALUE IS_GLOBAL
```

```
------------------ ------------------------ -------------- ---------
G1N1               PRIVATE_STATIC_AREA_SIZE 104857600      FALSE
G1N2               PRIVATE_STATIC_AREA_SIZE 200000000      FALSE
2 rows selected.
gSQL> alter system set private_static_area_size = 300000000;
System altered.
gSQL> select origin_member_name, property_name, property_value, is_global
    2   from gv$property
    3  where property_name = 'PRIVATE_STATIC_AREA_SIZE';
ORIGIN_MEMBER_NAME PROPERTY_NAME            PROPERTY_VALUE IS_GLOBAL
------------------ ------------------------ -------------- ---------
G1N1               PRIVATE_STATIC_AREA_SIZE 300000000      FALSE
G1N2               PRIVATE_STATIC_AREA_SIZE 300000000      FALSE
2 rows selected.
```

However, DDL_AUTOCOMMIT can be altered up to GLOBAL as follows. Therefore, an error occurs when specifying a member by using AT clause in *alter system set* statement.

```
% gsql test test
Connected to GOLDILOCKS Database.
gSQL> select origin_member_name, property_name, property_value, is_global
    2   from gv$property
    3  where property_name = 'DDL_AUTOCOMMIT';
ORIGIN_MEMBER_NAME PROPERTY_NAME  PROPERTY_VALUE IS_GLOBAL
------------------ -------------- -------------- ---------
G1N1               DDL_AUTOCOMMIT NO             TRUE
G1N2               DDL_AUTOCOMMIT NO             TRUE
2 rows selected.
gSQL> alter system set ddl_autocommit = false at g1n2;
ERR-42000(16398): the domain of property does not match with domain 'G1N2' :
alter system set ddl_autocommit = false at g1n2
                                          *
ERROR at line 1:
gSQL> alter system set ddl_autocommit = false;
System altered.
gSQL> select origin_member_name, property_name, property_value, is_global
    2   from gv$property
    3  where property_name = 'DDL_AUTOCOMMIT';
ORIGIN_MEMBER_NAME PROPERTY_NAME  PROPERTY_VALUE IS_GLOBAL
------------------ -------------- -------------- ---------
G1N1               DDL_AUTOCOMMIT NO             TRUE
```

```
G1N2                DDL_AUTOCOMMIT NO             TRUE
2 rows selected.
```

4.

What's New

4.1 Feature Matrix

This chapter briefly describes the features added to each major version.

Architecture

System Architecture

The following is a feature matrix for system architecture.

Table 4-1 Feature matrix for system architecture

Feature	2.x	3.x	20c.1	21c.1	22c.1
Shared Nothing Cluster	X	O	O	O	O
DA (Direct Attach)	O	O	O	O	O
JDBC DA (Direct Attach)	X	O	O	O	O
C/S (Client/Server) Dedicated	O	O	O	O	O
C/S (Client/Server) Shared	O	O	O	O	O
multi-process applications	O	O	O	O	O
multi-threaded applications	O	O	O	O	O
Linux platform	O	O	O	O	O
HP platform	O	O	O	O	O
AIX platform	O	O	O	O	O
Windows Client Platform	O	O	O	O	O
CDC(Change Data Capture) replication	O	O	O	O	O
CDC replication with log mirror	O	O	O	O	O
multi-level start up	O	O	O	O	O
parallel database loading	O	O	O	O	O
parallel index build	O	O	O	O	O
SQL plan cache	O	O	O	O	O
IPC	X	X	X	O	O

Storage Internal

The following is a feature matrix for storage internal.

Table 4-2 Feature matrix for storage internal

Feature	2.x	3.x	20c.1	21c.1	22c.1
memory dictionary tablespace	O	O	O	O	O
memory data tablespace	O	O	O	O	O
memory undo tablespace	O	O	O	O	O
memory temporary tablespace	O	O	O	O	O
memory bitmap data segment	O	O	O	O	O
memory bitmap undo segment	O	O	O	O	O
memory bitmap instant segment	O	O	O	O	O
memory heap table	O	O	O	O	O
memory instant table	O	O	O	O	O
memory B-tree index	O	O	O	O	O
memory instant B-tree	O	O	O	O	O
memory instant hash	O	O	O	O	O
global secondary index	X	O	O	O	O
disk data tablespace	X	X	O	O	O
disk bitmap data segment	X	X	O	O	O
disk B-tree index	X	X	O	O	O
disk global secondary index	X	X	O	O	O

Transaction Control

The following is a feature matrix for transaction control.

Table 4-3 Feature matrix for transaction control

Feature	2.x	3.x	20c.1	21c.1	22c.1
CDS(Concurrency Data Store) database mode	O	O	O	O	O
TDS(Transactional Data Store) database mode	O	O	O	O	O
read-only database	O	O	O	O	O
read/write database	O	O	O	O	O
flat transaction	O	O	O	O	O
distributed transaction	O	O	O	O	O
read-only transaction	O	O	O	O	O
read/write transaction	O	O	O	O	O
READ COMMITTED isolation level	O	O	O	O	O
SERIALIZABLE isolation level with SELECT FOR UPDATE	O	O	O	O	O
MVCC(Multi Version Concurrency Control)	O	O	O	O	O
multi-version read consistency	O	O	O	O	O
multi-statement consistent read	O	O	O	O	O
implicit lock for DML	O	O	O	O	O
writer don't blocks readers	O	O	O	O	O
row-level locking	O	O	O	O	O
deadlock detection	O	O	O	O	O
deadlock resolution	O	O	O	O	O
lock granularity	O	O	O	O	O
read lock	O	O	O	O	O
write lock	O	O	O	O	O
intention lock	O	O	O	O	O
WAL(Write Ahead Logging)	O	O	O	O	O
repeat history	O	O	O	O	O
restart recovery	O	O	O	O	O
circular logging	O	O	O	O	O
buffered logging	O	O	O	O	O
logging group	O	O	O	O	O
supplemental logging	O	O	O	O	O
mirrored logging	O	O	O	O	O
synchronous commit	O	O	O	O	O
asynchronous commit	O	O	O	O	O
grouped commit	O	O	O	O	O
total rollback	O	O	O	O	O
implicit statement rollback	O	O	O	O	O

Feature	2.x	3.x	20c.1	21c.1	22c.1
savepoint management	O	O	O	O	O

Backup & Recovery

The following is a feature matrix for backup & recovery.

Table 4-4 Feature matrix for backup & recovery

Feature	2.x	3.x	20c.1	21c.1	22c.1
off-line backup	O	O	O	O	O
on-line backup	O	O	O	O	O
full backup	O	O	O	O	O
incremental backup	O	O	O	O	O
complete recovery	O	O	O	O	O
incomplete recovery	O	O	O	O	O
auto instance recovery	O	O	O	O	O
tablespace recovery	O	O	O	O	O
file recovery	O	O	O	O	O
change tracking	X	X	O	O	O

Database Information

DICTIONARY_SCHEMA Schema

The following is a feature matrix for DICTIONARY_SCHEMA schema.

Table 4-5 Feature matrix for DICTIONARY_SCHEMA schema

Family	Feature	2.x	3.x	20c.1	21c.1	22c.1
Views of ALL_family	ALL_ALL_TABLES	O	O	O	O	O
	ALL_ARGUMENTS	X	O	O	O	O
	ALL_CATALOG	O	O	O	O	O
	ALL_CLUSTER_TABLES	X	O	O	O	O
	ALL_COL_COMMENTS	O	O	O	O	O
	ALL_COL_PLACE	X	X	X	X	X
	ALL_COL_PRIVS	O	O	O	O	O
	ALL_COL_PRIVS_MADE	O	O	O	O	O
	ALL_COL_PRIVS_RECD	O	O	O	O	O
	ALL_CONSTRAINTS	O	O	O	O	O
	ALL_CONS_COLUMNS	O	O	O	O	O
	ALL_DB_PRIVS	O	O	O	O	O
	ALL_DB_PRIVS_MADE	O	O	O	O	O
	ALL_DB_PRIVS_RECD	O	O	O	O	O
	ALL_DEPENDENCIES	X	O	O	O	O
	ALL_GLOBAL_SECONDARY_INDEXES	X	O	O	O	O
	ALL_GSI_PLACE	X	O	O	O	O
	ALL_INDEXES	O	O	O	O	O
	ALL_IND_COLUMNS	O	O	O	O	O
	ALL_IND_PLACE	X	O	O	O	O
	ALL_NONSCHEMA_COMMENTS	O	O	O	O	O
	ALL_OBJECTS	O	O	O	O	O
	ALL_PACKAGE_PRIVS	X	X	O	O	O
	ALL_PACKAGE_PRIV_MADE	X	X	O	O	O
	ALL_PACKAGE_PRIV_RECD	X	X	O	O	O
	ALL_PROCEDURES	X	O	O	O	O
	ALL_PROC_PRIVS	X	O	O	O	O
	ALL_PROC_PRIV_MADE	X	O	O	O	O
	ALL_PROC_PRIV_RECD	X	O	O	O	O
	ALL_SCHEMAS	O	O	O	O	O
	ALL_SCHEMA_PATH	O	O	O	O	O
	ALL_SCHEMA_PRIVS	O	O	O	O	O
	ALL_SCHEMA_PRIVS_MADE	O	O	O	O	O

Family	Feature	2.x	3.x	20c.1	21c.1	22c.1
	ALL_SCHEMA_PRIVS_RECD	O	O	O	O	O
	ALL_SEQUENCES	O	O	O	O	O
	ALL_SEQ_PRIVS	O	O	O	O	O
	ALL_SEQ_PRIVS_MADE	O	O	O	O	O
	ALL_SEQ_PRIVS_RECD	O	O	O	O	O
	ALL_SHARD_KEY_COLUMNS	X	O	O	O	O
	ALL_SOURCE	X	O	O	O	O
	ALL_SYNONYMS	O	O	O	O	O
	ALL_TABLES	O	O	O	O	O
	ALL_TAB_COLS	O	O	O	O	O
	ALL_TAB_COLUMNS	O	O	O	O	O
	ALL_TAB_COMMENTS	O	O	O	O	O
	ALL_TAB_IDENTITY_COLS	O	O	O	O	O
	ALL_TAB_PLACE	X	O	O	O	O
	ALL_TAB_SHARDS	X	O	O	O	O
	ALL_TAB_PRIVS	O	O	O	O	O
	ALL_TAB_PRIVS_MADE	O	O	O	O	O
	ALL_TAB_PRIVS_RECD	O	O	O	O	O
	ALL_TBS_PRIVS	O	O	O	O	O
	ALL_TBS_PRIVS_MADE	O	O	O	O	O
	ALL_TBS_PRIVS_RECD	O	O	O	O	O
	ALL_USERS	O	O	O	O	O
	ALL_VIEWS	O	O	O	O	O
	DBA_ALL_TABLES	O	O	O	O	O
	DBA_ARGUMENTS	X	O	O	O	O
	DBA_CATALOG	O	O	O	O	O
	DBA_CLUSTER	X	O	O	O	O
	DBA_CLUSTER_COMMENTS	X	O	O	O	O
	DBA_CLUSTER_TABLES	X	O	O	O	O
	DBA_COL_COMMENTS	O	O	O	O	O
	DBA_COL_PLACE	X	X	X	X	X
	DBA_COL_PRIVS	O	O	O	O	O
	DBA_CONSTRAINTS	O	O	O	O	O
	DBA_CONS_COLUMNS	O	O	O	O	O
	DBA_DB_PRIVS	O	O	O	O	O
	DBA_DEPENDENCIES	X	O	O	O	O
	DBA_EXTENTS	O	O	O	O	O
	DBA_GLOBAL_SECONDARY_INDEXES	X	O	O	O	O
	DBA_GSI_PLACE	X	O	O	O	O
	DBA_INDEXES	O	O	O	O	O

Family	Feature	2.x	3.x	20c.1	21c.1	22c.1
Views of DBA_family	DBA_IND_COLUMNS	O	O	O	O	O
	DBA_IND_PLACE	X	O	O	O	O
	DBA_NONSCHEMA_COMMENTS	O	O	O	O	O
	DBA_OBJECTS	O	O	O	O	O
	DBA_PACKAGE_PRIVS	X	X	O	O	O
	DBA_PROCEDURES	X	O	O	O	O
	DBA_PROC_PRIVS	X	O	O	O	O
	DBA_PROFILES	O	O	O	O	O
	DBA_RECYCLEBIN	X	X	O	O	O
	DBA_SCHEMAS	O	O	O	O	O
	DBA_SCHEMA_PATH	O	O	O	O	O
	DBA_SCHEMA_PRIVS	O	O	O	O	O
	DBA_SEQUENCES	O	O	O	O	O
	DBA_SEQ_PRIVS	O	O	O	O	O
	DBA_SHARD_KEY_COLUMNS	X	O	O	O	O
	DBA_SOURCE	X	O	O	O	O
	DBA_STAT_SYSTEM	X	O	O	O	O
	DBA_SYNONYMS	O	O	O	O	O
	DBA_SYS_PRIVS	O	O	O	O	O
	DBA_TABLES	O	O	O	O	O
	DBA_TABLESPACES	O	O	O	O	O
	DBA_TAB_COLS	O	O	O	O	O
	DBA_TAB_COLUMNS	O	O	O	O	O
	DBA_TAB_COMMENTS	O	O	O	O	O
	DBA_TAB_IDENTITY_COLS	O	O	O	O	O
	DBA_TAB_PLACE	X	O	O	O	O
	DBA_TAB_PRIVS	O	O	O	O	O
	DBA_TAB_SHARDS	X	O	O	O	O
	DBA_TBS_PRIVS	O	O	O	O	O
	DBA_USERS	O	O	O	O	O
	DBA_VIEWS	O	O	O	O	O
	USER_ALL_TABLES	O	O	O	O	O
	USER_ARGUMENTS	X	O	O	O	O
	USER_CATALOG	O	O	O	O	O
	USER_CLUSTER_TABLES	X	O	O	O	O
	USER_COL_COMMENTS	O	O	O	O	O
	USER_COL_PLACE	X	X	X	X	X
	USER_COL_PRIVS	O	O	O	O	O
	USER_COL_PRIVS_MADE	O	O	O	O	O
	USER_COL_PRIVS_RECD	O	O	O	O	O

Family	Feature	2.x	3.x	20c.1	21c.1	22c.1
Views of USER_family	USER_CONSTRAINTS	O	O	O	O	O
	USER_CONS_COLUMNS	O	O	O	O	O
	USER_DEPENDENCIES	X	O	O	O	O
	USER_EXTENTS	O	O	O	O	O
	USER_GLOBAL_SECONDARY_INDEXES	X	O	O	O	O
	USER_GSI_PLACE	X	O	O	O	O
	USER_INDEXES	O	O	O	O	O
	USER_IND_COLUMNS	O	O	O	O	O
	USER_IND_PLACE	X	O	O	O	O
	USER_OBJECTS	O	O	O	O	O
	USER_PACKAGE_PRIVS	X	X	O	O	O
	USER_PACKAGE_PRIVS_MADE	X	X	O	O	O
	USER_PACKAGE_PRIVS_RECD	X	X	O	O	O
	USER_PROCEDURES	X	O	O	O	O
	USER_PROC_PRIVS	X	O	O	O	O
	USER_PROC_PRIVS_MADE	X	O	O	O	O
	USER_PROC_PRIVS_RECD	X	O	O	O	O
	USER_RECYCLEBIN	X	X	O	O	O
	USER_SCHEMAS	O	O	O	O	O
	USER_SCHEMA_PATH	O	O	O	O	O
	USER_SCHEMA_PRIVS	O	O	O	O	O
	USER_SCHEMA_PRIVS_MADE	O	O	O	O	O
	USER_SCHEMA_PRIVS_RECD	O	O	O	O	O
	USER_SEQUENCES	O	O	O	O	O
	USER_SEQ_PRIVS	O	O	O	O	O
	USER_SEQ_PRIVS_MADE	O	O	O	O	O
	USER_SEQ_PRIVS_RECD	O	O	O	O	O
	USER_SHARD_KEY_COLUMNS	X	O	O	O	O
	USER_SOURCE	X	O	O	O	O
	USER_SYNONYMS	O	O	O	O	O
	USER_SYS_PRIVS	O	O	O	O	O
	USER_TABLES	O	O	O	O	O
	USER_TABLESPACES	O	O	O	O	O
	USER_TAB_COLS	O	O	O	O	O
	USER_TAB_COLUMNS	O	O	O	O	O
	USER_TAB_COMMENTS	O	O	O	O	O
	USER_TAB_IDENTITY_COLS	O	O	O	O	O
	USER_TAB_PLACE	X	O	O	O	O
	USER_TAB_PRIVS	O	O	O	O	O
	USER_TAB_PRIVS_MADE	O	O	O	O	O

Family	Feature	2.x	3.x	20c.1	21c.1	22c.1
	USER_TAB_PRIVS_RECD	O	O	O	O	O
	USER_TAB_SHARDS	X	O	O	O	O
	USER_USERS	O	O	O	O	O
	USER_VIEWS	O	O	O	O	O
Other views	AUDIT_POLICIES	X	O	O	O	O
	AUDIT_POLICY_ENABLED	X	O	O	O	O
	AUDIT_POLICY_OPTIONS	X	O	O	O	O
	AUDIT_TRAIL	X	O	O	O	O
	DATABASE_PROPERTIES	O	O	O	O	O
	DBC_TABLE_TYPE_INFO	O	O	O	O	O
	DICTIONARY	O	O	O	O	O
	DICT_COLUMNS	O	O	O	O	O
	IMPLEMENTATION_INFO	O	O	O	O	O
	IMPLEMENTATION_INFO_BASE	O	O	O	O	O
	JDBC_CLIENT_PROPS	O	O	O	O	O
	PRODUCT	O	O	O	O	O
	SESSION_PRIVS	O	O	O	O	O
	SUPPLEMENTAL_LOG_TABLE_INFO	O	O	O	O	O
Aliased synonym	COLS	O	O	O	O	O
	DICT	O	O	O	O	O
	IND	O	O	O	O	O
	OBJ	O	O	O	O	O
	RECYCLEBIN	X	X	O	O	O
	SEQ	O	O	O	O	O
	TABS	O	O	O	O	O

INFORMATION_SCHEMA Schema

The following is a feature matrix for INFORMATION_SCHEMA schema.

Table 4-6 Feature matrix for INFORMATION_SCHEMA schema

Feature	2.x	3.x	20c.1	21c.1	22c.1
COLUMNS	O	O	O	O	O
COLUMN_PRIVILEGES	O	O	O	O	O
CONSTRAINT_COLUMN_USAGE	O	O	O	O	O
CONSTRAINT_TABLE_USAGE	O	O	O	O	O
INFORMATION_SCHEMA_CATALOG_NAME	O	O	O	O	O
KEY_COLUMN_USAGE	O	O	O	O	O
MODULES	X	X	O	O	O
MODULE_BODY	X	X	O	O	O

Feature	2.x	3.x	20c.1	21c.1	22c.1
MODULE_BODY_MODULE_USAGE	X	X	O	O	O
MODULE_BODY_ROUTINE_USAGE	X	X	O	O	O
MODULE_BODY_SEQUENCE_USAGE	X	X	O	O	O
MODULEBODY_TABLE_USAGE	X	X	O	O	O
MODULE_MODULE_USAGE	X	X	O	O	O
MODULE_PRIVILEGES	X	X	O	O	O
MODULE_ROUTINE_USAGE	X	X	O	O	O
MODULE_SEQUENCE_USAGE	X	X	O	O	O
MODULE_TABLE_USAGE	X	X	O	O	O
PARAMETERS	X	O	O	O	O
REFERENTIAL_CONSTRAINTS	O	O	O	O	O
ROUTINES	X	O	O	O	O
ROUTINE_MODULE_USAGE	X	X	O	O	O
ROUTINE_PRIVILEGES	X	O	O	O	O
ROUTINE_ROUTINE_USAGE	X	O	O	O	O
ROUTINE_SEQUENCE_USAGE	X	O	O	O	O
ROUTINE_TABLE_USAGE	X	O	O	O	O
SCHEMATA	O	O	O	O	O
SEQUENCES	O	O	O	O	O
SQL_FEATURES	O	O	O	O	O
SQL_IMPLEMENTATION_INFO	O	O	O	O	O
SQL_PACKAGES	O	O	O	O	O
SQL_PARTS	O	O	O	O	O
SQL_SIZING	O	O	O	O	O
STATISTICS	O	O	O	O	O
TABLES	O	O	O	O	O
TABLE_CONSTRAINTS	O	O	O	O	O
TABLE_PRIVILEGES	O	O	O	O	O
USAGE_PRIVILEGES	O	O	O	O	O
VIEWS	O	O	O	O	O
VIEW_MODULE_USAGE	X	X	O	O	O
VIEW_ROUTINE_USAGE	X	O	O	O	O
VIEW_TABLE_USAGE	O	O	O	O	O

PERFORMANCE_VIEW_SCHEMA Schema

The following is a feature matrix for PERFORMANCE_VIEW_SCHEMA schema.

Table 4-7 Feature matrix for PERFORMANCE_VIEW_SCHEMA schema

Feature	2.x	3.x	20c.1	21c.1	22c.1
GV$____	X	O	O	O	O
V$AGABLE_INFO	X	O	O	O	O
V$ARCHIVELOG	O	O	O	O	O
V$AUDITABLE_DB_PRIVILEGES	X	O	O	O	O
V$AUDITABLE_SYSTEM_ACTIONS	X	O	O	O	O
V$BACKUP	O	O	O	O	O
V$BALANCER	O	O	O	O	O
V$BCH	X	X	O	O	O
V$BUFFER_STAT	X	X	O	O	O
V$CLUSTER_DISPATCHER	X	O	O	O	O
V$CLUSTER_LOCATION	X	O	O	O	O
V$CLUSTER_MEMBER	X	O	O	O	O
V$COLUMNS	O	O	O	O	O
V$CONTROLFILE	O	O	O	O	O
V$DATAFILE	O	O	O	O	O
V$DB_CHANGE_TRACKING	X	X	O	O	O
V$DB_FILE	O	O	O	O	O
V$DB_PROPERTY	X	X	X	X	O
V$DISPATCHER	O	O	O	O	O
V$ERROR_CODE	O	O	O	O	O
V$GLOBAL_TRANSACTION	O	O	O	O	O
V$JOURNALING	X	O	O	O	O
V$INCREMENTAL_BACKUP	O	O	O	O	O
V$INSTANCE	O	O	O	O	O
V$KEYWORDS	O	O	O	O	O
V$LATCH	O	O	O	O	O
V$LICENSE	X	X	X	X	O
V$LOCK_WAIT	O	O	O	O	O
V$LOCKED_OBJECT	X	X	O	O	O
V$LOGFILE	O	O	O	O	O
V$OPEN_CURSOR	X	X	X	X	O
V$PLAN_HISTORY	X	X	X	O	O
V$PLAN_HISTORY_LATEST	X	X	X	O	O
V$PROCESS_MEM_STAT	O	O	O	O	O
V$PROCESS_SQL_STAT	O	O	O	O	O
V$PROCESS_STAT	O	O	O	O	O
V$PROPERTY	O	O	O	O	O
V$PROPERTY_ALIAS	X	X	X	X	O
V$PSM_RESERVED_WORDS	X	O	O	O	O

Feature	2.x	3.x	20c.1	21c.1	22c.1
V$QUEUE	O	O	O	O	O
V$RESERVED_WORDS	O	O	O	O	O
V$SESSION	O	O	O	O	O
V$SESSION_AUDIT	X	O	O	O	O
V$SESSION_CONNECT_INFO	O	O	O	O	O
V$SESSION_EVENT	X	O	O	O	O
V$SESSION_MEM_STAT	O	O	O	O	O
V$SESSION_MEM_USAGE	X	X	X	O	O
V$SESSION_SQL_STAT	O	O	O	O	O
V$SESSION_STAT	O	O	O	O	O
V$SESSION_WAIT	X	O	O	O	O
V$SHARED_MODE	O	O	O	O	O
V$SHARED_SERVER	O	O	O	O	O
V$SHM_SEGMENT	O	O	O	O	O
V$SPROPERTY	O	O	O	O	O
V$SQLFN_METADATA	O	O	O	O	O
V$SQL_CACHE	O	O	O	O	O
V$SQL_COMMAND	X	O	O	O	O
V$SQL_HISTORY	X	O	O	O	O
V$STATEMENT	O	O	O	O	O
V$SYSTEM_EVENT	X	O	O	O	O
V$SYSTEM_MEM_STAT	O	O	O	O	O
V$SYSTEM_SQL_STAT	O	O	O	O	O
V$SYSTEM_STAT	O	O	O	O	O
V$TABLES	O	O	O	O	O
V$TABLESPACE	O	O	O	O	O
V$TABLESPACE_STAT	X	O	O	O	O
V$TRANSACTION	O	O	O	O	O
V$WAIT_EVENT_CLASS_NAME	X	O	O	O	O
V$WAIT_EVENT_NAME	X	O	O	O	O
V$XA_TRANSATION	X	O	O	O	O

Server Property

The following is a feature matrix for server property.

Table 4-8 Feature matrix for server property

Feature	2.x	3.x	20c.1	21c.1	22c.1
ADMIN_SESSION_POOL_INIT_SIZE	X	X	X	X	O
ADMIN_SESSION_POOL_NEXT_SIZE	X	X	X	X	O
AGING_INTERVAL	O	O	O	O	O
AGING_PLAN_INTERVAL	O	O	O	O	O
ARCHIVELOG_DIR	O	X	X	X	X
ARCHIVELOG_DIR_1 ~ DIR_10	O	O	O	O	O
ARCHIVELOG_FILE	O	O	O	O	O
ARCHIVELOG_MODE	O	O	O	O	O
BACKUP_DIR_1 ~ DIR_10	O	O	O	O	O
BLOCK_READ_COUNT	O	O	O	O	O
BROADCAST_INDEX_REBUILD_PROTOCOL	X	X	X	O	O
BROADCAST_REBALANCE_PROTOCOL	X	X	O	O	O
BUFFER_CACHE_SIZE	X	X	O	O	O
BUFFER_CHECKPOINT_LIST_COUNT	X	X	O	X	X
BUFFER_DIRTY_PAGE_LIMIT	X	X	X	X	O
BUFFER_FLUSH_THREADS	X	X	O	X	X
BUFFER_FLUSHING_INTERVAL	X	X	O	X	X
BUFFER_FREE_LIST_COUNT	X	X	O	O	O
BUFFER_HASH_BUCKETS	X	X	O	O	O
BUFFER_HOT_REGION_CRITERIA	X	X	O	O	O
BUFFER_HOT_REGION_PERCENT	X	X	O	O	O
BUFFER_LRU_LIST_COUNT	X	X	O	O	O
BUFFER_LRU_SCAN_PERCENT	X	X	X	X	O
BUFFER_MULTIPAGE_READ_COUNT	X	X	O	O	O
BUFFER_PREFETCH_PAGE_COUNT	X	X	X	O	O
BULK_IO_PAGE_COUNT	O	O	O	O	O
CDISPATCHER_HOT_POLICY_INTERVAL	X	O	O	O	O
CDISPATCHER_LOCKABLE_THREADS	X	X	X	X	O
CDISPATCHER_LOCKLESS_THREADS	X	X	O	O	O
CDISPATCHER_MAX_PACKET_BUFFER_SIZE	X	X	X	X	O
CDISPATCHER_SOCKET_BUFFER_SIZE	X	O	O	O	O
CHANGE_TRACKING	X	X	O	O	O
CHANGE_TRACKING_EXTENT_SIZE	X	X	O	O	O
CHANGE_TRACKING_FILE	X	X	O	O	O
CHAR_LENGTH_UNITS	O	O	O	O	O

Feature	2.x	3.x	20c.1	21c.1	22c.1
CHARACTER_SET	O	O	O	O	O
CHECK_DEDICATE_CONNECTION_INTERVAL	X	O	O	O	O
CHECK_DEDICATE_SOCKET	X	X	X	X	X
CHECKPOINT_LIST_COUNT_PER_IO_GROUP	X	X	X	X	O
CLIENT_MAX_COUNT	O	O	O	O	O
CLIENT_NUMA_POLICY	X	O	O	O	O
CLOSE_PSM_CHILD_STMTS	X	O	O	O	O
CLUSTER_ASYNC_COMMIT	X	O	O	O	O
CLUSTER_ASYNC_REPLICATION	X	O	O	O	X
CLUSTER_CM_BUFFER_COUNT	X	O	O	O	X
CLUSTER_CM_BUFFER_SIZE	X	O	O	O	O
CLUSTER_CM_READ_BUFFER_SIZE	X	O	O	O	O
CLUSTER_COMMIT_SLAVE_CSERVERS	X	X	X	X	O
CLUSTER_COMMIT_STREAM_ISOLATION	X	O	O	O	O
CLUSTER_CONNECTION	X	O	O	O	O
CLUSTER_CONNECTION_TIMEOUT_SEC	X	O	O	O	O
CLUSTER_DATA_SYNC_SERVERS	X	O	O	O	O
CLUSTER_DEADLOCK_TIMEOUT	X	X	O	O	O
CLUSTER_DISPATCHER_IN_QUEUE_SIZE	X	O	O	O	O
CLUSTER_DISPATCHER_NUMA_STREAM_MAP	X	O	O	O	O
CLUSTER_DISPATCHER_OUT_QUEUE_SIZE	X	O	O	O	O
CLUSTER_GSERVER_RESPONSE_QUEUE_SIZE	X	X	X	X	O
CLUSTER_HEARTBEAT_INTERVAL	X	O	O	O	O
CLUSTER_HEARTBEAT_RETRY_COUNT	X	O	O	O	O
CLUSTER_IGNORE_INACTIVE_MEMBER	X	O	O	O	O
CLUSTER_KEEPALIVE_IDLE_TIME	X	X	X	X	O
CLUSTER_LOCKABLE_CSERVERS	X	X	X	X	O
CLUSTER_LOCKLESS_CSERVERS	X	X	X	X	O
CLUSTER_MAX_PACKET_SIZE	X	O	O	O	O
CLUSTER_MAX_PAYLOAD_SIZE	X	O	O	O	O
CLUSTER_PACKET_ALLOCATION_TIMEOUT	X	O	O	O	O
CLUSTER_SESSION_HASH_BUCKETS	X	X	O	O	O
CLUSTER_SPLIT_BRAIN_RESOLUTION_POLICY	X	O	O	O	O
CLUSTER_SPLIT_BRAIN_RETRY_COUNT	X	O	O	O	O
COMMITTER_HOT_POLICY_INTERVAL	X	O	O	O	O
CONTROL_FILE_0 ~ FILE_7	O	O	O	O	O
CONTROL_FILE_COUNT	O	O	O	O	O
CONTROL_FILE_TEMP_NAME	O	O	O	O	O
COORDINATOR_COMMIT_WRITE_MODE	X	O	O	O	O
DA_CLIENT_NUMA_MODE	X	O	O	O	O

Feature	2.x	3.x	20c.1	21c.1	22c.1
DATA_STORE_MODE	O	O	O	O	O
DATABASE_ACCESS_MODE	O	O	O	O	O
DATABASE_INSTANCE_NAME	X	O	O	O	O
DDL_AUTOCOMMIT	O	O	O	O	O
DDL_LOCK_TIMEOUT	O	O	O	O	O
DEADLOCK_PRIORITY	X	X	O	O	O
DEFAULT_GLOBAL_SECONDARY_INDEX_CREATION	X	O	O	O	O
DEFAULT_INDEX_LOGGING	O	O	X	X	X
DEFAULT_INDEX_PCTFREE	X	O	O	O	O
DEFAULT_INITRANS	O	O	O	O	O
DEFAULT_MAXTRANS	O	O	O	O	O
DEFAULT_PCTFREE	O	O	O	O	O
DEFAULT_PCTUSED	O	O	O	O	O
DEFAULT_REMOVAL_BACKUP_FILE	O	O	O	O	O
DEFAULT_REMOVAL_OBSOLETE_BACKUP_LIST	O	O	O	O	O
DEFAULT_SHARDING	X	O	O	O	O
DISABLE_DDL	X	X	X	O	O
DISABLE_DDL_CDC_GIVEUP	O	O	O	O	O
DISABLE_SERIAL_DDL	X	X	X	O	O
DISABLE_UPDATE_PK_CDC_GIVEUP	O	O	O	O	O
DISALLOWED_PROTOCOL_TARGETTYPE	X	O	O	O	O
DISALLOWED_PROTOCOL_TARGETTYPE_WITH_ALL	X	O	O	O	O
DISALLOWED_PROTOCOL_TARGETTYPE_WITH_NAME	X	O	O	O	O
DISPATCHERS	O	O	O	O	O
DISPATCHER_CM_BUFFER_SIZE	O	O	O	O	O
DISPATCHER_CM_UNIT_SIZE	O	O	O	O	O
DISPATCHER_CONNECTIONS	O	O	O	O	O
DISPATCHER_HOT_POLICY_INTERVAL	X	O	O	O	O
DISPATCHER_LOAD_BALANCING	X	O	O	O	O
DISPATCHER_NUMA_STREAM_MAP	X	O	O	O	O
DISPATCHER_QUEUE_SIZE	O	O	O	O	O
DISPATCHER_REQUEST_MINI_QUEUE_COUNT	X	O	O	O	O
DISPATCHER_RESPONSE_MINI_QUEUE_COUNT	X	O	O	O	O
EXECUTE_INST_HASH_TABLE_USING_AVAILABLE_MEMORY	X	X	X	O	O
FETCH_FAILOVER	X	O	O	O	O
FULL_TABLE_SCAN_CACHING_THRESHOLD	X	X	X	X	O
GLOBAL_CONNECTION_ALLOW_SESSION_DEPENDENCY	X	O	O	O	O
GLOBAL_JOURNAL_BUFFER_SIZE	X	O	O	O	O
GLOBAL_JOURNAL_BUFFER_TOTAL_MAX_SIZE	X	O	O	O	O
GLOBAL_PROPERTY_LOCK_TIMEOUT	X	O	O	O	O

Feature	2.x	3.x	20c.1	21c.1	22c.1
GLOBAL_TRANSACTION_COMMIT_WRITE_MODE	X	O	O	O	O
GLOBAL_TRANSACTION_ISOLATION_SCOPE	X	O	O	O	O
GLOBAL_TRANSACTION_LOG_DIR	X	O	O	O	O
GLOBAL_TRANSACTION_LOG_FILE_SIZE	X	O	O	O	O
GMASTER_NUMA_NODE	X	O	O	O	O
GMON_AUTOSTART	X	O	O	O	O
HINT_ERROR	O	O	O	O	O
IDLE_TIMEOUT	O	O	O	O	O
IN_DOUBT_DECISION	O	O	O	O	O
IN_KEY_RANGE_ARRAY_COUNT	X	X	O	O	O
INCREMENTAL_BACKUP_SCAN_BUFFER_SIZE	X	X	O	O	O
INCREMENTAL_DATAFILE_HEADER_UPDATE_CRITERIA	X	X	X	X	O
INDEX_BUILD_PARALLEL_FACTOR	O	O	O	O	O
INDEX_MERGE_RUN_COUNT	O	O	O	O	O
INDEX_REBUILD_BLOCK_READ_COUNT	X	X	O	O	O
INDEX_SORT_RUN_SIZE	O	O	O	O	O
INDEX_TREE_MERGE_PARALLEL_FACTOR	X	O	O	O	O
INST_ALLOCATOR_COUNT	X	O	O	O	O
INST_HASH_TABLE_BUCKET_MAX_COUNT	X	X	X	O	O
INST_TABLE_BLOCK_SIZE	X	O	O	O	O
IPC_CHANNEL_COUNT	X	X	X	O	O
JOURNAL_TEMP_DIR	X	O	O	O	O
KEEPALIVE_IDLE_TIME	O	O	O	O	O
LOCAL_CLUSTER_MEMBER	X	O	O	O	O
LOCAL_CLUSTER_MEMBER_HOST	X	O	O	O	O
LOCAL_CLUSTER_MEMBER_PORT	X	O	O	O	O
LOCAL_JOURNAL_BUFFER_SIZE	X	O	O	O	O
LOCATION_FILE	X	O	O	O	O
LOCATOR_QUERY_TIMEOUT	X	O	O	O	O
LOCK_HASH_TABLE_SIZE	O	O	O	O	O
LOCKABLE_DISPATCHER_CM_BUFFER_COUNT	X	X	X	X	O
LOCKLESS_DISPATCHER_CM_BUFFER_COUNT	X	X	X	X	O
LOG_BLOCK_SIZE	O	O	O	O	O
LOG_BUFFER_SIZE	O	O	O	O	O
LOG_DIR	O	O	O	O	O
LOG_FILE_SIZE	O	O	O	O	O
LOG_GROUP_COUNT	O	O	O	O	O
LOG_MIRROR_MODE	O	O	O	O	O
LOG_MIRROR_SHARED_MEMORY_STATIC_SIZE	O	O	O	O	O
LOG_MIRROR_TIMEOUT	O	O	O	O	O

Feature	2.x	3.x	20c.1	21c.1	22c.1
LOG_SYNC_INTERVAL	O	O	O	O	O
LOG_SYNC_INTERVAL_MSEC	X	O	O	O	O
MAX_GROUP_COUNT	X	O	O	O	O
MAX_JOURNAL_FILE_SIZE	X	O	O	O	O
MAX_NODE_COUNT	X	O	O	O	O
MAXIMUM_CONCURRENT_ACTIVITIES	O	O	O	O	O
MAXIMUM_FILE_CACHE_SIZE	X	X	X	O	O
MAXIMUM_FLANGE_COUNT	X	O	O	O	X
MAXIMUM_FLUSH_BUFFER_PAGE_COUNT	X	X	X	O	O
MAXIMUM_FLUSH_LOG_BLOCK_COUNT	O	O	O	O	O
MAXIMUM_FLUSH_PAGE_COUNT	O	O	O	O	O
MAXIMUM_INDEX_REBUILD_JOURNAL_REPLAY_COUNT	X	X	O	O	O
MAXIMUM_JOURNAL_REPLAY_COUNT	X	O	O	O	O
MAXIMUM_NAMED_CURSOR_COUNT	O	O	O	O	O
MAXIMUM_PACKAGE_INSTANCE_COUNT	X	X	X	O	O
MAXIMUM_SESSION_CM_BUFFER_SIZE	O	O	O	O	O
MEASURE_CLUSTER_LATENCY	X	O	O	O	O
MEDIA_RECOVERY_LOG_BUFFER_SIZE	O	X	X	X	X
MIN_SAMPLE_ROW_COUNT	X	O	O	O	O
MINIMUM_UNDO_PAGE_COUNT	O	O	O	O	O
NET_BUFFER_SIZE	O	O	O	O	O
NLS_DATE_FORMAT	O	O	O	O	O
NLS_TIME_FORMAT	O	O	O	O	O
NLS_TIME_WITH_TIME_ZONE_FORMAT	O	O	O	O	O
NLS_TIMESTAMP_FORMAT	O	O	O	O	O
NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT	O	O	O	O	O
NUMA	X	O	O	O	O
NUMA_MAP	X	O	O	O	O
OFFLINE_MEMBER_AFTER_FAILOVER	X	O	O	O	O
ONLINE_INDEX_REBUILD_JOURNAL_REPLAY_THRESHOLD	X	X	O	O	O
ONLINE_JOURNAL_REPLAY_THRESHOLD	X	O	O	O	O
OS_GROUP_ACCESS	X	O	O	O	O
PACKET_COMPRESSION_THRESHOLD	X	X	O	O	O
PAGE_CHECKSUM_TYPE	O	O	O	O	O
PARALLEL_IO_FACTOR	O	O	O	O	O
PARALLEL_IO_GROUP_1 ~ GROUP_16	O	O	O	O	O
PARALLEL_LOAD_FACTOR	O	O	O	O	O
PENDING_LOG_BUFFER_COUNT	O	O	O	O	O
PLAN_CACHE	O	O	O	O	O
PLAN_CACHE_SIZE	O	O	O	O	O

Feature	2.x	3.x	20c.1	21c.1	22c.1
PLAN_HISTORY	X	X	X	O	O
PLAN_HISTORY_SIZE	X	X	X	O	O
PRIVATE_STATIC_AREA_INIT_SIZE	X	X	O	O	O
PRIVATE_STATIC_AREA_NEXT_SIZE	X	X	O	O	O
PRIVATE_STATIC_AREA_SHRINK_THRESHOLD	X	X	O	O	O
PRIVATE_STATIC_AREA_SIZE	O	O	O	O	O
PROCESS_MAX_COUNT	O	O	O	O	O
QUERY_TIMEOUT	O	O	O	O	O
READABLE_ARCHIVELOG_DIR_COUNT	O	O	O	O	O
READABLE_BACKUP_DIR_COUNT	O	O	O	O	O
REBALANCE_BLOCK_READ_COUNT	X	O	O	O	O
REBALANCE_SHARD_DIVISOR	X	X	X	X	O
RECOMPILE_CHECK_MINIMUM_PAGE_COUNT	O	X	X	X	X
RECOMPILE_PAGE_PERCENT	O	X	X	X	X
RECOVERY_LOG_BUFFER_SIZE	X	O	O	O	O
RECYCLEBIN	X	X	O	O	O
REDO_LOG_COMPRESSION_THRESHOLD	X	O	O	O	O
REFINE_RELATION	O	O	O	O	O
SESSION_FATAL_BEHAVIOR	O	O	O	O	O
SESSION_MEMORY_INIT_SIZE	X	O	O	O	O
SESSION_MEMORY_SHRINK_THRESHOLD	X	O	O	O	O
SESSION_POOL_INIT_SIZE	X	X	X	X	O
SESSION_POOL_NEXT_SIZE	X	X	X	X	O
SHARED_MEMORY_ADDRESS	O	O	O	O	O
SHARED_MEMORY_STATIC_KEY	O	O	O	O	O
SHARED_MEMORY_STATIC_NAME	O	O	O	O	O
SHARED_MEMORY_STATIC_SIZE	O	O	O	O	O
SHARED_REQUEST_QUEUE_COUNT	O	O	O	O	O
SHARED_SERVERS	O	O	O	O	O
SHARED_SESSION	O	O	O	O	O
SNAPSHOT_STATEMENT_TIMEOUT	O	O	O	O	O
SQL_HISTORY_SIZE	X	O	O	O	O
SUPPLEMENTAL_LOG_DATA_PRIMARY_KEY	O	O	O	O	O
SYNC_DISPATCHER_CM_BUFFER_COUNT	X	X	X	X	O
SYSTEM_DISK_DATA_TABLESPACE_SIZE	X	X	O	O	O
SYSTEM_FILE_IO	O	O	O	O	O
SYSTEM_MEMORY_AUX_TABLESPACE_SIZE	X	O	O	O	O
SYSTEM_MEMORY_DATA_TABLESPACE_SIZE	O	O	O	O	O
SYSTEM_MEMORY_DICT_TABLESPACE_SIZE	O	O	O	O	O
SYSTEM_MEMORY_TEMP_TABLESPACE_SIZE	O	O	O	O	O

Feature	2.x	3.x	20c.1	21c.1	22c.1
SYSTEM_MEMORY_UNDO_TABLESPACE_SIZE	O	O	O	O	O
SYSTEM_TABLESPACE_DIR	O	O	O	O	O
SYSTEM_UDS_DIR	X	O	O	O	O
TCP_NODELAY	X	O	O	O	O
TEMP_SEGMENT_CACHE_SIZE	X	O	O	O	O
TEMP_UNDO_ENABLED	X	O	O	O	O
TIMED_STATISTICS	X	O	O	O	O
TIMER_INTERVAL	O	X	X	O	O
TIMEZONE	O	O	O	O	O
TRACE_ALTER_SYSTEM	O	O	O	O	O
TRACE_DDL	O	O	O	O	O
TRACE_LOG_ID	O	O	O	O	O
TRACE_LOG_MSGBUG_SIZE	X	O	O	O	O
TRACE_LOG_TIME_DETAIL	O	O	O	O	O
TRACE_LOGGER	X	O	O	O	O
TRACE_LOGGER_REMOTE_HOST	X	O	O	O	O
TRACE_LOGGER_REMOTE_PORT	X	O	O	O	O
TRACE_LOGIN	O	O	O	O	O
TRACE_LONG_RUN_CURSOR	O	O	O	O	O
TRACE_LONG_RUN_SQL	O	O	O	O	O
TRACE_LONG_RUN_TIMER	X	X	O	O	O
TRACE_SYSTEM_DIR	X	X	X	X	O
TRACE_XA	O	O	O	O	O
TRANSACTION_ALLOCATION_TIMEOUT	X	O	O	O	O
TRANSACTION_COMMIT_WRITE_MODE	O	O	O	O	O
TRANSACTION_MAXIMUM_UNDO_PAGE_COUNT	O	O	O	O	O
TRANSACTION_TABLE_SIZE	O	O	O	O	O
TRANSACTION_TIMEOUT	X	O	O	O	O
UNDO_RELATION_ALLOCATION_TIMEOUT	X	O	O	O	O
UNDO_RELATION_COUNT	O	O	O	O	O
UNDO_SHRINK_THRESHOLD	O	O	O	O	O
USE_LARGE_PAGES	X	X	O	O	O
USER_DATA_TABLESPACE_MEDIA_TYPE	X	X	O	O	O
USER_DATA_TABLESPACE_SIZE	X	X	O	O	O
USER_DISK_DATA_TABLESPACE_NEXTSIZE	X	X	O	O	O
USER_TEMP_TABLESPACE_SIZE	O	O	O	O	O
XA_TRANSACTION_IDLE_TIMEOUT	X	X	O	O	O

Property Alias

The following is a feature matrix for property alias.

Table 4-9 Feature matrix for property alias

Feature	2.x	3.x	20c.1	21c.1	22c.1
CDISPATCHER_THREADS	X	O	O	O	O
CLUSTER_COMMIT_SLAVES	X	O	O	O	O
CLUSTER_SEREVER_RESPONSE_QUEUE_SIZE	X	O	O	O	O
CSERVER	X	O	O	O	O
INCREMENTAL_CHECKPOINT_CRITERIA	X	X	X	X	O
LOCKLESS_CSERVERS	X	X	O	O	O
MEMORY_MERGE_RUN_COUNT	O	O	O	O	O
MEMORY_SORT_RUN_SIZE	O	O	O	O	O
SYSTEM_LOGGER_DIR	O	O	O	O	O

SQL

SQL Element

Data Type

The following is a feature matrix for data type.

Table 4-10 Feature matrix for data type

Type	Feature	2.x	3.x	20c.1	21c.1	22c.1
Character string type	CHAR	O	O	O	O	O
	VARCHAR	O	O	O	O	O
	LONG VARCHAR	O	O	O	O	O
Binary string type	BINARY	O	O	O	O	O
	VARBINARY	O	O	O	O	O
	LONG VARBINARY	O	O	O	O	O
Decimal number type	SMALLINT	O	O	O	O	O
	INTEGER	O	O	O	O	O
	BIGINT	O	O	O	O	O
	NUMERIC	O	O	O	O	O
	DECIMAL	X	O	O	O	O
	NUMBER	O	O	O	O	O
	REAL	O	O	O	O	O
	DOUBLE PRECISION	O	O	O	O	O
	FLOAT	O	O	O	O	O
Binary number type	NATIVE_SMALLINT	O	O	O	O	O
	NATIVE_INTEGER	O	O	O	O	O
	NATIVE_BIGINT	O	O	O	O	O
	NATIVE_REAL	O	O	O	O	O
	NATIVE_DOUBLE	O	O	O	O	O
BOOLEAN type	BOOLEAN	O	O	O	O	O
Date/ time type	DATE	O	O	O	O	O
	TIME	O	O	O	O	O
	TIME WITH TIME ZONE	O	O	O	O	O
	TIMESTAMP	O	O	O	O	O
	TIMESTAMP WITH TIME ZONE	O	O	O	O	O
	INTERVAL YEAR TO MONTH	O	O	O	O	O
	INTERVAL YEAR	O	O	O	O	O
	INTERVAL MONTH	O	O	O	O	O
	INTERVAL DAY TO SECOND	O	O	O	O	O

Type	Feature	2.x	3.x	20c.1	21c.1	22c.1
INTERVAL type	INTERVAL DAY	O	O	O	O	O
	INTERVAL HOUR	O	O	O	O	O
	INTERVAL MINUTE	O	O	O	O	O
	INTERVAL SECOND	O	O	O	O	O
	INTERVAL DAY TO HOUR	O	O	O	O	O
	INTERVAL DAY TO MINUTE	O	O	O	O	O
	INTERVAL HOUR TO MINUTE	O	O	O	O	O
	INTERVAL HOUR TO SECOND	O	O	O	O	O
	INTERVAL MINUTE TO SECOND	O	O	O	O	O
ROWID type	ROWID	O	O	O	O	O

Function

The following is a feature matrix for function.

Table 4-11 Feature matrix for function

Feature	2.x	3.x	20c.1	21c.1	22c.1
expr1 * expr2	O	O	O	O	O
expr1 + expr2	O	O	O	O	O
datetime + interval	O	O	O	O	O
+ expr	O	O	O	O	O
expr1 - expr2	O	O	O	O	O
datetime - interval	O	O	O	O	O
- expr	O	O	O	O	O
expr1 / expr2	O	O	O	O	O
str1 || str2	O	O	O	O	O
expr <comp> expr	O	O	O	O	O
expr <comp> (subquery)	O	O	O	O	O
(subquery) <comp> expr	O	O	O	O	O
(subquery) <comp> (subquery)	O	O	O	O	O
(expr, ...) <comp> (expr, ...)	O	O	O	O	O
(expr, ...) <comp> (subquery)	O	O	O	O	O
(subquery) <comp> (expr, ...)	O	O	O	O	O
expr <comp> {ALL|ANY|SOME} (expr, ...)	O	O	O	O	O
expr <comp> {ALL|ANY|SOME} (subquery)	O	O	O	O	O
(subquery) <comp> {ALL|ANY|SOME} (expr, ...)	O	O	O	O	O
(subquery) <comp> {ALL|ANY|SOME} (subquery)	O	O	O	O	O
(expr, ...) <comp> {ALL|ANY|SOME} (expr_list, ...)	O	O	O	O	O
(expr, ...) <comp> {ALL|ANY|SOME} (subquery)	O	O	O	O	O
(subquery) <comp> {ALL|ANY|SOME} (expr_list, ...)	O	O	O	O	O

Feature	2.x	3.x	20c.1	21c.1	22c.1
ABS(num)	O	O	O	O	O
ACOS(num)	O	O	O	O	O
ADDDATE(date, interval)	O	O	O	O	O
ADDDATE(expr, days)	O	O	O	O	O
ADDTIME(expr1, expr2)	O	O	O	O	O
ADD_MONTHS(date, number)	O	O	O	O	O
AND	O	O	O	O	O
ASCII(char)	X	O	O	O	O
ASIN(num)	O	O	O	O	O
ATAN(num)	O	O	O	O	O
ATAN2(num1, num2)	O	O	O	O	O
AVG(num)	O	O	O	O	O
AVG(expr) OVER	X	X	X	X	O
expr1 [NOT] BETWEEN [ASYMMETRIC|SYMMETRIC] expr2 AND expr3	O	O	O	O	O
BITAND(num1, num2)	O	O	O	O	O
BITNOT(num)	O	O	O	O	O
BITOR(num1, num2)	O	O	O	O	O
BITXOR(num1, num2)	O	O	O	O	O
BIT_LENGTH(str)	O	O	O	O	O
BYTE_LENGTH(str)	O	O	O	O	O
CASE .. WHEN .. THEN .. ELSE .. END	O	O	O	O	O
CASE2(condition, result, ...)	O	O	O	O	O
CAST(expr AS datatype)	O	O	O	O	O
CBRT(num)	O	O	O	O	O
CEIL(num)	O	O	O	O	O
CEILING(num)	O	O	O	O	O
CHAR_LENGTH(str)	O	O	O	O	O
CHARACTER_LENGTH(str)	O	O	O	O	O
CHR(num)	X	O	O	O	O
CLOCK_DATE()	O	O	O	O	O
CLOCK_LOCALTIME()	O	O	O	O	O
CLOCK_LOCALTIMESTAMP()	O	O	O	O	O
CLOCK_TIME()	O	O	O	O	O
CLOCK_TIMESTAMP()	O	O	O	O	O
COALESCE(expr1, ..., exprN)	O	O	O	O	O
CONCAT(str1, str2)	O	O	O	O	O
CONCATENATE(str1, str2)	O	O	O	O	O
CONNECT_BY_ISCYCLE	X	X	X	O	O
CONNECT_BY_ISLEAF	X	X	X	O	O

Feature	2.x	3.x	20c.1	21c.1	22c.1
CONNECT_BY_ROOT expr	X	X	X	O	O
CORR(expr1, expr2) OVER	X	X	X	X	O
COS(num)	O	O	O	O	O
COT(num)	O	O	O	O	O
COUNT(expr)	O	O	O	O	O
COUNT(expr) OVER	X	X	X	X	O
COUNT(*)	O	O	O	O	O
COUNT(*) OVER	X	X	X	X	O
COVAR_POP(expr1, expr2) OVER	X	X	X	X	O
COVAR_SAMP(expr1, expr2) OVER	X	X	X	X	O
CUME_DIST() OVER	X	X	X	X	O
CURRENT_CATALOG	O	O	O	O	O
CURRENT_DATE	O	O	O	O	O
CURRENT_SCHEMA	O	O	O	O	O
CURRENT_TIME	O	O	O	O	O
CURRENT_TIMESTAMP	O	O	O	O	O
CURRENT_USER	O	O	O	O	O
seq.CURRVAL	O	O	O	O	O
CURRVAL(seq)	O	O	O	O	O
DATEADD(datepart, number, date)	O	O	O	O	O
DATEDIFF(datepart, startdate, enddate)	O	O	O	O	O
DATE_ADD(date, interval)	O	O	O	O	O
DATE_PART(field, datetime)	O	O	O	O	O
DECODE(expr, comparison, result, ...)	O	O	O	O	O
DEGREES(radians)	O	O	O	O	O
DENSE_RANK() OVER	X	X	X	X	O
DIGEST (data, type)	X	O	O	O	O
expr IS [NOT] DISTINCT FROM expr	X	X	X	X	O
(expr, ...) IS [NOT] DISTINCT FROM (expr, ...)	X	X	X	X	O
DUMP(expr)	O	O	O	O	O
EXISTS(subquery)	O	O	O	O	O
EXP(num)	O	O	O	O	O
EXTRACT(field FROM datetime)	O	O	O	O	O
FACTORIAL(num)	O	O	O	O	O
FIRST : aggr_func KEEP (DENSE_RANK FIRST ORDER BY expr, ...) OVER	X	X	X	X	O
FIRST_VALUE(expr) OVER	X	X	X	X	O
FLOOR(num)	O	O	O	O	O
FROM_BASE64(str)	X	O	O	O	O
FROM_TZ(timestamp, timezone)	X	X	X	O	O

Feature	2.x	3.x	20c.1	21c.1	22c.1
GREATEST(expr, ...)	O	O	O	O	O
HEX(str)	X	O	O	O	O
expr1 [NOT] IN (expr, ...)	O	O	O	O	O
expr1 [NOT] IN (subquery)	O	O	O	O	O
subquery [NOT] IN (<expr_list>)	O	O	O	O	O
subquery [NOT] IN (subquery)	O	O	O	O	O
<expr_list> [NOT] IN (<expr_list>, ...)	O	O	O	O	O
<expr_list> [NOT] IN (subquery)	O	O	O	O	O
subquery [NOT] IN (<expr_list>, ...)	O	O	O	O	O
INITCAP(str)	O	O	O	O	O
INSTR(str, substr, ...)	O	O	O	O	O
IS NOT NULL	O	O	O	O	O
IS NULL	O	O	O	O	O
LAG(expr [, offset [, default]]) OVER	X	X	X	X	O
LAST : aggr_func KEEP (DENSE_RANK LAST ORDER BY expr, ...) OVER	X	X	X	X	O
LAST_DAY(date)	O	O	O	O	O
LAST_IDENTITY_VALUE()	X	O	O	O	O
LAST_VALUE(expr) OVER	X	X	X	X	O
LEAD(expr [, offset [, default]]) OVER	X	X	X	X	O
LEAST(expr, ...)	O	O	O	O	O
LENGTH(str)	O	O	O	O	O
LENGTHB(str)	O	O	O	O	O
LEVEL	X	X	X	O	O
string [NOT] LIKE pattern ESCAPE escape_char	O	O	O	O	O
LISTAGG(str [, delimiter]) OVER	X	X	X	X	O
LN(num)	O	O	O	O	O
LNNVL(expr)	X	X	O	O	O
LOCALTIME	O	O	O	O	O
LOCALTIMESTAMP	O	O	O	O	O
LOCAL_GROUP_ID()	X	O	O	O	O
LOCAL_GROUP_NAME()	X	O	O	O	O
LOCAL_MEMBER_ID()	X	O	O	O	O
LOCAL_MEMBER_NAME()	X	O	O	O	O
LOG(num2)	O	O	O	O	O
LOG(num1, num2)	O	O	O	O	O
LOGON_USER()	O	O	O	O	O
LOWER(str)	O	O	O	O	O
LPAD(str, length, fill)	O	O	O	O	O
LTRIM(str, [str])	O	O	O	O	O

Feature	2.x	3.x	20c.1	21c.1	22c.1
MAX(expr)	O	O	O	O	O
MAX(expr) OVER	X	X	X	X	O
MEDIAN(expr) OVER	X	X	X	X	O
MIN(expr)	O	O	O	O	O
MIN(expr) OVER	X	X	X	X	O
MOD(num1, num2)	O	O	O	O	O
MONTHS_BETWEEN(date1, date2)	X	O	O	O	O
NEXT_DAY(date, day)	X	O	O	O	O
seq.NEXTVAL	O	O	O	O	O
NEXTVAL(seq)	O	O	O	O	O
NEXT VALUE FOR seq	O	O	O	O	O
NOT	O	O	O	O	O
NTH_VALUE(expr, n) OVER	X	X	X	X	O
NTILE(expr) OVER	X	X	X	X	O
NULLIF(expr1, expr2)	O	O	O	O	O
NUMTODSINTERVAL(num, interval_indicator)	X	X	O	O	O
NUMTOYMINTERVAL(num, interval_indicator)	X	X	O	O	O
NVL(expr1, expr2)	O	O	O	O	O
NVL2(expr1, expr2, expr3)	O	O	O	O	O
OCTET_LENGTH(str)	O	O	O	O	O
OVERLAY(str1 PLACING str2 FROM start FOR length)	O	O	O	O	O
OR	O	O	O	O	O
PERCENT_RANK() OVER	X	X	X	X	O
PERCENTILE_CONT(expr) OVER	X	X	X	X	O
PERCENTILE_DISC(expr) OVER	X	X	X	X	O
PHYSICAL_LENGTH(expr)	X	X	O	O	O
PI()	O	O	O	O	O
POSITION(str1 IN str2)	O	O	O	O	O
POWER(num1, num2)	O	O	O	O	O
PRIOR expr	X	X	X	O	O
RADIANS(degrees)	O	O	O	O	O
RANDOM(min, max)	O	O	O	O	O
RANK() OVER	X	X	X	X	O
RATIO_TO_REPORT(expr) OVER	X	X	X	X	O
REGR_AVGX(expr1, expr2) OVER	X	X	X	X	O
REGR_AVGY(expr1, expr2) OVER	X	X	X	X	O
REGR_COUNT(expr1, expr2) OVER	X	X	X	X	O
REGR_INTERCEPT(expr1, expr2) OVER	X	X	X	X	O
REGR_R2(expr1, expr2) OVER	X	X	X	X	O
REGR_SLOPE(expr1, expr2) OVER	X	X	X	X	O

Feature	2.x	3.x	20c.1	21c.1	22c.1
REGR_SXX(expr1, expr2) OVER	X	X	X	X	O
REGR_SXY(expr1, expr2) OVER	X	X	X	X	O
REGR_SYY(expr1, expr2) OVER	X	X	X	X	O
REPEAT(str, num)	O	O	O	O	O
REPLACE(str, from, to)	O	O	O	O	O
REVERSE(str)	X	O	O	O	O
ROUND(num)	O	O	O	O	O
ROUND(date, fmt)	O	O	O	O	O
ROW_NUMBER() OVER	X	X	X	X	O
ROWID_GRID_BLOCK_ID(rowid)	X	O	O	O	O
ROWID_GRID_BLOCK_SEQ(rowid)	X	O	O	O	O
ROWID_MEMBER_ID(rowid)	X	O	O	O	O
ROWID_OBJECT_ID(rowid)	O	O	O	O	O
ROWID_PAGE_ID(rowid)	O	O	O	O	O
ROWID_ROW_NUMBER(rowid)	O	O	O	O	O
ROWID_SHARD_ID(rowid)	X	O	O	O	O
ROWID_TABLESPACE_ID(rowid)	O	O	O	O	O
ROWNUM	X	O	O	O	O
RPAD(str, length, fill)	O	O	O	O	O
RTRIM(str, [str])	O	O	O	O	O
SESSION_ID()	O	O	O	O	O
SESSION_SERIAL()	O	O	O	O	O
SESSION_USER	O	O	O	O	O
SESSIONTIMEZONE()	X	X	X	X	O
SHARD_GROUP_ID(table, expr)	X	O	O	O	O
SHARD_GROUP_NAME(table_name, shard_key_value [, ...])	X	O	O	O	O
SHARD_ID(table, expr)	X	O	O	O	O
SHARD_NAME(table_name, shard_key_value [, ...])	X	O	O	O	O
SHIFT_LEFT(num, cnt)	O	O	O	O	O
SHIFT_RIGHT(num, cnt)	O	O	O	O	O
SIGN(num)	O	O	O	O	O
SIN(num)	O	O	O	O	O
SPLIT_PART(str, delimiter, field)	O	O	O	O	O
SQRT(num)	O	O	O	O	O
STATEMENT_DATE()	O	O	O	O	O
STATEMENT_LOCALTIME()	O	O	O	O	O
STATEMENT_LOCALTIMESTAMP()	O	O	O	O	O
STATEMENT_TIME()	O	O	O	O	O
STATEMENT_TIMESTAMP()	O	O	O	O	O
STATEMENT_VIEW_SCN()	O	O	O	O	O

Feature	2.x	3.x	20c.1	21c.1	22c.1
STATEMENT_VIEW_SCN_DCN()	X	O	O	O	O
STATEMENT_VIEW_SCN_GCN()	X	O	O	O	O
STATEMENT_VIEW_SCN_LCN()	X	O	O	O	O
STDDEV([ALL | DISTINCT] expr)	X	O	O	O	O
STDDEV(expr) OVER	X	X	X	X	O
STDDEV_POP(expr)	X	O	O	O	O
STDDEV_POP(expr) OVER	X	X	X	X	O
STDDEV_SAMP(expr)	X	O	O	O	O
STDDEV_SAMP(expr) OVER	X	X	X	X	O
STRING_AGG(str [, delimiter]) OVER	X	X	X	X	O
SUBSTR(str FROM start FOR length)	O	O	O	O	O
SUBSTR(str, start, length)	O	O	O	O	O
SUBSTRB(str, start, length)	O	O	O	O	O
SUBSTRING(str FROM start FOR length)	O	O	O	O	O
SUBSTRING(str, start, length)	O	O	O	O	O
SUM(expr)	O	O	O	O	O
SUM(expr) OVER	X	X	X	X	O
SYSDATE	O	O	O	O	O
SYS_CONNECT_BY_PATH(expr, 'string')	X	X	X	O	O
SYS_EXTRACT_UTC(datetime_with_timezone)	X	O	O	O	O
SYSTIME	O	O	O	O	O
SYSTIMESTAMP	O	O	O	O	O
TAN(num)	O	O	O	O	O
TO_CHAR(datetime, fmt)	O	O	O	O	O
TO_CHAR(number, fmt)	O	O	O	O	O
TO_BASE64(str)	X	O	O	O	O
TO_DATE(str, fmt)	O	O	O	O	O
TO_NATIVE_BIGINT(str, fmt)	X	X	O	O	O
TO_NATIVE_DOUBLE(str, fmt)	O	O	O	O	O
TO_NATIVE_INTEGER(str, fmt)	X	X	O	O	O
TO_NATIVE_REAL(str, fmt)	O	O	O	O	O
TO_NATIVE_SMALLINT(str, fmt)	X	X	O	O	O
TO_NUMBER(num, fmt)	O	O	O	O	O
TO_TIME(str, fmt)	O	O	O	O	O
TO_TIME_TZ(str, fmt)	O	O	O	O	O
TO_TIME_WITH_TIME_ZONE(str, fmt)	O	O	O	O	O
TO_TIMESTAMP(str, fmt)	O	O	O	O	O
TO_TIMESTAMP_TZ(str, fmt)	O	O	O	O	O
TO_TIMESTAMP_WITH_TIME_ZONE(str, fmt)	O	O	O	O	O
TRANSACTION_DATE()	O	O	O	O	O

Feature	2.x	3.x	20c.1	21c.1	22c.1
TRANSACTION_LOCALTIME()	O	O	O	O	O
TRANSACTION_LOCALTIMESTAMP()	O	O	O	O	O
TRANSACTION_TIME()	O	O	O	O	O
TRANSACTION_TIMESTAMP()	O	O	O	O	O
TRANSLATE(str, from, to)	O	O	O	O	O
TRIM(LEADING|TRAILING|BOTH trim_char FROM source)	O	O	O	O	O
TRUNC(num, scale)	O	O	O	O	O
TRUNC(date, fmt)	O	O	O	O	O
UPPER(str)	O	O	O	O	O
UNHEX(str)	X	O	O	O	O
UNHEX_TO_CHARSTR(str)	X	O	O	O	O
USER_ID()	O	O	O	O	O
UUID()	X	O	O	O	O
VAR_POP(expr)	X	O	O	O	O
VAR_POP(expr) OVER	X	X	X	X	O
VAR_SAMP(expr)	X	O	O	O	O
VAR_SAMP(expr) OVER	X	X	X	X	O
VARIANCE([ALL | DISTINCT] expr)	X	O	O	O	O
VARIANCE(expr) OVER	X	X	X	X	O
VERSION()	O	O	O	O	O
WIDTH_BUCKET(num, min, max, cnt)	O	O	O	O	O

Object

SQL Object

The following is a feature matrix for DDL which creates/ drops/ alters an SQL object.

Table 4-12 Feature matrix for SQL object DDL

Object	Feature	2.x	3.x	20c.1	21c.1	22c.1
Database object	ALTER DATABASE ARCHIVELOG	O	O	O	O	O
	ALTER DATABASE ADD LOGFILE	O	O	O	O	O
	ALTER DATABASE DROP LOGFILE	O	O	O	O	O
	ALTER DATABASE RENAME GLOBAL TRANSACTION LOGFILE	X	X	X	X	O
	ALTER DATABASE RENAME LOGFILE	O	O	O	O	O
	ALTER DATABASE BEGIN/END BACKUP	O	O	O	O	O
	ALTER DATABASE RECOVER	O	O	O	O	O
	ALTER DATABASE RECOVER TABLESPACE	O	O	O	O	O
	ALTER DATABASE REGISTER	O	O	O	O	O
	ALTER DATABASE RESTORE	O	O	O	O	O
	ANALYZE SYSTEM	X	O	O	O	O
	COMMENT ON object IS ..	O	O	O	O	O
Profile object	CREATE PROFILE	O	O	O	O	O
	DROP PROFILE	O	O	O	O	O
	ALTER PROFILE	O	O	O	O	O
Audit policy object	CREATE AUDIT POLICY	X	O	O	O	O
	DROP AUDIT POLICY	X	O	O	O	O
	ALTER AUDIT POLICY	X	O	O	O	O
	AUDIT POLICY	X	O	O	O	O
	NOAUDIT POLICY	X	O	O	O	O
Authorization object	CREATE USER	O	O	O	O	O
	DROP USER	O	O	O	O	O
	ALTER USER	O	O	O	O	O
	GRANT privileges TO	O	O	O	O	O
	REVOKE privileges FROM	O	O	O	O	O
Schema object	CREATE SCHEMA	O	O	O	O	O
	DROP SCHEMA	O	O	O	O	O
Tablespace object	CREATE MEMORY DATA TABLESPACE	O	O	O	O	O
	CREATE MEMORY TEMPORARY TABLESPACE	O	O	O	O	O
	DROP TABLESPACE	O	O	O	O	O
	ALTER TABLESPACE .. RENAME TO	O	O	O	O	O
	ALTER TABLESPACE .. BEGIN/END BACKUP	O	O	O	O	O

Object	Feature	2.x	3.x	20c.1	21c.1	22c.1
	ALTER TABLESPACE .. ADD [DATAFILE|MEMORY]	O	O	O	O	O
	ALTER TABLESPACE .. DROP [DATAFILE|MEMORY]	O	O	O	O	O
	ALTER TABLESPACE .. RENAME DATAFILE	O	O	O	O	O
	ALTER TABLESPACE .. { ONLINE | OFFLINE }	O	O	O	O	O
Table object	CREATE TABLE	O	O	O	O	O
	CREATE TABLE AS SELECT	O	O	O	O	O
	CREATE GLOBAL TEMPORARY TABLE	X	O	O	O	O
	CREATE GLOBAL TEMPORARY TABLE AS SELECT	X	O	O	O	O
	CREATE IMMUTABLE TABLE	X	X	O	O	O
	CREATE IMMUTABLE TABLE AS SELECT	X	X	O	O	O
	DROP TABLE	O	O	O	O	O
	TRUNCATE TABLE	O	O	O	O	O
	ALTER TABLE .. STORAGE	O	O	O	O	O
	ALTER TABLE .. RENAME TO	O	O	O	O	O
	ALTER TABLE .. ADD COLUMN	O	O	O	O	O
	ALTER TABLE .. SET UNUSED COLUMN	O	O	O	O	O
	ALTER TABLE .. ALTER COLUMN	O	O	O	O	O
	ALTER TABLE .. RENAME COLUMN	O	O	O	O	O
	ALTER TABLE .. RENAME CONSTRAINT	X	O	O	O	O
	ALTER TABLE .. ADD CONSTRAINT	O	O	O	O	O
	ALTER TABLE .. DROP CONSTRAINT	O	O	O	O	O
	ALTER TABLE .. ALTER CONSTRAINT	O	O	O	O	O
	ALTER TABLE .. ADD SUPPLEMENTAL LOG	O	O	O	O	O
	ALTER TABLE .. DROP SUPPLEMENTAL LOG	O	O	O	O	O
	ALTER TABLE .. READ { ONLY | WRITE }	X	O	O	O	O
	ANALYZE TABLE	X	O	O	O	O
	FLASHBACK TABLE	X	X	O	O	O
	PURGE	X	X	O	O	O
View object	CREATE VIEW	O	O	O	O	O
	DROP VIEW	O	O	O	O	O
	ALTER VIEW	O	O	O	O	O
Index object	CREATE INDEX	O	O	O	O	O
	DROP INDEX	O	O	O	O	O
	ALTER INDEX .. AGING	X	O	O	O	O
	ALTER INDEX .. STORAGE	O	O	O	O	O
	ALTER INDEX .. RENAME	X	O	O	O	O
	ALTER INDEX .. REBUILD	X	X	O	O	O
	ALTER INDEX .. COALESCE	X	X	X	X	O
Sequence	CREATE SEQUENCE	O	O	O	O	O
	DROP SEQUENCE	O	O	O	O	O

Object	Feature	2.x	3.x	20c.1	21c.1	22c.1
object	ALTER SEQUENCE	O	O	O	O	O
Synonym object	CREATE SYNONYM	O	O	O	O	O
	DROP SYNONYM	O	O	O	O	O
	CREATE PUBLIC SYNONYM	O	O	O	O	O
	DROP PUBLIC SYNONYM	O	O	O	O	O
Stored procedure object	CREATE PROCEDURE	X	O	O	O	O
	DROP PROCEDURE	X	O	O	O	O
	ALTER PROCEDURE	X	O	O	O	O
Stored function object	CREATE FUNCTION	X	O	O	O	O
	DROP FUNCTION	X	O	O	O	O
	ALTER FUNCTION	X	O	O	O	O
Package object	CREATE PACKAGE	X	X	O	O	O
	CREATE PACKAGE BODY	X	X	O	O	O
	ALTER PACKAGE	X	X	O	O	O
	DROP PACKAGE	X	X	O	O	O

Cluster Object

The following is a feature matrix for DDL which creates/ drops/ alters a cluster object.

Table 4-13 Feature matrix for cluster object DDL

Object	Feature	2.x	3.x	20c.1	21c.1	22c.1
Cluster system object	ALTER DATABASE REBALANCE	X	O	O	O	O
	ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS	X	O	O	O	O
	ALTER DATABASE DROP OFFLINE SEGMENTS	X	X	X	X	O
	ALTER DATABASE SYNCHRONIZE	X	X	X	X	O
Cluster group object	CREATE CLUSTER GROUP	X	O	O	O	O
	DROP CLUSTER GROUP	X	O	O	O	O
Cluster member object	ALTER CLUSTER GROUP name ADD MEMBER	X	O	O	O	O
	ALTER CLUSTER GROUP name OFFLINE MEMBER	X	O	O	O	O
	ALTER DATABASE RESET LOCAL CLUSTER MEMBER	X	O	O	O	O
	ALTER SYSTEM IRRECOVERABLE CLUSTER MEMBER	X	O	O	O	O
	ALTER SYSTEM JOIN DATABASE	X	O	O	O	O
Cluster location object	CREATE CLUSTER LOCATION	X	O	O	O	O
	DROP CLUSTER LOCATION	X	O	O	O	O
	ALTER CLUSTER LOCATION	X	O	O	O	O
Cluster table and shard object	ALTER TABLE name REBALANCE	X	O	O	O	O
	ALTER TABLE name DROP OFFLINE SEGMENTS	X	X	X	X	O
	ALTER TABLE name SYNCHRONIZE	X	X	X	X	O
	ALTER TABLE name MERGE SHARDS	X	X	O	O	O
	ALTER TABLE name MOVE SHARD	X	O	O	O	O
	ALTER TABLE name SPLIT SHARD	X	O	O	O	O
	ALTER TABLE name RENAME SHARD	X	O	O	O	O
Global secondary index object	ALTER TABLE name ADD GLOBAL SECONDARY INDEX	X	O	O	O	O
	ALTER TABLE name DROP GLOBAL SECONDARY INDEX	X	O	O	O	O
	ALTER TABLE name ALTER GLOBAL SECONDARY INDEX	X	O	O	O	O
	ALTER TABLE name ALTER GLOBAL SECONDARY INDEX REBUILD	X	X	O	O	O
	ALTER TABLE name ALTER GLOBAL SECONDARY INDEX COALESCE	X	X	X	X	O

SQL Language

DML

The following is a feature matrix for DML which manipulates data.

Table 4-14 Feature matrix for DML

Feature	2.x	3.x	20c.1	21c.1	22c.1
INSERT INTO ..	O	O	O	O	O
INSERT INTO .. RETURNING query	O	O	O	O	O
INSERT INTO .. RETURNING .. INTO ..	O	O	O	O	O
INSERT INTO .. UPDATE	X	X	X	O	O
INSERT INTO .. UPDATE .. RETURNING ..	X	X	X	O	O
INSERT INTO .. UPDATE .. RETURNING .. INTO ..	X	X	X	O	O
DELETE FROM ..	O	O	O	O	O
DELETE FROM .. RETURNING query	O	O	O	O	O
DELETE FROM .. RETURNING .. INTO ..	O	O	O	O	O
DELETE FROM .. WHERE CURRENT OF cursor	O	O	O	O	O
UPDATE ..	O	O	O	O	O
UPDATE .. RETURNING query	O	O	O	O	O
UPDATE .. RETURNING .. INTO ..	O	O	O	O	O
UPDATE .. WHERE CURRENT OF cursor	O	O	O	O	O
CALL proc_name	X	O	O	O	O

Query

The following is a feature matrix for SELECT statement which enquires data.

Table 4-15 Feature matrix for SELECT

Feature	2.x	3.x	20c.1	21c.1	22c.1
〈query expression〉	O	O	O	O	O
〈query specification〉	O	O	O	O	O
〈select list〉	O	O	O	O	O
〈from clause〉	O	O	O	O	O
〈joined table〉	O	O	O	O	O
〈where clause〉	O	O	O	O	O
〈group by clause〉	O	O	O	O	O
〈window clause〉	X	X	X	X	O
〈window partition clause〉	X	X	X	X	O
〈window order clause〉	X	X	X	X	O
〈window frame clause〉	X	X	X	X	O
〈window frame exclusion〉	X	X	X	X	O

Feature	2.x	3.x	20c.1	21c.1	22c.1
〈order by clause〉	O	O	O	O	O
〈offset limit clause〉	O	O	O	O	O
〈set operator〉	O	O	O	O	O
〈subquery〉	O	O	O	O	O
〈hint clause〉	O	O	O	O	O
〈with clause〉	X	X	X	O	O
〈search clause〉	X	X	X	O	O
〈cycle clause〉	X	X	X	O	O
〈start with clause〉	X	X	X	O	O
〈connect by clause〉	X	X	X	O	O
〈order siblings by clause〉	X	X	X	O	O

Control Language

The following is a feature matrix for control statement.

Table 4-16 Feature matrix for control statement

Control statement	Feature	2.x	3.x	20c.1	21c.1	22c.1
Transaction	COMMIT	O	O	O	O	O
	ROLLBACK	O	O	O	O	O
	SAVEPOINT	O	O	O	O	O
	RELEASE SAVEPOINT	O	O	O	O	O
	LOCK TABLE	O	O	O	O	O
	SET CONSTRAINTS	O	O	O	O	O
	SET TRANSACTION	O	O	O	O	O
Session	SET SESSION CHARACTERISTICS AS	O	O	O	O	O
	SET SESSION AUTHORIZATION	O	O	O	O	O
	SET SCHEMA	X	X	X	O	O
	SET TIME ZONE	O	O	O	O	O
	ALTER SESSION SET property	O	O	O	O	O
System	ALTER SYSTEM {OPEN|MOUNT} DATABASE	O	O	O	O	O
	ALTER SYSTEM CHECKPOINT	O	O	O	O	O
	ALTER SYSTEM KILL SESSION	O	O	O	O	O
	ALTER SYSTEM RECONNECT GLOBAL CONNECTION	X	O	O	O	O
	ALTER SYSTEM SWITCH LOGFILE	O	O	O	O	O
	ALTER SYSTEM SET property	O	O	O	O	O
	ALTER SYSTEM RESET property	O	O	O	O	O

PSM Language

The following is a feature matrix for Persistent Stored Module (PSM) language element.

Table 4-17 Feature matrix for Persistent Stored Module (PSM) language element

Feature	2.x	3.x	20c.1	21c.1	22c.1
Assignment Statement	X	O	O	O	O
Basic LOOP Statement	X	O	O	O	O
Block (BEGIN .. END)	X	O	O	O	O
CASE Statement	X	O	O	O	O
CLOSE Statement	X	O	O	O	O
Collection Method Invocation	X	O	O	O	O
Collection Variable Declaration	X	O	O	O	O
CONTINUE Statement	X	O	O	O	O
Cursor FOR LOOP Statement	X	O	O	O	O
Cursor Variable Declaration	X	O	O	O	O
DELETE Statement Extension	X	O	O	O	O
EXCEPTION_INIT Pragma	X	O	O	O	O
Exception Declaration	X	O	O	O	O
Exception Handler	X	O	O	O	O
EXECUTE IMMEDIATE Statement	X	O	O	O	O
EXIT Statement	X	O	O	O	O
Explicit Cursor Declaration and Definition	X	O	O	O	O
FETCH Statement	X	O	O	O	O
FOR LOOP Statement	X	O	O	O	O
GOTO Statement	X	O	O	O	O
IF Statement	X	O	O	O	O
Implicit Cursor Attribute	X	O	O	O	O
INSERT Statement Extension	X	O	O	O	O
Named Cursor Attribute	X	O	O	O	O
NULL Statement	X	O	O	O	O
OPEN Statement	X	O	O	O	O
OPEN FOR Statement	X	O	O	O	O
Procedure Call	X	O	O	O	O
Procedure Declaration and Definition	X	O	O	O	O
RAISE Statement	X	O	O	O	O
Record Variable Declaration	X	O	O	O	O
RETURN Statement	X	O	O	O	O
RETURN TABLE Statement	X	X	X	X	O
RETURNING INTO clause	X	O	O	O	O
%ROWTYPE Attribute	X	O	O	O	O

Feature	2.x	3.x	20c.1	21c.1	22c.1
Scalar Variable Declaration	X	O	O	O	O
SELECT INTO Statement	X	O	O	O	O
SQLCODE Function	X	O	O	O	O
SQLERRM Function	X	O	O	O	O
%TYPE Attribute	X	O	O	O	O
UPDATE Statement Extension	X	O	O	O	O
WHILE LOOP Statement	X	O	O	O	O

API

ODBC

The following is a feature matrix for the ODBC standard API.

Table 4-18 Feature matrix for the ODBC standard API

Feature	2.x	3.x	20c.1	21c.1	22c.1
SQLAllocHandle()	O	O	O	O	O
SQLBindCol()	O	O	O	O	O
SQLBindParameter()	O	O	O	O	O
SQLCloseCursor()	O	O	O	O	O
SQLColAttribute()	O	O	O	O	O
SQLColumnPrivileges()	O	O	O	O	O
SQLColumns()	O	O	O	O	O
SQLConnect()	O	O	O	O	O
SQLDescribeCol()	O	O	O	O	O
SQLDescribeParam()	O	O	O	O	O
SQLDisconnect()	O	O	O	O	O
SQLDriverConnect()	O	O	O	O	O
SQLEndTran()	O	O	O	O	O
SQLExecDirect()	O	O	O	O	O
SQLExecute()	O	O	O	O	O
SQLExtendedFetch()	O	O	O	O	O
SQLFetch()	O	O	O	O	O
SQLFetchScroll()	O	O	O	O	O
SQLForeignKeys()	O	O	O	O	O
SQLFreeHandle()	O	O	O	O	O
SQLFreeStmt()	O	O	O	O	O
SQLGetConnectAttr()	O	O	O	O	O
SQLGetCursorName()	O	O	O	O	O
SQLGetData()	O	O	O	O	O
SQLGetDescField()	O	O	O	O	O
SQLGetDescRec()	O	O	O	O	O
SQLGetDiagField()	O	O	O	O	O
SQLGetDiagRec()	O	O	O	O	O
SQLGetEnvAttr()	O	O	O	O	O
SQLGetFunctions()	O	O	O	O	O
SQLGetInfo()	O	O	O	O	O
SQLGetStmtAttr()	O	O	O	O	O

Feature	2.x	3.x	20c.1	21c.1	22c.1
SQLGetTypeInfo()	O	O	O	O	O
SQLMoreResults()	O	O	O	O	O
SQLNumParams()	O	O	O	O	O
SQLNumResultCols()	O	O	O	O	O
SQLParamData()	O	O	O	O	O
SQLPrepare()	O	O	O	O	O
SQLPrimaryKeys()	O	O	O	O	O
SQLProcedureColumns()	O	O	O	O	O
SQLProcedures()	O	O	O	O	O
SQLPutData()	O	O	O	O	O
SQLRowCount()	O	O	O	O	O
SQLSetConnectAttr()	O	O	O	O	O
SQLSetCursorName()	O	O	O	O	O
SQLSetDescField()	O	O	O	O	O
SQLSetDescRec()	O	O	O	O	O
SQLSetEnvAttr()	O	O	O	O	O
SQLSetPos()	O	O	O	O	O
SQLSetStmtAttr()	O	O	O	O	O
SQLSpecialColumns()	O	O	O	O	O
SQLStatistics()	O	O	O	O	O
SQLTablePrivileges()	O	O	O	O	O
SQLTables()	O	O	O	O	O

The following is a feature matrix for API other than the ODBC standard API.

Table 4-19 Feature matrix for API other than the ODBC standard

Feature	2.x	3.x	20c.1	21c.1	22c.1
xa_open	O	O	O	O	O
xa_close	O	O	O	O	O
xa_start	O	O	O	O	O
xa_end	O	O	O	O	O
xa_rollback	O	O	O	O	O
xa_prepare	O	O	O	O	O
xa_commit	O	O	O	O	O
xa_recover	O	O	O	O	O
xa_forget	O	O	O	O	O
SQLGetXaSwitch	O	O	O	O	O
SQLGetXaConnectionHandle	O	O	O	O	O
SQLGetGroupCount	X	O	O	O	O
SQLGetGroupIDs	X	O	O	O	O

Feature	2.x	3.x	20c.1	21c.1	22c.1
SQLGetGroupName	X	O	O	O	O
SQLGetSuitableGroupID	X	O	O	O	O

JDBC

The following is a class feature matrix for JDBC.

Table 4-20 Class feature matrix for JDBC

Feature	2.x	3.x	20c.1	21c.1	22c.1
CallableStatement	X	O	O	O	O
CommonDataSource	O	O	O	O	O
Connection	O	O	O	O	O
ConnectionPoolDataSource	O	O	O	O	O
DatabaseMetaData	O	O	O	O	O
DataSource	O	O	O	O	O
Driver	O	O	O	O	O
ParameterMetaData	O	O	O	O	O
PooledConnection	O	O	O	O	O
PreparedStatement	O	O	O	O	O
ResultSet	O	O	O	O	O
ResultSetMetaData	O	O	O	O	O
RowId	O	O	O	O	O
Savepoint	O	O	O	O	O
Statement	O	O	O	O	O
XAConnection	O	O	O	O	O
XADataSource	O	O	O	O	O
XAResource	O	O	O	O	O
GoldilocksInterval	O	O	O	O	O
GoldilocksTypes	O	O	O	O	O

Embedded SQL

Precompiler Option

The following is a feature matrix for precompiler option.

Table 4-21 Feature matrix for precompiler option

Feature	2.x	3.x	20c.1	21c.1	22c.1
--help	O	O	O	O	O
--include-path	O	O	O	O	O
--no-prompt	O	O	O	O	O
--output	O	O	O	O	O
--unsafe-null	O	O	O	O	O
--version	O	O	O	O	O
--no-lineinfo	X	O	O	O	O
--char_map	X	O	O	O	O
--cumulative	X	X	O	O	O
--autocommit	X	X	X	O	O

Embedded SQL-only Syntax

The following is a feature matrix of embedded SQL-only syntax.

Table 4-22 Feature matrix for embedded SQL-only syntax

Feature	2.x	3.x	20c.1	21c.1	22c.1
EXEC SQL AT	O	O	O	O	O
EXEC SQL ATOMIC INSERT	O	O	O	O	O
EXEC SQL AUTOCOMMIT	O	O	O	O	O
EXEC SQL BEGIN DECLARE SECTION	O	O	O	O	O
EXEC SQL COMMIT RELEASE	O	O	O	O	O
EXEC SQL CONNECT	O	O	O	O	O
EXEC SQL CONTEXT ALLOCATE	O	O	O	O	O
EXEC SQL CONTEXT FREE	O	O	O	O	O
EXEC SQL CONTEXT USE	O	O	O	O	O
EXEC SQL DISCONNECT	O	O	O	O	O
EXEC SQL END DECLARE SECTION	O	O	O	O	O
EXEC SQL FOR	O	O	O	O	O
EXEC SQL GET GROUPID	X	O	O	O	O
EXEC SQL INCLUDE	O	O	O	O	O
EXEC SQL INCLUDE SQLCA	O	O	O	O	O
EXEC SQL OPTION	O	O	O	O	O
EXEC SQL ROLLBACK RELEASE	O	O	O	O	O

Feature	2.x	3.x	20c.1	21c.1	22c.1
EXEC SQL WHENEVER	O	O	O	O	O

Host Variable Data Type

The following is a feature matrix for embedded SQL data type which can be used for HOST variables.

Table 4-23 Feature matrix for host variable data type

Feature	2.x	3.x	20c.1	21c.1	22c.1
C native type	O	O	O	O	O
struct, union	O	O	O	O	O
typedef	O	O	O	O	O
VARCHAR	O	O	O	O	O
LONG VARCHAR	O	O	O	O	O
BINARY	O	O	O	O	O
LONG VARBINARY	O	O	O	O	O
BOOLEAN	O	O	O	O	O
NUMBER	O	O	O	O	O
DATE	O	O	O	O	O
TIME	O	O	O	O	O
TIME WITH TIMEZONE	O	O	O	O	O
TIMESTAMP	O	O	O	O	O
TIMESTAMP WITH TIMEZONE	O	O	O	O	O
INTERVAL YEAR	O	O	O	O	O
INTERVAL MONTH	O	O	O	O	O
INTERVAL DAY	O	O	O	O	O
INTERVAL HOUR	O	O	O	O	O
INTERVAL MINUTE	O	O	O	O	O
INTERVAL SECOND	O	O	O	O	O
INTERVAL YEAR TO MONTH	O	O	O	O	O
INTERVAL DAY TO HOUR	O	O	O	O	O
INTERVAL DAY TO MINUTE	O	O	O	O	O
INTERVAL DAY TO SECOND	O	O	O	O	O
INTERVAL HOUR TO MINUTE	O	O	O	O	O
INTERVAL HOUR TO SECOND	O	O	O	O	O
INTERVAL MINUTE TO SECOND	O	O	O	O	O

Dynamic SQL

The following is a feature matrix for dynamic SQL.

Table 4-24 Feature matrix for dynamic SQL

Feature	2.x	3.x	20c.1	21c.1	22c.1
SELECT .. INTO	O	O	O	O	O
EXECUTE IMMEDIATE sql	O	O	O	O	O
PREPARE stmt	O	O	O	O	O
EXECUTE stmt	O	O	O	O	O
DECLARE cursor FOR sql	O	O	O	O	O
DECLARE cursor FOR stmt	O	O	O	O	O
OPEN cursor	O	O	O	O	O
OPEN cursor USING	O	O	O	O	O
FETCH cursor INTO	O	O	O	O	O
CLOSE cursor	O	O	O	O	O
DELETE .. WHERE CURRENT OF cursor	O	O	O	O	O
UPDATE .. WHERE CURRENT OF cursor	O	O	O	O	O

PyDBC

Module

The following is a method feature matrix for pygoldilocks provided by PyDBC.

Table 4-25 Feature matrix for pygoldilock method

Feature	2.x	3.x	20c.1	21c.1	22c.1
connect	X	O	O	O	O
Date	X	O	O	O	O
Time	X	O	O	O	O
Timestamp	X	O	O	O	O
DateFromTicks	X	O	O	O	O
TimeFromTicks	X	O	O	O	O
TimestampFromTicks	X	O	O	O	O
Binary	X	O	O	O	O
STRING	X	O	O	O	O
BINARY	X	O	O	O	O
NUMBER	X	O	O	O	O
DATETIME	X	O	O	O	O
ROWID	X	O	O	O	O
getDecimalSeparator	X	O	O	O	O
setDecimalSeparator	X	O	O	O	O

The following is an attribute feature matrix for pygoldilocks module.

Table 4-26 Feature matrix for pygoldilock attribute

Feature	2.x	3.x	20c.1	21c.1	22c.1
apilevel	X	O	O	O	O
threadsafety	X	O	O	O	O
paramstyle	X	O	O	O	O
version	X	O	O	O	O
lowercase	X	O	O	O	O

Connection

The following is a method feature matrix for connection object.

Table 4-27 Feature matrix for connection method

Feature	2.x	3.x	20c.1	21c.1	22c.1
cursor	X	O	O	O	O
commit	X	O	O	O	O

Feature	2.x	3.x	20c.1	21c.1	22c.1
rollback	X	O	O	O	O
close	X	O	O	O	O
getinfo	X	O	O	O	O
execute	X	O	O	O	O
set_attr	X	O	O	O	O

The following is an attribute feature matrix for connection object.

Table 4-28 Feature matrix for connection attribute

Feature	2.x	3.x	20c.1	21c.1	22c.1
autocommit	X	O	O	O	O
searchescape	X	O	O	O	O
timeout	X	O	O	O	O

Cursor

The following is a method feature matrix for cursor object.

Table 4-29 Feature matrix for cursor method

Feature	2.x	3.x	20c.1	21c.1	22c.1
excute	X	O	O	O	O
executemany	X	O	O	O	O
fetchone	X	O	O	O	O
fetchall	X	O	O	O	O
fetchmany	X	O	O	O	O
commit	X	O	O	O	O
rollback	X	O	O	O	O
skip	X	O	O	O	O
nextset	X	O	O	O	O
close	X	O	O	O	O
setinputsizes	X	O	O	O	O
setoutputsize	X	O	O	O	O
callproc	X	O	O	O	O
callfunc	X	O	O	O	O
tables	X	O	O	O	O
columns	X	O	O	O	O
statistics	X	O	O	O	O
rowIdColumns	X	O	O	O	O
rowVerColumns	X	O	O	O	O
primaryKeys	X	O	O	O	O

Feature	2.x	3.x	20c.1	21c.1	22c.1
foreignKeys	X	O	O	O	O
procedures	X	O	O	O	O
getTypeInfo	X	O	O	O	O

The following is an attribute feature matrix for cursor object.

Table 4-30 Feature matrix for cursor attribute

Feature	2.x	3.x	20c.1	21c.1	22c.1
Description	X	O	O	O	O
rowcount	X	O	O	O	O
arraysize	X	O	O	O	O
connection	X	O	O	O	O
fast_executemany	X	O	O	O	O

Row

The following is an attribute feature matrix for row object.

Table 4-31 Feature matrix for row attribute

Feature	2.x	3.x	20c.1	21c.1	22c.1
cursor_description	X	O	O	O	O

Utility

gcreatedb

Command Usage

The following is a feature matrix for command usage of gcreatedb.

Table 4-32 Feature matrix for command usage of gcreatedb

Feature	2.x	3.x	20c.1	21c.1	22c.1
--character_set	O	O	O	O	O
--char_length_units	O	O	O	O	O
--cluster	X	O	O	O	O
--db_comment	O	O	O	O	O
--help	O	O	O	O	O
--host	X	O	O	O	O
--member	X	O	O	O	O
--port	X	O	O	O	O
--silent	O	O	O	O	O
--timezone	O	O	O	O	O

glsnr

Command Usage

The following is a feature matrix for command usage of glsnr.

Table 4-33 Feature matrix for command usage of glsnr

Feature	2.x	3.x	20c.1	21c.1	22c.1
--help	O	O	O	O	O
--home	X	O	O	O	O
--silent	O	O	O	O	O
--start	O	O	O	O	O
--status	O	O	O	O	O
--stop	O	O	O	O	O

Configuration File

The following is a feature matrix for configuration of glsnr.

Table 4-34 Feature matrix for configuration of glsnr

Feature	2.x	3.x	20c.1	21c.1	22c.1
BACKLOG	O	O	O	O	O
DEFAULT_CS_MODE	O	O	O	O	O
LISTENER_LOG_DIR	X	O	O	O	O
LISTEN_PORT	O	O	O	O	O
TCP_EXCLUDED	O	O	O	O	O
TCP_INVITED	O	O	O	O	O
TCP_HOST	O	O	O	O	O
TCP_VALIDNODE_CHECKING	O	O	O	O	O
TIMEOUT	O	O	O	O	O
USR_DIR	X	O	O	O	O

gsql/ gsqlnet

Command Usage

The following is a feature matrix for command usage of gsql.

Table 4-35 Feature matrix for command usage of gsql

Feature	2.x	3.x	20c.1	21c.1	22c.1
username password	O	O	O	O	O
--as {SYSDBA|ADMIN}	O	O	O	O	O
--conn-string	O	O	O	O	O
--dsn	O	O	O	O	O
--enable-color	O	O	O	O	O
--help	O	O	O	O	O
--import	O	O	O	O	O
--no-prompt	O	O	O	O	O
--prompt	O	O	O	O	O
--silent	O	O	O	O	O
--version	O	O	O	O	O

Interactive gsql Command

The following is a feature matrix for interactive gsql command which is used in gsql prompt state.

Table 4-36 Feature matrix for interactive gsql command

Feature	2.x	3.x	20c.1	21c.1	22c.1
\\	O	O	O	O	O
\connect userid password [as sysdba]	O	O	O	O	O
\cshutdown	X	O	O	O	O
\cstartup	X	O	O	O	O
\ddl_cluster	X	O	O	O	O
\ddl_db	O	O	O	O	O
\ddl_tablespace	O	O	O	O	O
\ddl_profile	O	O	O	O	O
\ddl_audit_policy	X	O	O	O	O
\ddl_auth	O	O	O	O	O
\ddl_schema	O	O	O	O	O
\ddl_public_synonym	O	O	O	O	O
\ddl_table	O	O	O	O	O
\ddl_constraint	O	O	O	O	O
\ddl_index	O	O	O	O	O
\ddl_view	O	O	O	O	O

Feature	2.x	3.x	20c.1	21c.1	22c.1
\ddl_sequence	O	O	O	O	O
\ddl_synonym	O	O	O	O	O
\ddl_procedure	X	O	O	O	O
\ddl_package	X	O	O	O	O
\desc	O	O	O	O	O
\dynamic sql :var	O	O	O	O	O
\exec	O	O	O	O	O
\exec :var := :value	O	O	O	O	O
\exec sql	O	O	O	O	O
\explain plan [on¦only]	O	O	O	O	O
\help	O	O	O	O	O
\history	O	O	O	O	O
\host {os_command}	X	O	O	O	O
\import	O	O	O	O	O
\idesc	O	O	O	O	O
\{n}	O	O	O	O	O
\prepare sql	O	O	O	O	O
\print	O	O	O	O	O
\quit	O	O	O	O	O
\set autocommit	O	O	O	O	O
\set color	O	O	O	O	O
\set colsize	O	O	O	O	O
\set ddlsize	O	O	O	O	O
\set error	O	O	O	O	O
\set history	O	O	O	O	O
\set linesize	O	O	O	O	O
\set numsize	O	O	O	O	O
\set pagesize	O	O	O	O	O
\set sqlprompt	X	X	X	X	O
\set timing	O	O	O	O	O
\set vertical	O	O	O	O	O
\shutdown {abort¦immediate¦transactional¦normal}	O	O	O	O	O
\startup {nomount¦mount¦open}	O	O	O	O	O
\var	O	O	O	O	O

gloader/ gloadernet

Command Usage

The following is a feature matrix for command usage of gloader.

Table 4-37 Feature matrix for command usage of gloader

Feature	2.x	3.x	20c.1	21c.1	22c.1
username password	O	O	O	O	O
--array	O	O	O	O	O
--atomic	O	O	O	O	O
--bad	O	O	O	O	O
--buffered	O	O	O	O	O
--commit	O	O	O	O	O
--control	O	O	O	O	O
--data	O	O	O	O	O
--dsn	O	O	O	O	O
--errors	O	O	O	O	O
--export	O	O	O	O	O
--fieldterm	X	O	O	O	O
--filesize	O	O	O	O	O
--format	O	O	O	O	O
--help	O	O	O	O	O
--import	O	O	O	O	O
--lineterm	X	O	O	O	O
--log	O	O	O	O	O
--no-prompt	O	O	O	O	O
--parallel	O	O	O	O	O
--propagation	O	O	O	O	O
--qualifier	X	O	O	O	O
--silent	O	O	O	O	O
--AsTIMESTAMP	O	O	O	O	O
--where	X	O	O	O	O
--group-id	X	O	O	O	O
--directio-size	X	O	O	O	O

Control File Syntax

The following is a feature matrix for control file syntax of gloader.

Table 4-38 Feature matrix for control file syntax of gloader

Feature	2.x	3.x	20c.1	21c.1	22c.1
CHARACTERSET	O	O	O	O	O
FIELDS TERMINATED BY	O	O	O	O	O
OPTIONALLY ENCLOSED BY	O	O	O	O	O
TABLE table_name	O	O	O	O	O
TABLE schema_name.table_name	O	O	O	O	O
LTRIM	X	O	O	O	O
RTRIM	X	O	O	O	O
LINES TERMINATED BY	X	O	O	O	O
WHERE	X	O	O	O	O

gdump

Command Usage

The following is a feature matrix for command usage of gdump.

Table 4-39 Feature matrix for command usage of gdump

Item	Feature	2.x	3.x	20c.1	21c.1	22c.1
Common arguments	--silent	O	O	O	O	O
File type	BACKUP	O	O	O	O	O
	COMMIT_LOG	X	O	O	O	O
	CONTROL	O	O	O	O	O
	DATA	O	O	O	O	O
	LOG	O	O	O	O	O
	LOG_BUFFER	X	O	O	O	O
	PEND_BUFFER	X	O	O	O	O
	PROPERTY	O	O	O	O	O
BACKUP file arguments	--body	O	O	O	O	O
	--tbs	O	O	O	O	O
	--number	O	O	O	O	O
	--fetch	O	O	O	O	O
CONTROL file arguments	--section	O	O	O	O	O
DATA file arguments	--header	O	O	O	O	O
	--number	O	O	O	O	O
	--fetch	O	O	O	O	O
LOG file arguments	--all	X	O	O	O	O
	--fetch	O	O	O	O	O
	--header	X	O	O	O	O
	--number	O	O	O	O	O
	--offset	O	O	O	O	O

tablediff

Configuration File

The following is a feature matrix for configuration file of tablediff.

Table 4-40 Feature matrix for configuration file of tablediff

Item	Feature	2.x	3.x	20c.1	21c.1	22c.1
Source table	SOURCE_PASSWORD	O	O	O	O	O
	SOURCE_SCHEMA	O	O	O	O	O
	SOURCE_TABLE	O	O	O	O	O
	SOURCE_URL	O	O	O	O	O
	SOURCE_USER	O	O	O	O	O
Target table	TARGET_PASSWORD	O	O	O	O	O
	TARGET_SCHEMA	O	O	O	O	O
	TARGET_TABLE	O	O	O	O	O
	TARGET_URL	O	O	O	O	O
	TARGET_USER	O	O	O	O	O
Sync operation	TARGET_INSERT	O	O	O	O	O
	TARGET_UPDATE	O	O	O	O	O
	TARGET_DELETE	O	O	O	O	O
	SOURCE_INSERT	O	O	O	O	O
Operation options	DIFF_BIN_FILE	O	O	O	O	O
	DIFF_OUT_FILE	O	O	O	O	O
	DISPLAY_CALL_STACK	O	O	O	O	O
	DISPLAY_ROW_UNIT	O	O	O	O	O
	EXCLUDE_COLUMNS	O	O	O	O	O
	LOGGING_ON_DIFF	O	O	O	O	O
	LOGGING_ON_SUCCESS	O	O	O	O	O
	JOB_QUEUE_SIZE	O	O	O	O	O
	JOB_THREAD	O	O	O	O	O
	JOB_UNIT_SIZE	O	O	O	O	O
	PARTITION_RANGE	O	O	O	O	O
	SYNC_OUT_FILE	O	O	O	O	O
	WHERE_CLAUSE	O	O	O	O	O

gsyncher

Command Usage

The following is a feature matrix for command usage of gsyncher.

Table 4-41 Feature matrix for command usage of gsyncher

Feature	2.x	3.x	20c.1	21c.1	22c.1
--log	O	O	O	O	O
--silent	O	O	O	O	O
--home	X	O	O	O	O
--copy-right	O	O	O	O	O
--backup-path	O	O	O	O	O
--help	O	O	O	O	O

gmon

Command Usage

The following is a feature matrix for command usage of gmon.

Table 4-42 Feature matrix for command usage of gmon

Feature	2.x	3.x	20c.1	21c.1	22c.1
--start	X	O	O	O	O
--stop	X	O	O	O	O
--status	X	O	O	O	O
--home	X	O	O	O	O
--uds_dir	X	X	O	O	O
--silent	X	O	O	O	O
--no-copyright	X	O	O	O	O
--help	X	O	O	O	O

gtrclogger

Command Usage

The following is a feature matrix for command usage of gtrclogger.

Table 4-43 Feature matrix for command usage of gtrclogger

Feature	2.x	3.x	20c.1	21c.1	22c.1
--dir	X	O	O	O	O
--help	X	O	O	O	O
--port	X	O	O	O	O
--start	X	O	O	O	O
--stop	X	O	O	O	O

glocator

Command Usage

The following is a feature matrix for command usage of glocator.

Table 4-44 Feature matrix for command usage of glocator

Feature	2.x	3.x	20c.1	21c.1	22c.1
--create	X	O	O	O	O
--start	X	O	O	O	O
--stop	X	O	O	O	O
--conf	X	O	O	O	O
--status	X	O	O	O	O
--sync	X	O	O	O	O
--silent	X	O	O	O	O
--no-copyright	X	O	O	O	O
--help	X	O	O	O	O

Configuration File

The following is a feature matrix for configuration file of glocator.

Table 4-45 Feature matrix for configuration file of glocator

Feature	2.x	3.x	20c.1	21c.1	22c.1
PORT	X	O	O	O	O
WORKER_COUNT	X	O	O	O	O
SESSION_QUEUE_SIZE	X	O	O	O	O
SESSION_ALLOCATOR_SIZE	X	O	O	O	O
PACKET_ALLOCATOR_SIZE	X	O	O	O	O
SYSTEM_LOGGER_DIR	X	O	O	O	O
SYSTEM_UDS_DIR	X	O	O	O	O
LOCATION_FILE_DIR	X	O	O	O	O
LOCATION_FILE_SIZE	X	O	O	O	O
LOCATION_FILE_MAX_SIZE	X	O	O	O	O
SESSION_TIMEOUT	X	O	O	O	O
FAILOVER_TIMEOUT	X	O	O	O	O
ALTERNATE_LOCATORS	X	O	O	O	O
SYNC_RETRY_COUNT	X	O	O	O	O
SYNC_RESPONSE_TIMEOUT	X	O	O	O	O

gagent

Command Usage

The following is a feature matrix for command usage of gagent.

Table 4-46 Feature matrix for command usage of gagent

Feature	2.x	3.x	20c.1	21c.1	22c.1
--start	X	O	O	O	O
--stop	X	O	O	O	O
--conf	X	O	O	O	O
--status	X	O	O	O	O
--home	X	O	O	O	O
--silent	X	O	O	O	O
--no-copyright	X	O	O	O	O
--help	X	O	O	O	O

Configuration File

The following is a feature matrix for configuration file of gagent.

Table 4-47 Feature matrix for configuration file of gagent

Feature	2.x	3.x	20c.1	21c.1	22c.1
PORT	X	O	O	O	O
LOCATOR_HOST	X	O	O	O	O
LOCATOR_PORT	X	O	O	O	O
COMMAND_QUEUE_SIZE	X	O	O	O	O
COMMAND_ALLOCATOR_SIZE	X	O	O	O	O
PACKET_ALLOCATOR_SIZE	X	O	O	O	O
SYSTEM_LOGGER_DIR	X	O	O	O	O
SESSION_TIMEOUT	X	O	O	O	O
UPDATE_LOCATION_TIME	X	O	O	O	O
ALTERNATE_LOCATORS	X	O	O	O	O

gloctl

Command Usage

The following is a feature matrix for command usage of gloctl.

Table 4-48 Feature matrix for command usage of gloctl

Feature	2.x	3.x	20c.1	21c.1	22c.1
--dsn	X	X	X	X	X
--conf	X	O	O	O	O
--ip	X	O	O	O	O
--port	X	O	O	O	O
--import	X	O	O	O	O
--silent	X	O	O	O	O
--no-copyright	X	O	O	O	O
--help	X	O	O	O	O

Configuration File

The following is a feature matrix for configuration file of gloctl.

Table 4-49 Feature matrix for configuration file of gloctl

Feature	2.x	3.x	20c.1	21c.1	22c.1
PORT	X	O	O	O	O
LOCATOR_HOST	X	O	O	O	O
LOCATOR_PORT	X	O	O	O	O

Replication

cyclone

Command Usage

The following is a feature matrix for command usage of cyclone.

Table 4-50 Feature matrix for command usage of cyclone

Feature	2.x	3.x	20c.1	21c.1	22c.1
--conf	O	O	O	O	O
--encrypt	X	O	O	O	O
--group	O	O	O	O	O
--help	O	O	O	O	O
--key	X	O	O	O	O
--master	O	O	O	O	O
--reset	O	O	O	O	O
--silent	O	O	O	O	O
--slave	O	O	O	O	O
--start	O	O	O	O	O
--status	O	O	O	O	O
--stop	O	O	O	O	O
--sync	O	O	O	O	O
--stand-alone	X	O	O	O	O
--recovery	X	O	O	O	O
--local	X	O	O	O	O

Configuration File

The following is a feature matrix for configuration file of cyclone.

Table 4-51 Feature matrix for configuration file of cyclone

Configuration	Feature	2.x	3.x	20c.1	21c.1	22c.1
Common configuration	COMM_CHUNK_COUNT	O	O	O	O	O
	DSN	O	O	O	O	O
	USER_ENCRYPT_PW	X	O	O	O	O
	GROUP_NAME	O	O	O	O	O
	HOST_IP	O	O	O	O	O
	HOST_EXTERNAL_IP	X	O	O	O	O
	HOST_PORT	O	O	O	O	O
	PORT	O	O	O	O	O

Configuration	Feature	2.x	3.x	20c.1	21c.1	22c.1
	PROTOCOL	X	O	O	O	O
	USER_ID	O	O	O	O	O
	USER_PW	O	O	O	O	O
MASTER configuration	CAPTURE_TABLE	O	O	O	O	O
	LOG_PATH	O	O	O	O	O
	READ_LOG_BLOCK_COUNT	O	O	O	O	O
	TRANS_SORT_AREA_SIZE	O	O	O	O	O
	TRANS_FILE_PATH	O	O	O	O	O
	SYNCHER_COUNT	O	O	O	O	O
	SYNC_ARRAY_SIZE	O	O	O	O	O
	GIVEUP_INTERVAL	O	O	O	O	O
	LOG_CAPTURE_INTERVAL_1	X	O	O	O	O
	LOG_CAPTURE_INTERVAL_2	X	O	O	O	O
SLAVE configuration	APPLIER_COUNT	O	O	O	O	O
	APPLY_ARRAY_SIZE	O	X	X	X	X
	APPLY_COMMIT_SIZE	O	O	O	O	O
	APPLY_TABLE	O	O	O	O	O
	MASTER_IP	O	O	O	O	O
	PROPAGATE_MODE	O	O	O	O	O
	CLUSTER	X	O	O	O	O
	ORACLE_DRIVER	X	O	O	O	O

clustone

Command Usage

The following is a feature matrix for command usage of clustone.

Table 4-52 Feature matrix for command usage of clustone

Feature	2.x	3.x	20c.1	21c.1	22c.1
--conf	X	X	X	X	O
--encrypt	X	X	X	X	O
--group	X	X	X	X	O
--help	X	X	X	X	O
--key	X	X	X	X	O
--master	X	X	X	X	O
--reset	X	X	X	X	O
--silent	X	X	X	X	O
--slave	X	X	X	X	O
--start	X	X	X	X	O
--status	X	X	X	X	O
--stop	X	X	X	X	O
--stand-alone	X	X	X	X	O
--dump	X	X	X	X	O
--info	X	X	X	X	O

Configuration File

The following is a feature matrix for configuration file of clustone.

Table 4-53 Feature matrix for configuration file of clustone

Configuration	Feature	2.x	3.x	20c.1	21c.1	22c.1
Common configuration	COMM_CHUNK_COUNT	X	X	X	X	O
	DSN	X	X	X	X	O
	USER_ENCRYPT_PW	X	X	X	X	O
	GROUP_NAME	X	X	X	X	O
	HOST_IP	X	X	X	X	O
	HOST_EXTERNAL_IP	X	X	X	X	O
	HOST_PORT	X	X	X	X	O
	PORT	X	X	X	X	O
	PROTOCOL	X	X	X	X	O
	USER_ID	X	X	X	X	O
	USER_PW	X	X	X	X	O
	HEARTBEAT_TIMEOUT	X	X	X	X	O

Configuration	Feature	2.x	3.x	20c.1	21c.1	22c.1
MASTER configuration	CAPTURE_TABLE	X	X	X	X	O
	TXDATA_FILE_SIZE	X	X	X	X	O
	TXDATA_FILE_PATH	X	X	X	X	O
	LOG_PATH	X	X	X	X	O
	READ_LOG_BLOCK_COUNT	X	X	X	X	O
	TRANS_SORT_AREA_SIZE	X	X	X	X	O
	TRANS_FILE_PATH	X	X	X	X	O
	GIVEUP_INTERVAL	X	X	X	X	O
	LOG_CAPTURE_INTERVAL_1	X	X	X	X	O
	LOG_CAPTURE_INTERVAL_2	X	X	X	X	O
	PACKET_COMPRESSION_MODE	X	X	X	X	O
SLAVE configuration	APPLIER_COUNT	X	X	X	X	O
	APPLY_COMMIT_SIZE	X	X	X	X	O
	APPLY_TABLE	X	X	X	X	O
	MASTER_IP	X	X	X	X	O
	PROPAGATE_MODE	X	X	X	X	O
	CLUSTER	X	X	X	X	O

logmirror

Command Usage

The following is a feature matrix for command usage of logmirror.

Table 4-54 Feature matrix for command usage of logmirror

Feature	2.x	3.x	20c.1	21c.1	22c.1
--conf	O	O	O	O	O
--help	O	O	O	O	O
--infiniband	O	O	O	O	O
--master	O	O	O	O	O
--silent	O	O	O	O	O
--slave	O	O	O	O	O
--start	O	O	O	O	O
--stop	O	O	O	O	O

Configuration File

The following is a feature matrix for configuration file of logmirror.

Table 4-55 Feature matrix for configuration file of logmirror

Configuration	Feature	2.x	3.x	20c.1	21c.1	22c.1
Common configuration	PORT	O	O	O	O	O
MASTER configuration	DSN	O	O	O	O	O
	HOST_IP	O	O	O	O	O
	HOST_PORT	O	O	O	O	O
	PROTOCOL	X	O	O	O	O
	USER_ID	O	O	O	O	O
	USER_PW	O	O	O	O	O
SLAVE configuration	LOG_PATH	O	O	O	O	O
	MASTER_IP	O	O	O	O	O

cymon

Command Usage

The following is a feature matrix for command usage of cymon.

Table 4-56 Feature matrix for command usage of cymon

Feature	2.x	3.x	20c.1	21c.1	22c.1
--conf	O	O	O	O	O
--help	O	O	O	O	O
--cycle	O	O	O	O	O
--key	X	O	O	O	O
--start	O	O	O	O	O
--stop	O	O	O	O	O
--status	O	O	O	O	O

cyfile

Command Usage

The following is a feature matrix for command usage of cyfile.

Table 4-57 Feature matrix for command usage of cyfile

Feature	2.x	3.x	20c.1	21c.1	22c.1
--conf	X	X	O	O	O
--help	X	X	O	O	O
--reset	X	X	O	O	O
--key	X	X	O	O	O
--silent	X	X	O	O	O
--info	X	X	O	O	O
--start	X	X	O	O	O
--stop	X	X	O	O	O
--group	X	X	O	O	O
--encrypt	X	X	O	O	O
--status	X	X	O	O	O

Configuration File

The following is a feature matrix for configuration file of cyfile.

Table 4-58 Feature matrix for configuration file of cyfile

Feature	2.x	3.x	20c.1	21c.1	22c.1
DSN	X	X	O	O	O
HOST_IP	X	X	O	O	O
HOST_PORT	X	X	O	O	O
PROTOCOL	X	X	O	O	O
USER_ID	X	X	O	O	O
USER_PW	X	X	O	O	O
GROUP_NAME	X	X	O	O	O
USER_ENCRYPT_PW	X	X	O	O	O
CAPTURE_TABLE	X	X	O	O	O
READ_LOG_BLOCK_COUNT	X	X	O	O	O
TRANS_SORT_AREA_SIZE	X	X	O	O	O
TRANS_FILE_PATH	X	X	O	O	O
LOG_CAPTURE_INTERVAL_1	X	X	O	O	O
LOG_CAPTURE_INTERVAL_2	X	X	O	O	O
DATA_FILE_PATH	X	X	O	O	O
DATA_FILE_PREFIX	X	X	O	O	O

Feature	2.x	3.x	20c.1	21c.1	22c.1
DATA_FILE_SIZE	X	X	O	O	O
UPDATE_BEFORE_VALUE	X	X	O	O	O

4.2 What's New in GOLDILOCKS 22c.1

This chapter briefly describes the features added to GOLDILOCKS 22c.1.

Architecture

System Architecture

It has not been changed.

Storage Internal

It has not been changed.

Transaction Control

It has not been changed.

Backup & Recovery

It has not been changed.

Database Information

DICTIONARY_SCHEMA

It has not been changed.

INFORMATION_SCHEMA

It has not been changed.

PERFORMANCE_VIEW_SCHEMA

V$OPEN_CURSOR has been added.

V$PROPERTY_ALIAS has been added.

V$DB_PROPERTY has been added.

V$LICENSE has been added.

Server Property

REBALANCE_SHARD_DIVISOR has been added.

SESSION_MEMORY_INIT_SIZE has been added.

SESSION_POOL_NEXT_SIZE has been added.

ADMIN_SESSION_POOL_INIT_SIZE has been added.

ADMIN_SESSION_POOL_NEXT_SIZE has been added.

The default value of **DEFAULT_INDEX_PCTFREE** has been changed to 10.

The default value of **DEFAULT_MAXTRANS** has been changed to 32.

The name of the property INCREMENTAL_CHECKPOINT_CRITERIA which sets the criteria to perform the i ncremental checkpoint of system, has been changed to **BUFFER_DIRTY_PAGE_LIMIT**. Its default value als o has been changed to 0.

BUFFER_LRU_SCAN_PERCENT property has been added to improve the buffer management algorithm fo r the disk tablespace.

CLUSTER_CM_BUFFER_COUNT property, which specifies the number of communication buffers in a clust er environment, has been deprecated. Instead, **LOCKABLE_DISPATCHER_CM_BUFFER_COUNT**, **LOCKLES S_DISPATCHER_CM_BUFFER_COUNT**, and **SYNC_DISPATCHER_CM_BUFFER_COUNT** properties that spe cify the number of communication buffers for lockable, lockless, and synchronization dispatchers have be en added.

The threshold of the table size determines whether to cache the tables created in the disk tablespace to b uffer cache when performing a full scan. **FULL_TABLE_SCAN_CACHING_THRESHOLD** property, which set s this threshold value, has been added.

INST_HASH_TABLE_BUCKET_MAX_COUNT property has been added to set the maximum expected buck et counts of the hash instant table.

The maximum number of SQL to be cached in the plan cache used to be controlled by two properties. M AXIMUM_FLANGE_COUNT property was deprecated, so only **PLAN_CACHE_SIZE** controls the maximum number of SQL.

SQL

SQL Element

Data Type

It has not been changed.

Function

DISTINCT Condition has been added.

SESSIONTIMEZONE has been added.

Window Function has been added.

Object

SQL Object

It has not been changed.

Cluster Object

ALTER DATABASE DROP OFFLINE SEGMENTS has been added.

ALTER DATABASE SYNCHRONIZE has been added.

ALTER TABLE name DROP OFFLINE SEGMENTS has been added.

ALTER TABLE name SYNCHRONIZE has been added.

SHARD DIVISOR and PARALLEL options have been added to **ALTER DATABASE MOVE SHARD**, **ALTER DATABASE REBALANCE**, **ALTER DATABASE REBALANCE EXCLUDE CLUSTER GROUP**, **ALTER TABLE name MOVE SHARD**, **ALTER TABLE name REBALANCE**, **ALTER TABLE name REBALANCE EXCLUDE CLUSTER GROUP cluster_group_list**.

ALTER TABLE REBUILD GLOBAL SECONDARY INDEX statement has been changed to **ALTER TABLE name ALTER GLOBAL SECONDARY INDEX REBUILD**.

SQL Language

DML

It has not been changed.

Query

Lateral Inline View

Lateral inline view has been added to **from clause**.

Table Function Derived Table

Table function derived table has been added to **from clause**.

WINDOW Clause

WINDOW clause has been added, which defines the execution range of the window function.
For more information, refer to **window clause**.

Control Language

It has not been changed.

PSM Language

Table Function

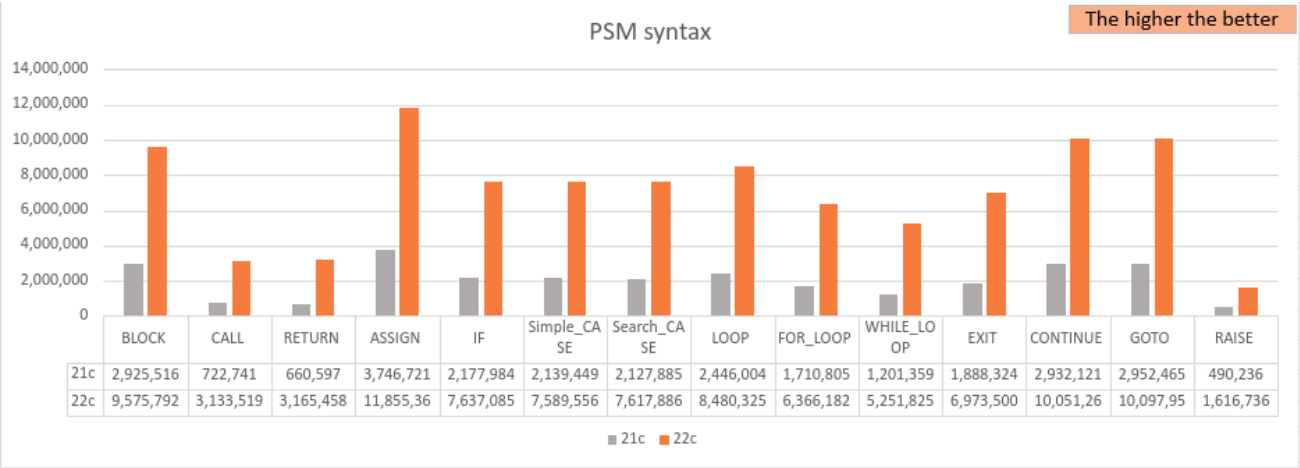
TABLE (table function column list) statement has been added to **return clause**.
The table function can be created by defining the table type when performing function DDL.
For more information, refer to **CREATE FUNCTION**.

RETURN TABLE Statement

RETURN TABLE Statement has been added to the PSM statement.

Performance Improvement of PSM statement

The performance of PSM syntax has been improved as follows.



	BLOCK	CALL	RETURN	ASSIGN	IF	Simple_CASE	Search_CASE	LOOP	FOR_LOOP	WHILE_LOOP	EXIT	CONTINUE	GOTO	RAISE
21c	2,925,516	722,741	660,597	3,746,721	2,177,984	2,139,449	2,127,885	2,446,004	1,710,805	1,201,359	1,888,324	2,932,121	2,952,465	490,236
22c	9,575,792	3,133,519	3,165,458	11,855,36	7,637,085	7,589,556	7,617,886	8,480,325	6,366,182	5,251,825	6,973,500	10,051,26	10,097,95	1,616,736

API

ODBC

TRACE_POLICY has been added to **Data Source Configuration**.

JDBC

It has been modified to prevent missing microsecond when outputting time type and timestamp type in s tring.

It supports the auto-generated key.

GoldilocksTypes.REF_CURSOR has been added.

GoldilocksPreparedStatement.setFixedCHAR(), a JDBC non standard function, has been added.

Embedded SQL

gpec supports **Declaring Function Argument**.

PDO

It has not been changed.

PyDBC

It has not been changed.

Ruby

It has not been changed.

Hibernate

It has not been changed.

Utility

gcreatedb

It has not been changed.

glsnr

It has not been changed.

gsql/gsqlnet

\set sqlprompt command has been added.

gloader/gloadernet

It has not been changed.

gdump

It has not been changed.

tablediff

It has not been changed.

gsyncher

It has not been changed.

gmon

It has not been changed.

gtrclogger

It has not been changed.

glocator

It has not been changed.

gagent

It has not been changed.

gloctl

It has not been changed.

Replication

cyclone

It has not been changed.

clustone

It has not been changed.

logmirror

It has not been changed.

cymon

It has not been changed.

cyfile

It has not been changed.

4.3 Patch Notes

22c.1.4 Patch Notes

ISSUE-6381 The conversion rule from the numeric type to the character type is different from other DBMS.

Description

1. If converting the numeric type to the varchar type, it is converted to an exponential type or to a real number type according to the varchar precision. In this case, it is converted by rounding off so that it can be expressed appropriately for the space used as much as possible.
   Therefore, the precision of the valid number representation for the original number decreases.

2. The result of the numeric type → CHAR type conversion and the numeric type → VARCHAR type con version are different.
   I. NUMBER / NUMERIC
      i. Conversion to CHAR: It is converted only to a real number type. (If it can not be expressed within the precision, then an error occurs.)
      ii. Conversion to VARCHAR: It is converted to an exponential type or to a real number type ac cording to precision by rounding off.
   II. NATIVE_REAL / NATIVE_DOUBLE
      i. CHAR: It is converted to an exponential type so that all of the valid numbers are expressed. (If it is truncated, then an error occurs.)
      ii. VARCHAR: It is converted to an exponential type according to precision by rounding off.

Troubleshooting

- If all valid numbers of the numeric type can be expressed, then modify it as follows.
  - NUMBER/ NUMERIC: Convert it to an exponential type or to a real number type according to the precision of the character type.
  - NATIVE_REAL/ NATIVE_DOUBLE: Convert it to an exponential type.
- The result of the numeric type → CHAR type conversion and the numeric type → VARCHAR type conv ersion are same.

Symptom

It is converted by rounding off so that it can be expressed appropriately for the space used as much as po ssible, so the precision of the valid number representation for the original number decreases.

```
gSQL> create table t1 ( c1 varchar(2) );
Table created.
gSQL> insert into t1 values ( 2.4 );
1 row created.
gSQL> insert into t1 values ( 2.5 ) ;
1 row created.
gSQL> commit;
Commit complete.
gSQL> select * from t1;
C1
--
2
3
2 rows selected.
```

Workaround

Specify the appropriate character type precision to express all valid numbers for the original number.

```
gSQL> create table t1 ( c1 varchar(10) );
Table created.
gSQL> insert into t1 values ( 2.4 );
1 row created.
gSQL> insert into t1 values ( 2.5 ) ;
1 row created.
gSQL> commit;
Commit complete.
gSQL> select * from t1;
C1
---
2.4
2.5
2 rows selected.
```

ISSUE-6255 [CDC] PWD was exposed in the connection string recorded in the trace log, so it is replaced with '*' when it is recorded.

Description

PWD was exposed in the connection string recorded in the trace log of cyclone, clustone, cymon and cyfile, and this error has been fixed by replacing it with '*' when it is recorded.

Symptom

PWD was exposed in the connection string of cyclone, clustone, cymon and cyfile when it was recorded i n the trace log.

```
connection string [PROTOCOL=DA;DSN=goldilocks_jinsil;PORT=11100;UID=test;PWD=test]
```

Workaround

The patch is required.

ISSUE-6030 An error occurred in cyclone when rebalancing members i n the Cluster environments, and this error has been fixed.

Description

When rebalancing cluster members while cyclone was in operation in a cluster environment, cyclone coul d not process it, and this error has been fixed.

Symptom

The followings were recorded in cyclone operated in the cluster member where rebalancing was execute d, and no further operation was executed.

```
[2023-12-08 16:33:58.782398 THREAD(3292,140620625983232)]
Ready to Rebalance-Tx commit. (Waiting for slave response)
```

Workaround

The patch is required.

ISSUE-6116 The performance of long procedure's direct execution has been improved.

Description

The direct execution performance of the procedure which consists of massive PL stmt and expression has been improved.

Symptom

proc1 is the procedure which consists of 2000 BEGIN .. END blocks, 12000 PL stmt, and 230,000 expressi ons in the following example.

If proc1() was called as follows, it used to take 150 ms or more.

```
gSQL> call proc1(200,439);
Procedure Call complete.
Elapsed time: 157.09300 ms
```

The performance has been improved as follows by widely improving the plan size of procedure execution and the optimization process.

```
gSQL> call proc1(200,439);
Procedure Call complete.
Elapsed time: 12.35000 ms
```

Workaround

Call the procedure with a prepare-execution.

```
--# prepare
gSQL> \prepare sql call proc1(200,439);
SQL prepared.
--# 1st execution
--# data optimize - execute
gSQL> \exec
Procedure Call complete.
Elapsed time: 158.82600 ms
--# 2nd execution
--# execute
gSQL> \exec
Procedure Call complete.
Elapsed time: 2.79600 ms
```

ISSUE-6231 An error occurs when entering GLOBAL OPEN with an invalid IP.

Description

It failed when trying to go up to GLOBAL OPEN with an invalid remote IP, and this error has been fixed.

Symptom

When trying to go up to GLOBAL OPEN with an invalid remote IP, it should have gone up to GLOBAL OPEN excluding the failed node, but it fails as follows.

```
gSQL> alter system open global database;
ERR-HY000(11047): MEMBER(G1N2): invalid network address : invalid address()
```

Workaround

Alter the IP of the failed node to the valid IP by using ALTER CLUSTER LOCATION statement.

```
gSQL> alter cluster location g1n2 host '127.0.0.1' port 12150;
altered.
```

ISSUE-6358 Cache coherency error in weak memory ordering device

Description

The server was abnormally terminated due to a discrepancy between the sequence of memory access and the program order, and this error has been fixed.

Symptom

The server may experience abnormal behavior or termination because outdated data in the CPU cache is used instead of the most recent data.

Workaround

The patch is required.

22c.1.3 Patch Notes

ISSUE-5799 An error did not occur even though the default expression was not valid when performing CREATE TABLE/ ALTER TABLE.

Description

If defining the default clause when performing CREATE TABLE/ ALTER TABLE, it checks whether the default expression is valid.

Symptom

An error did not occur even though the default expression was not valid.

```
CREATE TABLE t1 ( c1 INTEGER DEFAULT 1 / 0 );
Table created.
```

It has been fixed now, so the error occurs as follows.

```
CREATE TABLE t1 ( c1 INTEGER DEFAULT 1 / 0 );
ERR-22012(12122): divisor is equal to zero
```

Workaround

The patch is required.

ISSUE-5828 The join query including ROWNUM should not be sent to t he remote node, but sometimes it is sent.

Description

The join query including ROWNUM should not be sent to the remote node. If each node stores data in a different order then the result may be wrong even though it is a clone table.

Symptom

```
\explain plan
SELECT COUNT(*)
  FROM ( SELECT c1
           FROM t_clone
          WHERE ROWNUM <= 1000
       ) X
     , t_shard Y
 WHERE X.c1 = Y.c1
;
COUNT(*)
--------
    1005
1 row selected.
>>>  start print plan
< Execution Plan >
================================================================================
¦IDX¦ NODE DESCRIPTION                                                         ¦
--------------------------------------------------------------------------------
¦  0¦  SELECT STATEMENT                                                        ¦
```

```
┆  1┆    QUERY BLOCK ("$QB_IDX_2")                                          ┆
┆  2┆      SINGLE CLUSTER                                                   ┆
┆  3┆        CLUSTER PUSHER ("_$NI_6")                                      ┆
┆  4┆          INLINE_VIEW ("X")                                            ┆
┆  5┆            QUERY BLOCK ("$QB_IDX_6")                                  ┆
┆  6┆              COUNT                                                    ┆
┆  7┆                TABLE ACCESS ("T_CLONE")                               ┆
┆  8┆        AGGREGATION BY HASH                                            ┆
┆  9┆          NESTED JOIN (INNER JOIN)                                     ┆
┆ 10┆            PUSHER TABLE ACCESS ("_$NI_6")                             ┆
┆ 11┆            INDEX ACCESS ("T_SHARD" AS Y, "T_SHARD_PRIMARY_KEY_INDEX")┆
================================================================================
     1  -  TARGET : COUNT(*)
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE USE_HASH_IN( _A1, 7801 ) NO_MERGE( _A2 ) INDEX(
_A1, "PUBLIC"."T_SHARD_PRIMARY_KEY_INDEX" ) */ COUNT(*) FROM ( ( SELECT /*+ FULL( _A3 ) */
"_A3"."C1" FROM "PUBLIC"."T_CLONE"@LOCAL AS "_A3" WHERE ROWNUM <= :_V0 ) AS "_A2"("C1") INNER
JOIN "PUBLIC"."T_SHARD"@LOCAL AS "_A1" ON "_A1"."C1" = "_A2"."C1") ALIAS "_A4"
           TARGET DOMAIN : G1(G1N1) 1 rows, G2(G2N1) 1 rows, G3(G3N1) 1 rows
           RE-AGGREGATION
             AGGREGATION : SUM( COUNT(*) )
     4  -  TARGET : COUNT(*)
     5  -  AGGREGATION : COUNT(*)
     6  -  JOINED COLUMN : NOTHING
     7  -  COLUMN : _A3.C1 AS C1
     8  -  TARGET : _A3.C1
     9  -  STOP KEY FILTER : ROWNUM <= :_V0
    10  -  CLONED
           READ COLUMN : _A3.C1
    11  -  HASH KEY : _A1.C1
           READ KEY COLUMN : _A1.C1
             HASH FILTER : _A1.C1 = _A2.C1
           FETCH ONE ROW
    12  -  HASH SHARD ( # 3 )
           READ INDEX COLUMN : _A1.C1
<<<  end print plan
```

The join query including ROWNUM should not be sent to the remote node, but ROWNUM filter is sent to the remote query node in the query above.

Workaround

Use */*+ LOCAL_JOIN(Y) */* hint.

ISSUE-5810 [CYCLONE] If a record does not exists in the table containing a long varchar column, then it fails to SYNC.

Description

The error occurs because the null check for a long varchar column is incorrectly performed while checking the existence of a record in the table of the master during SYNC. It determines that there is a record even though a record does not exist. Then, it tries to INSERT the null data to the slave, and it outputs the error message "cannot insert NULL into" and it fails to SYNC.

Symptom

If a record does not exist in the table containing a long varchar column while transferring the data of the master to the slave by using SYNC feature during the replication using CYCLONE, then "cannot insert NULL into" error occurs, and it fails to SYNC.

Workaround

The patch is required.

ISSUE-5665 If the join including three or more tables is performed by using the instant nested loop join method, then the result may be wrong.

Description

If the join including three or more tables is performed by using the instant nested loop join method, then the result may be wrong.

Symptom

```
--# result : 27
\EXPLAIN PLAN
SELECT COUNT(t2.col1)
  FROM t2, t3, t4, t1
 WHERE t2.col1 + t1.col1 = t4.col1
   AND t3.col1 + t1.col1 = t4.col1;
```

```
COUNT(T2.COL1)
--------------
             0
1 row selected.
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                                                     |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                     |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                          |
|    2  |      AGGREGATION BY HASH                                              |
|    3  |        NESTED JOIN (INNER JOIN)                                       |
|    4  |          TABLE ACCESS ("T1")                                          |
|    5  |          SORT JOIN INSTANT                                            |
|    6  |            NESTED JOIN (INNER JOIN)                                   |
|    7  |              NESTED JOIN (INNER JOIN)                                 |
|    8  |                INDEX ACCESS ("T2", "T2_COL1")                         |
|    9  |                INDEX ACCESS ("T3", "T3_COL1")                         |
|   10  |              INDEX ACCESS ("T4", "T4_COL1")                           |
================================================================================
     1  -  TARGET : COUNT( T2.COL1 )
     2  -  AGGREGATION : COUNT( T2.COL1 )
     3  -  JOINED COLUMN : T2.COL1
     4  -  READ COLUMN : T1.COL1
     5  -  SORT KEY : "T4.COL1 ASC NULLS LAST"
           RECORD COLUMN : T2.COL1
           READ KEY COLUMN : T4.COL1
           READ RECORD COLUMN : T2.COL1
             MIN RANGE : T4.COL1 = T2.COL1 + {T1.COL1} AND T4.COL1 = T3.COL1 + {T1.COL1}
             MAX RANGE : T4.COL1 = T2.COL1 + {T1.COL1} AND T4.COL1 = T3.COL1 + {T1.COL1}
     6  -  JOINED COLUMN : T4.COL1, T2.COL1, T3.COL1
     7  -  JOINED COLUMN : T2.COL1, T3.COL1
     8  -  READ INDEX COLUMN : T2.COL1
     9  -  READ INDEX COLUMN : T3.COL1
    10  -  READ INDEX COLUMN : T4.COL1
<<<  end print plan
```

The result from the query above should be '27', but actually '0' is output.

The WHERE clause condition 'T4.COL1 = T2.COL1 + T1.COL1 AND T4.COL1 = T3.COL1 + T1.COL1' can n ot be used as an index range condition, but actually it is used.

However, if the query is modified, then the correct result is output as follows.

```
\EXPLAIN PLAN
SELECT COUNT(t2.col1)
  FROM t2, t3, t4, t1
 WHERE t2.col1 + t1.col1 = t4.col1
   AND t3.col1 + t1.col1 = t4.col1;
COUNT(T2.COL1)
--------------
            27
1 row selected.
>>>  start print plan
< Execution Plan >
============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                 ¦
----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                    ¦
¦    2  ¦      AGGREGATION BY HASH                                        ¦
¦    3  ¦        NESTED JOIN (INNER JOIN)                                 ¦
¦    4  ¦          NESTED JOIN (INNER JOIN)                               ¦
¦    5  ¦            NESTED JOIN (INNER JOIN)                             ¦
¦    6  ¦              INDEX ACCESS ("T2", "T2_COL1")                     ¦
¦    7  ¦              INDEX ACCESS ("T3", "T3_COL1")                     ¦
¦    8  ¦            INDEX ACCESS ("T4", "T4_COL1")                       ¦
¦    9  ¦          FLAT JOIN INSTANT                                      ¦
¦   10  ¦            TABLE ACCESS ("T1")                                  ¦
============================================================================
     1  -  TARGET : COUNT( T2.COL1 )
     2  -  AGGREGATION : COUNT( T2.COL1 )
     3  -  JOINED COLUMN : T3.COL1, T1.COL1, T4.COL1, T2.COL1
             ON FILTER : ( T3.COL1 + T1.COL1 ) = T4.COL1 AND ( T2.COL1 + T1.COL1 ) = T4.COL1
     4  -  JOINED COLUMN : T3.COL1, T4.COL1, T2.COL1
     5  -  JOINED COLUMN : T3.COL1, T2.COL1
     6  -  READ INDEX COLUMN : T2.COL1
     7  -  READ INDEX COLUMN : T3.COL1
     8  -  READ INDEX COLUMN : T4.COL1
     9  -  RECORD COLUMN : T1.COL1
           READ COLUMN : T1.COL1
    10  -  READ COLUMN : T1.COL1
<<<  end print plan
```

Workaround

Use a hint other than USE_INL(t1). For example, use USE_NL(t1), USE_HASH(t1) or USE_MERGE(t1).

ISSUE-5710 If a view exists inside a view and group by exists in the innermost view, then the complex view merging occurs in succession, causing an error.

Description

If a view exists inside a view and group by exists in the innermost view, then the complex view merging occurs in succession, causing an error. In this case, the error occurs in an aggregation expression of SELECT list.

Even though the conditions above are satisfied, if an aggregation does not exist in SELECT list, then an error does not occur.

Symptom

```
\EXPLAIN PLAN
SELECT sum(case_sum )
  FROM (
         SELECT
                v1.col1, v1.col2, ( case when v1.sum > 0 then 2
                                    else 1
                                    end
                                  )as case_sum
          FROM ( SELECT col1, col2, sum(col3) as sum
                   FROM t1
                  WHERE col3 > 0
                  GROUP BY col1, col2
                ) v1
       ) v2
      , t2
 WHERE v2.col1 = t2.col1;
ERR-42000(12119): comparison is not applicable: ROWID and NUMBER :
```

If v1 is merged first, and then v2 is merged, then it can not find v1.sum in 'case when then' statement.

Workaround

If either one of two views fails to merge, then an error does not occur. Therefore, use /*+ NO_MERGE(v 1) */ hint or /*+ NO_MERGE(v2)*/ hint.

ISSUE-5667 When a subquery refers to the outer query, if the subquery refers to both a materialized view and an ordinary table, then the serv er is abnormally terminated.

Description

If all of the following conditions are satisfied, then the server is abnormally terminated.

1. When tables are listed in <from clause>, the materialized view specified by <with clause> is listed ahe ad, followed by the ordinary table.
2. A subquery exists and the subquery refers to both a materialized view and an ordinary table.

Symptom

```
DROP TABLE IF EXISTS r;
DROP TABLE IF EXISTS s;
DROP TABLE IF EXISTS t;
CREATE TABLE r ( c1 INTEGER, c2 INTEGER );
INSERT INTO r VALUES(1,1);
CREATE TABLE s ( c1 INTEGER, c2 INTEGER, c3 INTEGER );
INSERT INTO s VALUES(1,1,1);
CREATE TABLE t ( c1 INTEGER, c2 INTEGER );
INSERT INTO t VALUES(1,1);
COMMIT;
--# result : 1 row
\EXPLAIN PLAN
WITH w AS ( SELECT /*+ MATERIALIZE */
                   c1
              FROM r
             WHERE c1 > 0 )
SELECT *
  FROM w, s
 WHERE s.c1 IN ( SELECT t.c1
                   FROM t
                  WHERE s.c1 = t.c1
                    AND w.c1 = t.c2
```

```
                    ) ;
```

In the query above, w and s are listed in an order of w,s in FROM clause, and the subquery refers to both s.c1 and w.c1. In this case, the server is abnormally terminated.

Workaround

List the materialized view last in FROM clause.

ISSUE-5634 V$LICENSE has been added.

Description

The view has been added to view the license information of the currently running server.

Symptom

N/A

Workaround

The patch is required.

ISSUE-5442 [CYCLONE] If replicating the column with a unique attribute, then the transaction may fail.

Description

If a column has a unique attribute in the table to replicate, then it is normally processed in an source database, but unique violation or NO_ROWS error may occur in a remote database. Therefore, to solve this problem, the feature to control concurrency of the column has been added when replicating the column with the unique attribute.

Symptom

```
[APPLIER #1(SID:100)-INSERT] ERR-23000(16057) : unique constraint (PUBLIC.TEST1) violated
[APPLIER #2(SID:101)-UPDATE] Conflict.
[APPLIER #3(SID:102)-DELETE] Conflict.
```

If a column has a unique attribute in the replicating table, then logs are frequently recorded on the trace log of the slave as given above.

Workaround

The patch is required.

ISSUE-5568 [CYCLONE] If replicating recovery spans two redo log files, then the starting point of the recovery is incorrectly set.

Description

When restarting after terminating the replication, cyclone finds the starting point of the recovery by using the information applied to the existing applier, then restarts the replication.

When recovering, it finds the optimal starting point by comparing the information between multiple appli ers. If the redo log file information between appliers is different, then it discards the information of the ol d redo log file, and determines the starting point of the recovery by using only the new redo log file infor mation, and this error has been fixed.

Symptom

When restarting after terminating the replication, a transaction may not be replicated.

Workaround

The patch is required.

ISSUE-5511 [CYCLONE] The internal transaction ID is set incorrectly in t he distributor.

Description

It is guaranteed that the transaction IDs are not duplicated among simultaneously performed transactions in an original database. However, the completed transaction ID can be reused later.

When transferring the transaction to the slave after extracting the original transaction for the replication, the execution time of the transaction ID may be different from the original due to the parallel apply. In thi s case, the slave changes the separate transaction ID into the internally distinguishable ID to prevent ident ity duplication by reusing the transaction ID.

This internal transaction ID is allocated by the distributor, but it uses the original value instead of the inter nal value while analyzing specific logs, so the problem occurs, which a single transaction has two transact ion IDs.

Two transaction ID values used as a delimiter value of concurrency control in distributor are allocated to a single transaction, so self dead-lock may occur in a specific situation.

Symptom

```
CREATE TABLE T1 ( C1 INTEGER PRIMARY KEY, C2 LONG VARCHAR );
INSERT INTO T1 VALUES( 1, 'AAA' );
DELETE FROM T1 WHERE C1=1;
INSERT INTO T1 VALUES( 1,'AAAAAAA .......' );  ❶ INSERT the data over 8K
COMMIT;
```

As above, if performing the query processing the same key value and INSERT which inserts the record ove r 8K in a single transaction, then the dead-lock occurs.

Workaround

The patch is required.

ISSUE-5499 If the argument of the window function is a scalar subquer y expression, then the value is not evaluated.

Description

If the scalar subquery expression which can become a constant is used as the argument of the window fu nction in the cluster environment, then the expression is not evaluated when executing the query, so the result is NULL.

Symptom

```
--# BUGBUG
--# result : 1
SELECT SUM( ( SELECT 1 FROM dual ) ) OVER() AS C_NAME
  FROM dual@G2;
C_NAME
------
  null
```

In the cluster environment, all expressions which can become constants are evaluated on the driver node and the result value is transferred to the generated query.

When using the subquery expression as an argument of the window function, then it did not determine whether to configure the constant, so it did not become a constant. If it is required to transfer the result v alue of the subquery expression through the generated query, then a result error occurs.

If the query is performed only on the local node in the standalone or the cluster environment, then the ex pression in the window function is evaluated even when it is not become a constant, so an error does not

occur.

Workaround

The patch is required.

ISSUE-5505 If the access method for leftmost table in the join is the unique index access, and only part of key columns in the group by belong to the unique index, then the result may be wrong.

Description

If the access method for leftmost table in the join is the unique index access, and only part of key columns in the group by belong to the unique index, then the result may be wrong.

Symptom

```
DROP TABLE IF EXISTS r;
DROP TABLE IF EXISTS s;
DROP TABLE IF EXISTS t;
CREATE TABLE r( c1 INTEGER, c2 INTEGER, c3 INTEGER, c4 INTEGER, c5 INTEGER );
INSERT INTO r VALUES(1, 1, 1, 1, 1);
INSERT INTO r VALUES(1, 2, 1, 1, 1);
INSERT INTO r VALUES(1, 2, 1, 1, 1);
INSERT INTO r VALUES(2, 1, 2, 1, 1);
INSERT INTO r VALUES(2, 1, 2, 2, 1);
INSERT INTO r VALUES(3, 1, 2, 2, 1);
INSERT INTO r VALUES(3, 2, 3, 2, 1);
INSERT INTO r VALUES(4, 1, 3, 3, 1);
INSERT INTO r VALUES(5, 1, 3, 3, 1);
INSERT INTO r VALUES(1, 3, 2, 1, 1);
INSERT INTO r VALUES(1, 1, 1, 1, 1);
COMMIT;
CREATE TABLE s( c1 INTEGER PRIMARY KEY, c2 INTEGER, c3 INTEGER, c4 INTEGER, c5 INTEGER );
INSERT INTO s VALUES(1, 1, 1, 1, 1);
INSERT INTO s VALUES(2, 3, 1, 1, 1);
INSERT INTO s VALUES(3, 3, 1, 1, 1);
INSERT INTO s VALUES(4, 2, 2, 1, 1);
INSERT INTO s VALUES(5, 2, 2, 2, 1);
INSERT INTO s VALUES(6, 1, 2, 2, 1);
INSERT INTO s VALUES(7, 1, 3, 2, 1);
```

```
INSERT INTO s VALUES(8, 3, 3, 3, 1);
INSERT INTO s VALUES(9, 3, 3, 3, 1);
COMMIT;
CREATE TABLE t( c1 INTEGER, c2 INTEGER, c3 INTEGER, c4 INTEGER, c5 INTEGER );
INSERT INTO t VALUES(1, 1, 1, 1, 1);
INSERT INTO t VALUES(1, 3, 3, 1, 1);
INSERT INTO t VALUES(1, 3, 3, 1, 1);
INSERT INTO t VALUES(1, 2, 2, 1, 1);
INSERT INTO t VALUES(1, 2, 2, 1, 1);
INSERT INTO t VALUES(2, 3, 1, 1, 1);
INSERT INTO t VALUES(3, 3, 1, 1, 1);
INSERT INTO t VALUES(4, 2, 2, 1, 1);
INSERT INTO t VALUES(5, 2, 2, 2, 1);
COMMIT;
--# result : 6 rows
\EXPLAIN PLAN
SELECT s.c1, s.c2, t.c3
  FROM s, r, t
 WHERE s.c1 > 0 AND s.c1 < 5
   AND s.c1 = r.c1
   AND r.c1 = t.c1
 GROUP BY s.c1, s.c2, t.c3;
C1 C2 C3
-- -- --
 1  1  2
 1  1  3
 1  1  1
 1  1  2
 1  1  3
 1  1  1
 1  1  2
...
18 rows selected
< Execution Plan >
================================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                     ¦
--------------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                    ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                       ¦
¦    2  ¦      GROUP                                                         ¦
¦    3  ¦        HASH JOIN (INNER JOIN)                                      ¦
```

```
|    4  |          MERGE JOIN (INNER JOIN)                              |
|    5  |            INDEX ACCESS ("S", "S_PRIMARY_KEY_INDEX")          |
|    6  |            INDEX ACCESS ("R", "R_IDX")                        |
|    7  |          HASH JOIN INSTANT                                    |
|    8  |            TABLE ACCESS ("T")                                 |
=====================================================================
```

The result from the query above should be 6 rows, but actually 18 rows are output. The leftmost table s u ses 'S_PRIMARY_KEY_INDEX', so the join result is sorted for s.c1. In the plan above, group by performs t he grouping by using the sorted result of the join. However, the sorted records are not sorted for all grou p key columns, so it may lead to the wrong result.

Workaround

Use */*+ USE_GROUP_HASH */* hint.

ISSUE-5500 GoldilocksPreparedStatement.setFixedCHAR(), a JDBC non standard function, has been added.

Description

GoldilocksPreparedStatement.setFixedCHAR(), a JDBC non standard function, has been added.

Symptom

If binding a CHAR column to the WHERE clause in a SELECT statement by using PreparedStatement.setSr ting(), then the result is not found.

```
create table x (c char(4));
insert into x (c) values ('a');  -- inserts 'a   '
```

```
PreparedStatement stmt =
  conn.prepareStatement("select * from x where c = ?");
stmt.setString(1, "a");    // This won't return any records
stmt.executeQuery();
```

Workaround

Change the CHAR column to a VARCHAR column.

ISSUE-5482 If the logs are insufficient during the incomplete recovery, then the recovery fails.

Description

The recovery fails during the incomplete recovery even though it could be completed, and this error has b
een fixed.

Symptom

If performing the incomplete recovery by using the backup redo logs because the redo log is lost, and the backup redo log is earlier than the archive log, then it can not find the log later than the archive log, so th
e error occurs.

```
gSQL> ALTER DATABASE RECOVER UNTIL TIME '2023-04-05 19:05:01.559752';
ERR-HY000(14068): logfile does not exist -
'/goldilocks/goldilocks_data/archive_log/archive_4.log'
```

Workaround

If the redo log later than the archive log does not exist, then perform an interactive incomplete recovery which performs the recovery by using undefected log only as follows.

```
gSQL> ALTER DATABASE BEGIN INCOMPLETE RECOVERY;
ERR-01000(14104): Warning: suggestion '/goldilocks/archive_log/archive_0.log'
ERR-01000(14103): Warning: media recovery needs a logfile including log (Lsn 139992)
Database altered.
gSQL> ALTER DATABASE RECOVER AUTOMATICALLY;
ERR-01000(14104): Warning: suggestion '/goldilocks/archive_log/archive_4.log'
ERR-01000(14103): Warning: media recovery needs a logfile including log (Lsn 144143)
Database altered.
gSQL> ALTER DATABASE END INCOMPLETE RECOVERY;
Database altered.
```

ISSUE-5445 Even if LOGFILE GROUP size is set sufficiently it fails to add LOGFILE GROUP when performing ADD LOGFILE GROUP.

Description

Even if LOGFILE GROUP size is set sufficiently it fails to add LOGFILE GROUP with a error message saying the logfile is smaller than the minimum size.
It is because the minimum size of the logfile should be calculated based on the number of log buffers and pending log buffers which were revised during the startup, but actually it is calculated based on the property value when performing ADD LOGFILE GROUP.

Symptom

If attempting to add a logfile group after starting up to mount phase with PROPERTY set as follows, then it fails.

```
LOG_BUFFER_SIZE=1G
PENDING_LOG_BUFFER_COUNT=32
```

```
gSQL> STARTUP MOUNT
Startup success
gSQL> ALTER DATABASE ADD LOGFILE GROUP 4 ('redo_4_0.log') SIZE 512M;
ERR-42000(16198): size of log file is smaller than minimum size of log file(1107296256 bytes).
```

Workaround

Create a logfile group by revising LOG_BUFFER_SIZE property and PENDING_LOG_BUFFER_COUNT property.

ISSUE-5424 If executing SELECT statement of an embedded SQL without INTO clause, then an error occurs even when the data exists, and this error has been fixed.

Description

If repeatedly executing SELECT statement of an embedded SQL without INTO clause, then an error occurs, and this error has been fixed.

Symptom

```
EXEC SQL SELECT 1 FROM DUAL;
EXEC SQL SELECT 1 FROM DUAL;
```

If repeatedly executing the same SELECT statement as given above, then the following error occurs.

```
Invalid cursor state : A cursor was open on the StatementHandle.
```

Workaround

Add INTO clause to the SELECT statement, then execute it.

ISSUE-5411 It throws NullPointerException when using getBinaryStream method of ResultSet class in JDBC, and this error has been fixed.

Description

It throws NullPoinerException when using getBinaryStream method of ResultSet class to get null data of long varbinary type, and this error has been fixed.

Symptom

C_BLOB of LONG VARBINARY type in table TEST_LOB has NULL data.

```
CREATE TABLE PUBLIC.TEST_LOB
(
   ID NUMBER( 10, 0 ),
   C_BLOB LONG VARBINARY
);
INSERT INTO TEST_LOB VALUES(1,UNHEX(HEX('XXXXXXXX')));
INSERT INTO TEST_LOB VALUES(2,null);
COMMIT;
```

The following is a code which uses getBinaryStream method to get the data of LONG VARBINARY type.

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT ID, C_BLOB FROM TEST_LOB");
while (rs.next()) {
    try{
        InputStream is = rs.getBinaryStream("c_blob");
        byte[] bytes = new byte[0];
        bytes = new byte[is.available()];
        is.read(bytes);
    } catch( Exception e){
        System.out.println(e);
    }
}
```

When executing the code above, it throws java.lang.NullPointerException.

Workaround

The patch is required.

ISSUE-5398 The data is lost when gloader uploads the data in the text mode, and this error has been fixed.

Description

gloader uses the field delimiter and the line delimiter which start with the same character when uploadin g the data in the text mode. If the data includes the first character of these delimiter, then the data is tru ncated and uploaded in an invalid form, and this error has been fixed.

Symptom

The following is an example of a data file.

```
data 1^^^Cc__Cc^data 2^Rr__Rr
```

The following is an example of a control file.

```
TABLE TEST
FIELD TERMINATED BY '^Cc__Cc^'
LINE TERMINATED BY '^Rr__Rr\n'
```

If uploading the data by using the control file and the data file above, then the data is truncated.

```
gloader test test -i --control test.ctl --data test.dat --array 1 --no-copyright
COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 3 RECORDS, SUCCEEDED 3 RECORDS
gSQL> SELECT * FROM TEST;
C1      C2
------- -------------
data 1^ null
^       null
null    data 2^Rr__Rr
```

Workaround

Use the the field delimiter and the line delimiter which start with the different character each other.

ISSUE-5407 Cluster peer which is waiting for the lock to be release can not recognize that driver node is killed.

Description

If the driver member is abnormally terminated while waiting for the lock to be released on the remote, th en the remotely created session remains alive.

Symptom

The following is an example of an environment that the cluster group G1 has G1N1, G1N2 as members, and the table T1 is created.

The record of table T1 is updated on G1N1.

```
gSQL> UPDATE T1 SET A = A + 1 WHERE A = 1;
1 row updated.
```

If the same record is updated on G1N2, then it will wait.

```
gSQL> UPDATE T1 SET A = 10 WHERE A = 1;
```

When checking the session on G1N1, then it is seen that the cluster session is waiting for the update sent from G1N2.

```
gSQL> SELECT SESSION_STATUS FROM V$SESSION@G1N1
       WHERE PROGRAM_NAME = 'cluster peer';
SESSION_STATUS
--------------
CONNECTED
```

Even when gsql waiting for the update sent from G1N2 is killed, the session still remains alive on G1N1 a s follows.

```
gSQL> SELECT SESSION_STATUS FROM V$SESSION@G1N1
       WHERE PROGRAM_NAME = 'cluster peer';
SESSION_STATUS
--------------
CONNECTED
```

Workaround

The patch is required.

22c.1.2 Patch Notes

ISSUE-5174 gpec supports declaring function arguments.

Description

gpec supports declaring function arguments. For more information, refer to **Declaring Function Argument**.

Symptom

N/A

Workaround

The patch is required.

ISSUE-5353 When connecting and disconnecting by using the window ODBC, then the number of program handles increase and this error has been fixed.

Description

When repeatedly connecting and disconnecting by using the window ODBC, then the number of entire program handles increase and this error has been fixed.

Symptom

When repeatedly connecting and disconnecting by using the window ODBC, then the number of entire program handles increase.

Workaround

The patch is required.

ISSUE-5344 When performing view projection pruning it deletes the column used in the upper block.

Description

If the following conditions are satisfied, the server may be abnormally terminated due to improperly performed view projection pruning.

- *group by* or *order by* exists within a view.
- expr to be deleted from the view's select list is an argument of another function expression.

Symptom

The following is a sample query which can cause an error.

```
SELECT sum_col1
  FROM ( SELECT sum(col1) as sum_col1
              , DECODE( sum(col1), NULL, 0 ) as decode_sum_col1
           FROM t1
          GROUP BY col2
       ) v1;
```

decode_sum_col1 is not used in the upper block in v1, so it is pruned. Moreover, sum(col1) which is an argument of DECODE is also pruned. However, sum(col1) is already specified in select list and it is used in the view's upper block, so it should not be deleted.

Workaround

The patch is required.

ISSUE-5336 When using SUBQUERY in SELECT INTO statement of gpec, then it is not processed as a SELECT INTO statement.

Description

If gpec parses the SQL which has SUBQUERY after a SELECT INTO statement, then it is not processed as a SELECT INTO statement.

Symptom

The following is an example of using SUBQUERY after using the host variable array in a SELECT INTO statement.

```
EXEC SQL BEGIN DECLARE SECTION;
int no[10];
int count[10];
EXEC SQL END DECLARE SECTION;
EXEC SQL SELECT empno, B.COUNT
    INTO :no, :count,
    FROM emp, (SELECT count(*) as COUNT FROM emp);
```

When performing the example above, then the following error occurs.

```
SQLCODE :-16289
SQLSTATE: 42000
ERROR MSG : into clause can have only one row
```

Workaround

Fetch the cursor instead of using the host variable array in a SELECT INTO statement.

ISSUE-5277 A hang occurs because buffers cannot be reused after deleting a disk tablespace.

Description

If the disk tablespace is deleted, then the ager thread moves the buffer cache which cached the the deleted tablesapace to the free list. In this case, it only displays that the dirty pages should be discarded. However, if the checkpoint does not occur, then the discarded pages can not be reused, so the hang occurs in the session looking for the free buffer, and this error has been fixed.

Symptom

If the buffers which cached pages of the deleted disk tablespace become the dirty page, then the session accessing the disk tablespace can not get the free buffer, so the hang occurs.

Workaround

Clear the dirty pages of the deleted disk tablespace by performing the checkpoint.

ISSUE-4945 Use SQLTables to check the existence of the table while creating a table to operate CYCLONE, CLUSTONE, CYMON.

Description

It has been modified to create the table after checking the existence of the table required to operate CYCLONE, CLUSTONE, CYMON by using SQLTables.

Symptom

N/A

Workaround

Validate the table existence when preparing, then use the result to determine whether the table exist.

ISSUE-5218 Using async commit leads to an excessive use of transaction by a single session.

Description

Async commit in the cluster environment can use a new transaction while the transaction is not terminated, so a single session may use two ore more transactions.

Symptom

An excessive use of transactions may cause the lack of transactions.

Workaround

The patch is required.

ISSUE-5029 Even when the master rejoins with reset all option while operating Cyclone in the cluster environment, but the existing replication information is not initialized.

Description

When restarting the previously operated master with reset all option while Cyclone is operating in the cluster environment, then it should not use the existing replication information, and the replication should be resumed from the current time.

Symptom

Even when restarting the master with reset all option, the recovery process uses the existing replication o perating information.

Workaround

The patch is required.

22c.1.1 Patch Notes

ISSUE-5132 It supports DML execution for the single domain table with out the global secondary index in the cluster environment.

Description

If the global secondary index is not configured when executing DML for the single domain table in cluster environment, then it may fail. DML query requires the global secondary index to guarantee the data consi stency between servers. Therefore, if the data is stored in a single server, and it is not required to guarant ee the data consistency between server, then it supports DML without the global secondary index.

It is recommended to configure the global secondary index to manage a single table in multiple servers in cluster environment.

Symptom

```
--# G4 group has G4N1 only.
CREATE TABLE r ( c1 INTEGER )
   CLONED
   AT CLUSTER GROUP g4
   WITHOUT GLOBAL SECONDARY INDEX;
--# result: success
INSERT INTO r VALUES (1), (2), (3);
--# Improvement
--# result: success
DELETE FROM r WHERE c1 = 2;
ERR-42000(16519): global secondary index expected in cluster DML
```

Workaround

The patch is required.

ISSUE-5193 When executing a query including index backward scan in cluster environment, then the result is wrong.

Description

When executing the user query in cluster environment, if it is required to access the remote server and th e user query includes index backward scan by ORDER BY statement or by a hint, then the query result ma y be in an order of index forward scan.

Symptom

If it is required to access the remote server when executing the user query in cluster environment, then th e generated query is configured and transferred to the remote server. However, the remote server perfor ms the index forward scan because the access path hint information is wrong when configuring the gene rated query including the index backward scan.

```
CREATE TABLE t1
(
    c1 INTEGER
)
    SHARDING BY RANGE ( c1 )
    SHARD s1 VALUES LESS THAN ( 10 )       AT CLUSTER GROUP g1,
    SHARD s2 VALUES LESS THAN ( MAXVALUE ) AT CLUSTER GROUP g2;
INSERT INTO t1 VALUES ( 12 );
INSERT INTO t1 VALUES ( 11 );
INSERT INTO t1 VALUES ( NULL );
CREATE INDEX t1_idx1 ON t1( c1 ASC NULLS LAST );
--# BUGBUG
--# result: 3 rows
--#         null
--#           12
--#           11
\EXPLAIN PLAN
SELECT c1 FROM t1@g2 ORDER BY c1 DESC NULLS FIRST;
  C1
----
  11
  12
```

```
null
3 rows selected.
```

Workaround

The patch is required.

ISSUE-5109 Cyclone malfunctions after shard rebalancing and split brai n, and this error has been fixed.

Description

When recovering after shard rebalancing and split brain, Cyclone is terminated with "internal error occurr ed (Not Need Rebalance)" error.

Symptom

Even when cyclone determines rebalancing is not required, but it is rebalanced in a specific situation. This may happen in shard rebalancing and split brain situation.

Workaround

Restart Cyclone with --reset.

ISSUE-5162 Even when SQL_ATTR_CONNECTION_TIMEOUT is set in O DBC, it may wait longer than the settings, and this error has been fixe d.

Description

Even when SQL_ATTR_CONNECTION_TIMEOUT is set in ODBC to detect the network disconnection, it m ay wait longer than the given timeout setting, and this error has been fixed.

Symptom

Even when SQL_ATTR_CONNECTION_TIMEOUT is set in ODBC, it can not detect the network disconnecti on in a specific situation, so it keeps waiting for the server's response in ODBC.

Workaround

Add the following attributes to odbc.ini, so that it can quickly detect the network disconnection.

- KEEPALIVE_IDLE_TIME

- KEEPALIVE_INTERVAL
- KEEPALIVE_COUNT

Or, alter the following kernel attributes, so that it can quickly detect the network disconnection.

- net.ipv4.tcp_keepalive_intvl
- net.ipv4.tcp_keepalive_probes
- net.ipv4.tcp_keepalive_time
- net.ipv4.tcp_retries2

ISSUE-5160 When LONGVARCHAR, LONGVARBINARY parameters exist in ODBC global connection environment, the client's memory increases, and this error has been fixed.

Description

When repeatedly performing the statement including LONGVARCHAR, LONGVARBINARY parameters in ODBC global connection environment, the client's memory increases, and this error has been fixed.

Symptom

When repeatedly performing the SQL including LONGVARCHAR, LONGVARBINARY parameters with the same statement in ODBC global connection environment, the client's memory increased.

Workaround

The patch is required.

ISSUE-5156 SQLSTATEs of some errors have been changed.

Description

SQLSTATEs of some errors have been changed.

Symptom

SQLSTATEs of some errors have been changed as follows.

Error number	Old SQLSTATE	Modified SQLSTATE	Message
13034	RD000	08S01	Service is not available
16351	08000	HY000	failed to connect to the cluster member '%s'
16523	HY000	08S01	the database system is shutting down

Error number	Old SQLSTATE	Modified SQLSTATE	Message
25001	HY000	08001	Server is not running

Workaround

The patch is required.

ISSUE-5147 Even when it was set as PROTOCOL=TCP in the configuration of CYMON, it was connected by using DA, and this error has been fixed.

Description

Even when it was set as PROTOCOL=TCP in the configuration of CYMON, it was connected by using DA. However, this error has been fixed, so it is connected by using TCP now.

Symptom

If it is set as PROTOCOL=TCP in the configuration of CYMON, it should be connected by using TCP, but, in reality, it is connected by using DA.

Workaround

The patch is required.

ISSUE-5004 A deadlock occurs on the slave's pre-process phase during the replication with CYCLONE in the cluster environment.

Description

A deadlock intermittently occurs during the replication with CYCLONE in the cluster environment.

Symptom

The replication is not proceeding and it seems to be stop. It is because a deadlock occurs during the pre-process for the replication in the cluster environment. The replication is not proceeding any more even when monitoring with CYMON.

Workaround

Reset the master and the slave of CYCLONE.

ISSUE-4985 The progressing information in slave has been added to the CYCLONE's monitoring information.

Description

The information being processed in slave (Apply_FileSeq, Apply_BlockSeq, Apply_Commit_Lsn) has been added to the CYCLONE's monitoring information.

Symptom

N/A

Workaround

The patch is required.

ISSUE-4882 It is modified to report the detatiled error message when an error occurs while processing CYCLONE SYNC.

개요

It is modified to record the detailed error message together with *ERROR OCCURRED* on the trace log when an error occurs while processing CYCLONE SYNC.

Symptom

N/A

Workaround

The patch is required.

Part II.

Administration Manual

5.

Basic Management of GOLDILOCKS Database

5.1 Creating and Configuring GOLDILOCKS Database

Creating Database

Create database using gcreatedb which is included in GOLDILOCKS package. Before creating database, the followings should be considered.

Table 5-1 Considerations when creating database

Considerations	For more information, refer to
Consider the space size of tables and indexes to be used in database.	• **Structure and Storage Structure of GOLDILOCKS Database**
Consider the location to create database files. It is because the database performance will be improved by properly distributing the files and dispersing the disk IO. For example, a user can allocate redo log file to a separate disk or stripe it, and then disperse data file into a number of disks, to execute a parallel disk IO operation.	• **Managing Redo Log File**
Be aware of each property's concepts and its operation which are set in server property file, then constantly manage them.	• **Specifying Initial Property** • **Managing Initial Property Using GOLDILOCKS Configuration File** • **Server Property**

In addition to the above considerations, refer to **Creating Database** in **Getting Started** for more options to be considered when creating database.

A user should set environment variables to use GOLDILOCKS, and the variables are $GOLDILOCKS_HOME and $GOLDILOCKS_DATA. The property file, *goldilock.properties.conf*, for creating and managing GOLDILOCKS database is in *$GOLDILOCKS_DATA/conf*.

- GOLDILOCKS_HOME: Binaries which are included in GOLDILOCKS package are installed here, and it can be overwritten during software version upgrading (The license backup is required.)
- GOLDILOCKS_DATA: Log file, data file, control file which are used by GOLDILOCKS database are installed here, and it can not be overwritten.

Specifying Initial Property

A user can use properties to control the information for operation and management in GOLDILOCKS.

Initial Property

Set the properties of when creating or starting the database as follows.

1. Setting system environment variable
    I. Change it by typing in the command window in which GOLDILOCKS is installed and database is created or operated.
    II. It is necessary to add the prefix before the property name, and the prefix is *GOLDILOCKS_*.

- The following is an example of changing SHARED_MEMORY_STATIC_SIZE to 100M.

```
export GOLDILOCKS_SHARED_MEMORY_STATIC_SIZE=100M
```

2. Property file: The prefix is not required, so change it directly in the property file.

- The following is an example of changing SHARED_MEMORY_STATIC_SIZE to 200M.
- Shared memory static size (100 M ~ 32 G)

```
SHARED_MEMORY_STATIC_SIZE = 200M
```

If the same property is set to system environment variable and property file, then the value of th e property file will be applied. For example, if SHARED_MEMORY_STATIC_SIZE is set to 100 M as the system environment variable, and SHARED_MEMORY_STATIC_SIZE is set to 200 M as the property file, then SHARED_MEMORY_STATIC_SIZE will be applied to 200 M when operating d atabase.

Managing Initial Property Using GOLDILOCKS Configuration F ile

Property file is in *$GOLDILOCKS_DATA/conf*, and is divided into two according to the file format.

- Text property file
    - File name: goldilocks.properties.conf
    - It is "property name = value" format file, and a user can directly edit it.

    - If a binary property file exists, it does not read text property file.

- Binary property file
    - It is created by the system, and a user can not edit it directly.
    - File name: goldilocks.properties.binary
    - Use gdump tool to retrieve the binary file's contents.

If a text property file and a binary property file exist together, it reads the binary property file only. The text property file is not processed. This binary property file is managed by user SQL (ALTER SYSTEM SET), and can be edited using SQL only.

For information, refer to **ALTER SYSTEM SET property_name**, **ALTER SYSTEM RESET property_name**.

- Editing text property file
    - Use "PROPERTY_NAME = VALUE" format.
    - Use '#' for annotation.
    - Property has one of the following three data types.
        - Character data type: Use a single quote (') to set the character value. Use *<GOLDILOCKS_DATA>* if the character value includes $GOLDILOCKS_DATA link.
        - Numeric data type: Do not use calculation for numeric data. For example, LOG_BUFFER_SIZE =1024 * 1024 is not allowed to use. Its size is selectable accordingly such as K (kilobyte), M (megabyte), G (gigabyte), T (terabyte), and P (petabyte)
        - Boolean data type: Allowed to use ON/OFF, ENABLE/DISABLE, 1/0 and TRUE/FALSE, YES/NO.

5.2 Starting up and Shutting down GOLDILOCKS Instance

This chapter describes starting up and shutting down of GOLDILOCKS instance.

Starting up Instance

GOLDILOCKS instance can be started only by a user with SYSDBA privilege.
GOLDILOCKS instance can be started by using Direct Attach (D/A) method and dedicated method of Client/Server (C/S). However, it can not be started by using shared method of C/S.

Multi-level Startup

GOLDILOCKS has multi-level startup procedure. Multi-level startup procedure enables to change the database's state by the intervention of administrator in each phase.

The phases are idle, nomount, mount, open, and each phase has the following features.

Idle Phase

The idle phase is a state in which an instance is not started up.

If connecting to gsql when an instance is not started, then it will be connected to idle instance as follows. In this phase, any sever command can not be executed except for \startup.

```
% gsql sys gliese --as sysdba
Connected to an idle instance.
gSQL> select * from dual;
ERR-08003(40044): connection does not exist
gSQL>
```

- Administrator operations in idle phase
  - Adjusting the property required to start up the instance
  - Transition to nomount using `\startup` in gsql

- Operation in an instance at the nomount transition
  - Starting up gmaster which is a daemon managing GOLDILOCKS instance
  - Starting up timer thread and cleanup thread in gmaster
  - Allocating and initializing Shared memory Static Area (SSA).

Table 5-2 Applied properties at the transition to nomount

Property name	Description
CLIENT_MAX_COUNT	Maximum number of connectable sessions
CONTROL_FILE_0 ~ 7	Path of control file
CONTROL_FILE_COUNT	The number of valid path among the path of control file
DATA_STORE_MODE	Store mode of GOLDILOCKS instance
PLAN_CACHE_SIZE	Maximum size of shared memory for plan cache
PROCESS_MAX_COUNT	Maximum number of connectable process
SHARED_MEMORY_ADDRESS	Address of shared memory
SHARED_MEMORY_STATIC_NAME	Name of shared memory
SHARED_MEMORY_STATIC_KEY	Key value for creating shared memory
SHARED_MEMORY_STATIC_SIZE	Size of shared memory to be created
SYSTEM_LOGGER_DIR	Path of system logger

Properties can not be changed in idle phase. An administrator changes the properties as follows.

To use an environment variable, then set GOLDILOCKS_[property_name] to a desired value and transit it t o nomount. Then, the properties will be applied.

```
% export GOLDILOCKS_CLIENT_MAX_COUNT=1000
```

To use 'SCOPE = FILE', record the property value to be changed in the file. The recorded property will be applied at nomount transition.

```
gSQL> alter system set client_max_count = 1000 scope = file;
System altered.
gSQL> \shutdown
Shutdown success
gSQL> \startup
Startup success
gSQL>
```

Nomount Phase

The nomount phase is a state in which it is not mounted to database, and only the gmaster process has b een started. A gmaster is a daemon which manages GOLDILOCKS instance.

The following describes how to transit idle phase to nomount phase.

```
% gsql sys gliese --as sysdba
Connected to an idle instance.
gSQL> \startup nomount
Startup success
```

```
gSQL>
```

- Administrator operations in nomount phase
  - Adjusting nomount property
  - For more information, refer to **ALTER SYSTEM {MOUNT | OPEN} DATABASE**, **ALTER DATABASE R ESTORE**.

- Operation in an instance at the mount transition
  - Loading the control file to database
  - Preparing for database recovery
  - Starting up threads in gmaster such as checkpoint, log flusher, page flusher, IO slave, archive log

Table 5-3 Updatable properties in nomount phase

Property name	Description
LOG_BUFFER_SIZE	Redo log buffer size
PARALLEL_LOAD_FACTOR	The number of threads for parallel operations after loading database
PARALLEL_IO_FACTOR	The number of parallel threads for loading database
PARALLEL_IO_GROUP_1 ~ 16	Datafile group at parallel loading
PENDING_LOG_BUFFER_COUNT	The number of delayed log buffers
TRANSACTION_TABLE_SIZE	Transaction table size
UNDO_RELATION_COUNT	The number of undo relations

Mount Phase

The mount phase is a state in which it is mounted to database, and the database recognizes the control fi le. All sections in the control file is controllable in this phase.

The following describes how to transit nomount phase to mount phase.

```
% gsql sys gliese --as sysdba
Connected to an idle instance.
gSQL> \startup nomount
Startup success
gSQL> alter system mount database;
System altered.
gSQL>
```

- Administrator operations in mount phase
  - Adjusting mount properties
  - **ALTER SYSTEM {MOUNT | OPEN} DATABASE**
  - **ALTER DATABASE ADD LOGFILE**
  - **ALTER DATABASE DROP LOGFILE**
  - **ALTER DATABASE RENAME GLOBAL TRANSACTION LOGFILE**

  - **ALTER DATABASE RENAME LOGFILE**
  - **ALTER DATABASE { ARCHIVELOG | NOARCHIVELOG }**
  - **ALTER DATABASE DELETE BACKUP**
  - **ALTER DATABASE REGISTER**
  - **ALTER DATABASE RECOVER**
  - **ALTER DATABASE RESTORE**
  - **ALTER SESSION SET property_name**
  - **ALTER SYSTEM RESET property_name**
  - **ALTER SYSTEM SWITCH LOGFILE**
  - **ALTER SYSTEM [KILL | DISCONNECT] SESSION**
  - **ALTER TABLESPACE name ADD [DATAFILE|MEMORY]**
  - **ALTER TABLESPACE name RENAME DATAFILE**
  - **ALTER TABLESPACE name [ONLINE|OFFLINE]**

- Operation in an instance at the open transition
  - Loading all data file used in an instance to shared memory
  - Performing instance recovery
  - Creating NOLOGGING index
  - Deleting objects or files which are not deleted by ager thread
  - Creating cache for dictionary objects
  - If "SHARED_SESSION" property is set to *YES*, process monitor thread in gmaster will be started up.
  - Process monitor thread executes balancer process, dispatcher process, shared-server process.

Table 5-4 Updatable property in mount

Property name	Description
ARCHIVELOG_FILE	Prefix name of archive file
IN_DOUBT_DECISION	Decision about in-doubt transaction
LOCK_HASH_TABLE_SIZE	Hash table size of lock administrator
LOG_MIRROR_MODE	Log mirroring mode
LOG_MIRROR_SHARED_MEMORY_STATIC_SIZE	Shared memory size for log mirroring
SUPPLEMENTAL_LOG_DATA_PRIMARY_KEY	Whether to perform supplemental logging at the database-level

Open Phase

The open phase is a state in which all the data file is loaded to memory, and it is ready for service. All operations are allowed in this phase.

```
% gsql sys gliese --as sysdba
Connected to an idle instance.
gSQL> \startup mount
```

```
Startup success
gSQL> alter system open database;
System altered.
gSQL>
```

Diagnosis

It is possible to execute several phases at a same time using a single command(\startup) when starting up instance. If a certain phase fails, the administrator should figure out in which phase the failure occurs. Check using to which phase the instance has come using V$INSTANCE. Then the administrator can continue operations of startup instance from the subsequent phase.

The following is an example of starting up instance to open phase after \startup failure.

```
% gsql sys gliese --as sysdba
Connected to an idle instance.
gSQL> \startup
ERR-42000(14051): media recovery required - 'TEST_TBS'
gSQL> select INSTANCE_STATUS from v$instance;
INSTANCE_STATUS
---------------
MOUNTED
1 row selected.
...
gSQL> alter system open database;
System altered.
```

Shutting down Instance

GOLDILOCKS instance can be terminated only by a user with SYSDBA privilege. It is not allowed to create new session during shutdown.

GOLDILOCKS instance can be shut down by Direct Attach (D/A) method and dedicated method of Client/Server (C/S), but it can not be shut down by shared method of C/S.

There are four types of instance shutdown mode, and they are shutdown normal, shutdown transactional, shutdown immediate and shutdown abort.

Shutdown Normal

The shutdown normal is set by default if a particular mode is not specified when shutting down. Use this type if a user wants to shut down an instance normally.

```
gSQL> \shutdown normal
Shutdown success
gSQL>
```

The followings describe the characteristics of shutdown normal.

- New session is not allowed.
- New transactios and statements are allowed in session which already have been connected.
- Waiting for all the sessions which are connected to the instance to be shut down.
- It does not execute instance recovery process when starting up the instance.

Shutdown Transactional

Use this type to shut down the currently proceeding transactions normally even when the session is forcib ly shut down.

```
gSQL> \shutdown transactional
Shutdown success
gSQL>
```

The followings describe the characteristics of shutdown transactional.

- Neither new session nor is new transaction allowed.
- New statements are allowed in ongoing transaction.
- Waiting for the ongoing transactions to be shut down.
- The session will be automatically terminated after the ongoing transactions is terminated.
- It does not execute instance recovery process when starting up the instance.

Shutdown Immediate

Use this type to terminate the instance when a user can not terminate current transactions.

```
gSQL> \shutdown immediate
Shutdown success
gSQL>
```

The followings describe the characteristics of shutdown immediate.

- Neither new session nor is new transaction allowed.
- Forcibly shut down ongoing sessions and transactions.
- Waiting until the background thread of the system to be completed.
- It does not execute instance recovery process when starting up the instance.

Shutdown Abort

Use this type when a user judged the instance is in an abnormal state.

```
gSQL> \shutdown abort
Shutdown success
gSQL>
```

The followings describe the characteristics of shutdown abort.

- Neither new session nor is new transaction allowed.
- Forcibly shut down ongoing sessions and transactions.
- Instantly shut down the background thread of the system.
- It executes instance recovery process when starting up the instance.

5.3 Managing Process

This chapter describes the background processes of GOLDILOCKS instance.

Master Process

Master process performs the asynchronous operation for database performance and monitoring. It consis ts of many threads.

The master process' executable file name is gmaster.

Checkpoint Thread

Checkpoint thread executes asynchronous checkpoint events which occur in the log flushing thread. The checkpoint event occurs whenever redo log file is switched.

Checkpoint event is executed asynchronously regardless of a user, and its log is recorded in system.trc as f ollows.

```
[2014-09-11 14:04:34.704465 THREAD(14497,140178383427328)] [INFORMATION]
[CHECKPOINT] begin
...
[2014-09-11 14:04:34.743933 THREAD(14497,140178383427328)] [INFORMATION]
[CHECKPOINT] save control file
[2014-09-11 14:04:34.759521 THREAD(14497,140178383427328)] [INFORMATION]
[CHECKPOINT] end
```

Log Flushing Thread

User transactions record redo log in log buffer, and it is periodically recorded in log file by log flushing thr ead.

Log switching occurs when redo log is recorded to the end of the log file, then it will be recorded in the n ext redo log file. When the log switching occurs, checkpoint event is transferred to a checkpoint thread.

If reusable log file does not exist when the log file is switched, all queries except for the read-only queries will wait until the reusable log file is created.

The message remains as follows in system.trc when logging is blocked.

```
...
[2014-09-11 14:31:44.315871 THREAD(19102,139674683647744)] [INFORMATION]
[LOG FLUSHER] disable logging - blocked lfsn(1)
...
```

Log Archiving Thread

The log archiving thread asynchronously archives redo log file. This thread is executed only when databas e is operated in ARCHIVELOG mode.

Log archiving is a part of checkpoint process, and it is executed by log archiving event in which the check point thread occurred.

The message remains as follows in system.trc when redo_0_0.log is archived to archive_0.log.

```
[2014-09-11 14:13:32.515996 THREAD(16913,140631135463168)] [INFORMATION]
[ARCHIVELOG BEGIN] LOG(/home/test/work/product/Gliese/home/wal/redo_0_0.log(0)) =>
ARCHIVE(/home/test/work/product/Gliese/home/archive_log/archive_0.log)
[2014-09-11 14:13:33.145850 THREAD(16913,140631135463168)] [INFORMATION]
[ARCHIVELOG END] (/home/test/work/product/Gliese/home/archive_log/archive_0.log) : SUCCESS
...
```

Ager Thread

Ager thread physically drops the logically dropped database objects.

DROP TABLE statement executes only logical drop operation to maintain statement level consistency in G OLDILOCKS. Therefore, the statement which was invoked before DROP TABLE can explore the records of dropped table even when DROP TABLE is executed.

The message remains as follows in system.trc when table and tablespaces are physically dropped.

```
[2014-09-11 14:13:37.966788 THREAD(16925,139892990408448)] [INFORMATION]
[AGER] aging table - object scn(4561), object view scn(4562), type(0), physical
id(25043954302976)
...
[2014-09-11 14:13:37.966917 THREAD(16925,139892990408448)] [INFORMATION]
[AGER] aging tablespace - object scn(4561), object view scn(4564), tablespace id(61)
```

Timer Thread

Timer thread asynchronously sets time to the system to reduce the time measuring cost of user transactio ns, and the user transactions read the time set in the system.
The time precision is set according to **TIMER_INTERVAL**, and the default value is 10 ms.

The followings describe the case of using the time set in the timer thread. The time error occurs as much as TIMER_INTERVAL value. For example, if TIMER_INTERVAL is 10 ms, then the time error is also 10 ms.

- Time out : QUERY_TIMEOUT, IDLE_TIMEOUT, DDL_LOCK_TIMEOUT, ...
- Message record time written in trace log
- Startup time of login statement or transaction.
- Time recorded in transaction commit redo log

Page Flusher & IO Slave Threads

It applies the updated data pages to the disk when a checkpoint occurs. The updated information is store d in multiple data files in a tablespace. For that, the page flusher thread distributes operations to IO slave threads per each tablespace and data files and manages them. Then the IO slave threads record the upda ted pages in the data file in parallel.

If tables and index pages stored in the disk tablespace are updated by caching them to the buffer, they ar e linked to the checkpoint list, and the pages updated for the reuse in the buffer cache are linked to the b uffer replace list. I/O slave threads reflect the pages in IO checkpoint list and the pages in the buffer repla ce list on the disk at the checkpoint or regularly.

Storing the updated pages as many as possible at a time improves performance. The number of written p ages at each time follow the properties in **MAXIMUM_FLUSH_PAGE_COUNT**.

After the updated pages are written to the data file, the message remains in system.trc is as follows.

```
[2014-09-11 14:13:38.329162 THREAD(16925,139893221086976)] [INFORMATION]
[IO SLAVE] flush datafile ( tablespace : 0, datafile : 0 )
[2014-09-11 14:13:38.552161 THREAD(16925,139893221086976)] [INFORMATION]
[IO SLAVE] flush datafile ( tablespace : 1, datafile : 0 )
[2014-09-11 14:13:38.587510 THREAD(16925,139893221086976)] [INFORMATION]
[IO SLAVE] flush datafile ( tablespace : 2, datafile : 0 )
[2014-09-11 14:13:38.587831 THREAD(16925,139893221086976)] [INFORMATION]
[IO SLAVE] flush datafile ( tablespace : 62, datafile : 0 )
[2014-09-11 14:13:38.620239 THREAD(16925,139893221086976)] [INFORMATION]
[IO SLAVE] flush datafile ( tablespace : 63, datafile : 0 )
```

Cleanup Thread

Cleanup thread cleans up the system resource asynchronously, and performs the following operations.

- It cleans up normally terminated session. GOLDILOCKS processes termination of a user session logical ly, and uses cleanup thread for physical termination of a session.
- It cleans up abnormally terminated session. If the session is using transactions, then they will be rolled back.
- It checks the time-out of snapshot statements. If a session exceeds the time-out, it is forced to be ter minated.

When cleaning up the abnormally terminated session, a message remains in system.trc as follows.

```
[2014-09-12 10:34:38.387349 THREAD(23003,140722556352256)] [WARNING]
[CLEANUP] cleaning session - env(19), session(20), transaction(FFFFFFFFFFFFFFFF),
program(gsql), pid(23209), thread(140080441665280)
[2014-09-12 10:34:38.387515 THREAD(23003,140722556352256)] [WARNING]
[CLEANUP] cleaning up 1 sessions
```

If a snapshot statement exceeds the time-out, a message remains in system.trc as follows.

```
[2014-09-12 10:49:21.842179 THREAD(3972,139706316711680)] [WARNING]
[CLEANUP] long statement timeout - pid(8029), thread(140053960505088), program(gsql),
statement start time(2014-09-12 10:48:49.963471)
```

If abnormally terminated session is terminated by 'kill-9' signal during changing shared memory when exc lusive latch is acquired, it is no longer possible to operate the database. In this situation, a message remai ns in system.trc as follows, and the instance should be terminated using *SHUTDOWN ABORT*.

```
[2014-09-12 11:12:58.809249 THREAD(15313,140671386121984)] [WARNING]
[CLEANUP] failed to cleaning session - server restart required
...... dead session in critical section - env(3), session(4), transaction(47001E0004),
pid(15296), thread(140178075756288)
```

Process Monitor Thread

The process monitor thread executes processes, and monitors them.

- It is executed only when "SHARED_SESSION" property is set to *YES*.
- It executes load-balancer (gbalancer), dispatcher(gdispatcher), shared-server(gserver), and re-execute s them when they are abnormally terminated.
- It does not monitor listener (glsnr) process.

Cluster Recover Thread

When starting up a node on a cluster system environment, if the node phases up to the mount phase, th en the cluster recover thread is created. If there is an in-doubt transaction, the cluster recover thread com municates with the cluster recover thread on the remote node, and recovers the in-doubt transaction.

If there is an in-doubt transaction, the cluster recover thread communicates with the cluster recover threa d on the remote node, and figures out the status of the in-doubt transaction. If the remote node is restar ted and it is not recovered yet, then it transfers a message requesting the preferential completion of reco very and recovers the in-doubt transaction status after the recovery is completed.

The in-doubt transaction statuses figured out through the remote node are NONE, PREPARE, COMMIT, a nd ROLLBACK. If the cluster recover thread received the response such as COMMIT, ROLLBACK from at le ast one remote node, then it performs COMMIT or ROLLBACK. If the cluster recover thread received the r esponse such as NONE, PREPARE from all remote nodes, then it performs ROLLBACK because COMMIT o r ROLLBACK never has been performed on any cluster node.

The in-doubt transaction recovery using the cluster recover thread leaves the following messages in syste m.trc.

```
[2018-11-22 16:52:00.466805 INSTANCE(G3N2) THREAD(7828,140557371479808)] [WARNING]
[CLUSTER RECOVER] begin recovery
[2018-11-22 16:52:00.467221 INSTANCE(G3N2) THREAD(7828,140557371479808)] [WARNING]
[CLUSTER RECOVER] commit in-doubt transaction - commit scn(999.0.439), global transaction
id(1.29294650), local transaction id(4)
[2018-11-22 16:52:00.468253 INSTANCE(G3N2) THREAD(7828,140557371479808)] [WARNING]
[CLUSTER RECOVER] rollback in-doubt transaction - commit scn(1000.439), global transaction
id(4.34406459), local transaction id(59)
[2018-11-22 16:52:00.469198 INSTANCE(G3N2) THREAD(7828,140557371479808)] [WARNING]
[CLUSTER RECOVER] commit in-doubt transaction - commit scn(1001.0.439), global transaction
id(5.35127356), local transaction id(60)
```

Failover Thread

When starting up a node on a cluster system environment, if the node phases up to the LOCAL OPEN pha se, then the cluster failover thread is created. Cluster failover thread performs the failover for the error no de by reselecting a coordinator or offlining when an error occurs on a specific node or the network in the cluster system.

In a situation requiring the failover, a normal node which acquired a failover lock among other normal no des communicates with failover threads of other nodes, then performs the failover.

The failover performed by the failover thread leaves the following messages in system.trc.

```
[2018-11-22 15:27:34.619208 INSTANCE(G1N1) THREAD(20140,140219957368576)] [INFORMATION]
[FAILOVER] begin - failover member(5)
[2018-11-22 15:27:34.619418 INSTANCE(G1N1) THREAD(20140,140219957368576)] [INFORMATION]
[FAILOVER] acquire failover lock - driver(0), target(5), driver seq(1)
[2018-11-22 15:27:34.619692 INSTANCE(G1N1) THREAD(20183,140317097449216)] [INFORMATION]
[CDISPATCHER-S2] disconnect member - target member(5)
[2018-11-22 15:27:34.619893 INSTANCE(G1N1) THREAD(20183,140317097449216)] [INFORMATION]
[CDISPATCHER-S2] finalize sender socket - member(5)
[2018-11-22 15:27:34.621860 INSTANCE(G1N1) THREAD(20140,140219957368576)] [INFORMATION]
[FAILOVER] acquire failover lock
...
[2018-11-22 15:27:38.726436 INSTANCE(G1N1) THREAD(20140,140219957368576)]
[INFORMATION][FAILOVER] member(5) has failovered
[2018-11-22 15:27:38.728624 INSTANCE(G1N1) THREAD(20140,140219957368576)] [INFORMATION]
[FAILOVER] release failover lock - driver(-1), target(5), driver seq(1)
[2018-11-22 15:27:38.728786 INSTANCE(G1N1) THREAD(20140,140219957368576)] [WARNING]
reset remote session map - member(5)
[2018-11-22 15:27:38.729679 INSTANCE(G1N1) THREAD(20140,140219957368576)] [INFORMATION]
[FAILOVER] finished
```

Buffer Checkpoint Flusher, Buffer Replace Flusher

The table and index pages stored in disk table spaces are cached in a buffer and linked to a checkpoint lis t as changes are made. The buffer checkpoint flusher reflects pages linked to the checkpoint list to the co rresponding data file. The buffer checkpoint flusher is performed when a checkpoint event occurs, or is p eriodically performed after sleeping for the period set in the **BUFFER_FLUSHING_INTERVAL** property. The number of buffer checkpoint flushers is determined by **BUFFER_CHECKPOINT_LIST_COUNT** property.

If the requested page does not exist in the buffer cache, then it fills the buffer cache by reading pages fro m the data file. When looking for the available space in the buffer caches, then it links the updated pages which are not currently used to the buffer replace list. Then, the buffer replace flusher reflects the pages l inked to the buffer replace list to the corresponding data file. The buffer replace flusher wakes up by the event when an available space does not exist because the buffer is full, or is periodically performed after s leeping for the period set in the **BUFFER_FLUSHING_INTERVAL** property. The number of buffer replace flu shers is determined by **BUFFER_FLUSH_THREADS** property.

Listener Process

Listener process enables remote access through the network in a client/server environment. Listener proc ess waits for a client connection using **LISTEN_PORT**. If connected in dedicated mode, new gserver starts, and connects a client. If connected in shared mode, it selects underloaded dispatcher (gdispatcher) using load-balancer (gbalancer), and connects a client.

gserver is a kind of operation server, and it executes a client request.

If LISTEN_PORT is already in use, the following error occurs.

```
% glsnr --start
ERR-HY000(11077): given address is already in use
```

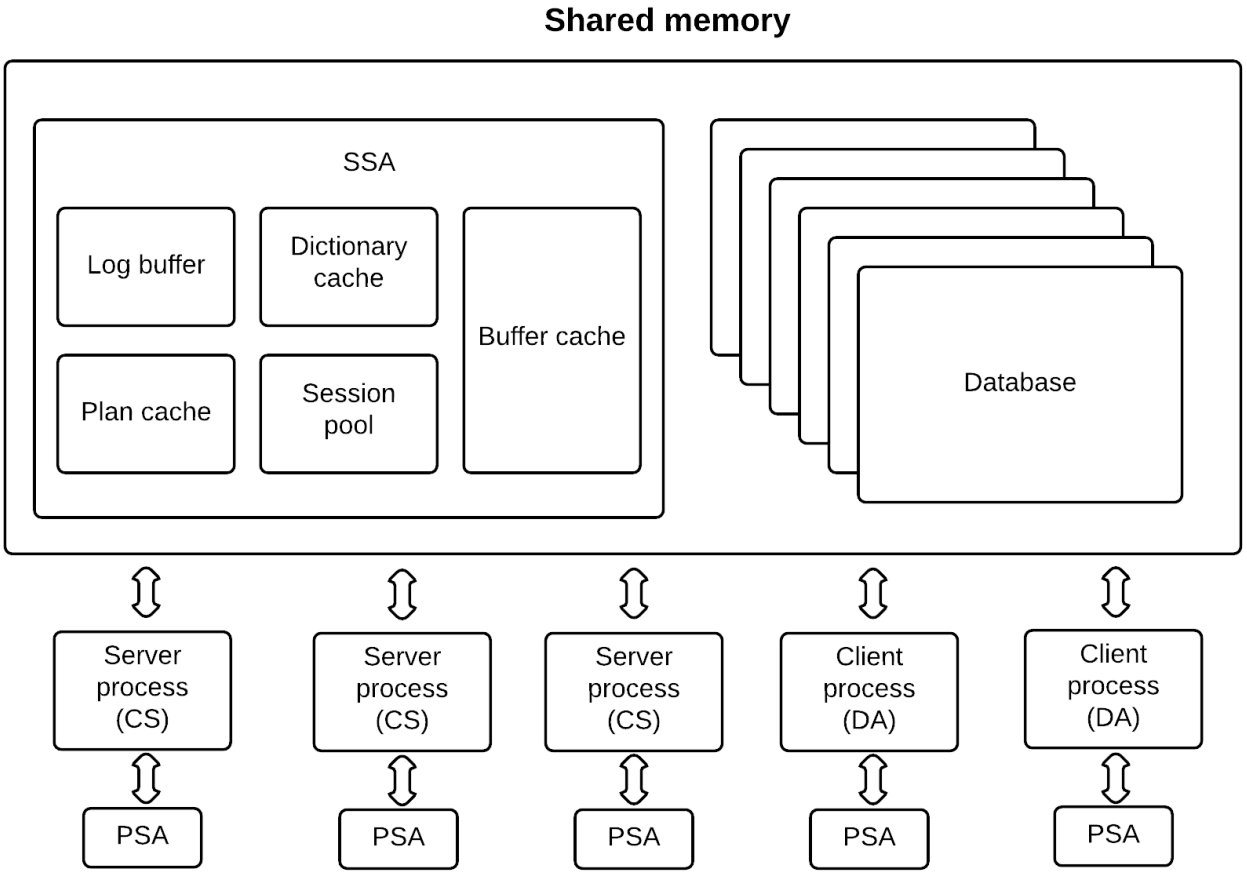
Listener process operates independently from the instance. In other words, listener process can start up a nd shut down regardless of instance start up anytime.

5.4 Managing Memory

GOLDILOCKS Memory Architecture

GOLDILOCKS uses SSA which is a memory shared by all sessions in the system, a shared memory for data base pages, and PSA (heap memory) which each session uses independently.

Figure 1 Shared memory



Managing SSA

Shared Static Area (SSA) is a memory area to store information which is shared by the system's all session s.

A new process should use the same physical address for using SSA. It is because the position of all inform ation referenced by SSA uses the physical address.

The physical starting address of SSA is determined by **SHARED_MEMORY_STATIC_KEY** and **SESSION_FATAL_BEHAVIOR**. The following error occurs when another program is already using shared memory key assigned by the same SHARED_MEMORY_STATIC_KEY, and the memory assigned by SHARED_MEMORY_ADDRESS.

```
% gsql sys gliese --as sysdba
Connected to an idle instance.
gSQL> \startup
ERR-HY000(11029): shared memory segment exists
gSQL>
```

SSA stores main information, and they are log buffer, dictionary cache, plan cache, session pool, lock pool, and transaction pool.

SSA size is determined by **SHARED_MEMORY_STATIC_SIZE**. The system automatically manages memories used in session/lock/transaction pool and dictionary cache, and a user can not arbitrarily manage them. However, a user can arbitrarily manage the usage of log buffer and plan cache.

If the default value of log buffer and plan cache are increased, SHARED_MEMORY_STATIC_SIZE should be increased accordingly. Otherwise, the following error occurs.

```
% gsql sys gliese --as sysdba
Connected to an idle instance.
gSQL> \startup
ERR-HY000(13010): Insufficient static area
gSQL>
```

Managing PSA

Private Static Area (PSA) is a heap memory area, and it is used independently by each session. **PRIVATE_STATIC_AREA_SIZE** determines the maximum size of PSA.

An initial size is allocated to PSA when a session is created. If additional memory is required in the session, PSA can be allocated up to its maximum size. The following error occurs when it exceeds the maximum size.

```
ERR-HY000(13011): Unable to extend memory: [MAX: 104857600, TOTAL: 102764408, ALLOC: 2097240]
DESC: private static area
```

5.5 Monitoring

Database monitoring is needed not only to detect and prevent the problem which can be an issue in the f uture, but also to find a way to improve database management. For monitoring, GOLDILOCKS database provides text file type trace log and several performance views.

Monitoring with Trace File

From when the instance starts up until it terminates, GOLDILOCKS database provides the system log whic h records the overall system error, warning, information. In addition, it provides XA transaction log, trace log such as DDL log, and SQL Trace Log. For more information, refer to **SQL Trace Log**.

Managing Trace Log File

GOLDILOCKS database's trace log file consists of system.trc file and xa.trc file. The system.trc file records system log and DDL log, and xa.trc file records XA transaction log. Trace log file is created in directory set in SYSTEM_LOGGER_DIR property. Therefore, it is generally created in trc directory below the directory s pecified in GOLDILOCKS_DATA environment variable. The trace log file's size is 10 Mbytes. If its space is i nsufficient, the existing trace log file with unique file extension is preserved, and the new trace log file is created.

Listener trace log file is created as *listener.trc* in trc directory below the directory specified by GOLDILOCK S_DATA environment variable. The log file size is 10 Mbytes. If the space is insufficient, the existing log fil e with unique file extension is preserved, and the new trace log file is created.

Except for system log, XA transaction log and DDL log can ON/OFF the monitoring. Set the TRACE_DDL p roperty value to 0 to turn off the DDL log, or set it to 1 to turn on the DDL log. XA log is set in the same way using the TRACE_XA property.

System Log

The errors, warnings, and information which occurred in database instance from when master process st arts up until it terminates are recorded in the system log.

System Log Format

System log is recorded in the following format.

```
['log record data and time' THREAD('process Id', 'thread handle')] ['log level']
['log prefix'] 'log body'
```

- 'log record date and time' is the date and time of the recorded log.
- THREAD('process Id', 'thread handle') is the information of log process id and thread handle.
- 'log prefix' is the entity or feature which created the log, and 'log body' is the detailed information.
- 'Log level' is the level of log recorded in system log, and its value is FATAL, ABORT, WARNING, INFO. 'log level' has the properties as follows.

Table 5-5 Log level properties

Log level	Description	Processing
FATAL	The state which master or client process is shut down abnormally.	The client should be reconnected in case of client process FATAL, and the database instance should be shut down and restarted in case of system FATAL. Backup the data file, control file, redo log file, system log file and contact the manufacturer.
ABORT	The state to continue service after rollback.	It is the normal state for system operation. Execute it again after eliminating the rollback cause.
WARNING	Operational warnings	The abnormal state of database instance. there is no operational problem but cause analysis is needed.
INFO	Operational information	-

For example, the following system log recorded the operational information at 17:30:55 on September 11 in 2014 by process id 21395 (The thread handle is 139982731163392). Log prefix is 'STARTUP-SM' and was executing the storage manager when the master process of GOLDILOCKS database started up. It means that the transition to NO-MOUNT phase had been done during multilevel startup.

```
[2014-09-11 17:30:55.758164 THREAD(21395,139982731163392)] [INFORMATION]
[STARTUP-SM] NO-MOUNT PHASE
```

Operational Information of GOLDILOCKS Database

It is the log for creating database instance, multilevel startup and shutdown, loading data file, recovering restart and media. It records the necessary information for the operation of the master process from when master process starts up until it terminates.

- GOLDILOCKS instance creation log

The system log records the following information when database instance is created. It creates the control file after the transition to NO-MOUNT phase to create database.

```
================================================
 Startup GOLDILOCKS
 TIME    : 2014-09-03 14:43:17.321020
================================================
[2014-09-03 14:43:17.321134 THREAD(14979,140542517491456)] [INFORMATION]
[STARTUP-SM] NO-MOUNT PHASE
[2014-09-03 14:43:17.321658 THREAD(14979,140542517491456)] [INFORMATION]
[STARTUP-SM] DATA_STORE_MODE(2)
[2014-09-03 14:43:17.335809 THREAD(14979,140542517491456)] [INFORMATION]
.... copy control file from '/goldilocks_data/wal/control_0.ctl' to
'/goldilocks_data/wal/control_1.ctl'
```

Then, it moves to transition to OPEN phase, and creates the system tablespace.

```
[2014-09-03 14:43:17.401356 THREAD(14979,140542517491456)] [INFORMATION]
[STARTUP-SM] MOUNT PHASE
[2014-09-03 14:43:19.319769 THREAD(14979,140542517491456)] [INFORMATION]
[STARTUP-SM] PRE-OPEN PHASE
[2014-09-03 14:43:19.320494 THREAD(14979,140542517491456)] [INFORMATION]
[STARTUP-SM] RECOVER TABLESPACE AND DATAFILE STATE
[2014-09-03 14:43:19.326269 THREAD(14979,140542517491456)] [INFORMATION]
[STARTUP-SM] OPEN PHASE
[2014-09-03 14:43:21.005536 THREAD(14979,140542517491456)] [INFORMATION]
[TABLESPACE] Create Tablespace(0)
[2014-09-03 14:43:21.005593 THREAD(14979,140542517491456)] [INFORMATION]
[TABLESPACE] Create Tablespace(1)
...
```

After creating the system tablespace, it executes checkpoint, and terminates the database instance.

```
[2014-09-03 14:43:21.788129 THREAD(14979,140542517491456)] [INFORMATION]
[CHECKPOINT] begin - checkpoint lid(0,10128,13), checkpoint lsn(10512), oldest lsn(10512)
[2014-09-03 14:43:21.788188 THREAD(14979,140542517491456)] [INFORMATION]
[CHECKPOINT] body - checkpoint lid(-1,0,0), checkpoint lsn(-1), active transaction count(0)
[2014-09-03 14:43:21.788203 THREAD(14979,140542517491456)] [INFORMATION]
[CHECKPOINT] end - checkpoint lid(0,10128,77), checkpoint lsn(10513)
[2014-09-03 14:43:21.788214 THREAD(14979,140542517491456)] [INFORMATION]
[CHECKPOINT] flush redo log
[2014-09-03 14:43:21.949589 THREAD(14979,140542517491456)] [INFORMATION]
[CHECKPOINT] save control file
[2014-09-03 14:43:21.957563 THREAD(14979,140542517491456)] [INFORMATION]
```

```
[SHUTDOWN-SM] CLOSE
[2014-09-03 14:43:21.957595 THREAD(14979,140542517491456)] [INFORMATION]
[SHUTDOWN-SM] POST CLOSE
[2014-09-03 14:43:21.992521 THREAD(14979,140542517491456)] [INFORMATION]
[SHUTDOWN-SM] DISMOUNT
[2014-09-03 14:43:21.992557 THREAD(14979,140542517491456)] [INFORMATION]
[SHUTDOWN-SM] INIT
```

- GOLDILOCKS instance startup log

Master process records logs such as multilevel startup of database instance, loading data file, restart reco very, media recovery. After the transition to MOUNT phase, the data file is loaded.

```
================================================
 Startup GOLDILOCKS
 TIME    : 2014-09-03 14:43:22.162601
================================================
[2014-09-03 14:43:22.162765 THREAD(14982,140025756808960)] [INFORMATION]
[STARTUP-SM] NO-MOUNT PHASE
[2014-09-03 14:43:22.163389 THREAD(14982,140025756808960)] [INFORMATION]
[STARTUP-SM] DATA_STORE_MODE(2)
[2014-09-03 14:43:22.429311 THREAD(14983,140025756808960)] [INFORMATION]
[STARTUP-SM] MOUNT PHASE
[2014-09-03 14:43:22.559395 THREAD(14983,140025756808960)] [INFORMATION]
[EVENT] system startup : SUCCESS
[2014-09-03 14:43:22.568526 THREAD(14981,139649517561600)] [INFORMATION]
[STARTUP] MOUNT PHASE
[2014-09-03 14:43:22.571200 THREAD(14983,140025756808960)] [INFORMATION]
[STARTUP-SM] LOAD DATAFILES
[2014-09-03 14:43:22.571241 THREAD(14983,140025756808960)] [INFORMATION]
.... datafile '/goldilocks_data/db/system_dict.dbf' assigned to PARALLEL_IO_GROUP_1
...
[2014-09-03 14:43:22.571562 THREAD(14983,140025280841472)] [INFORMATION]
.... LOAD DATAFILE(/goldilocks_data/db/system_dict.dbf)
...
```

After loading data file to the memory, it executes the recovery.

```
[2014-09-03 14:43:23.537256 THREAD(14983,140025756808960)] [INFORMATION]
[STARTUP-SM] REFINE TABLESPACE AND DATAFILE
[2014-09-03 14:43:23.631974 THREAD(14983,140025756808960)] [INFORMATION]
[RESTART REDO] begin
```

```
[2014-09-03 14:43:23.634374 THREAD(14983,140025756808960)] [INFORMATION]
[RESTART REOD] read checkpoint log - checkpoint log id(0,10128,13), oldest lsn(10512), system
scn(7)
[2014-09-03 14:43:23.756293 THREAD(14983,140025756808960)] [INFORMATION]
[RESTART REDO] ready to redo - start lid(0,10128,13), lsn(10512)
...
[2014-09-03 14:43:24.090755 THREAD(14983,140025756808960)] [INFORMATION]
[RESTART REDO] end - restart lsn(10514), restart scn(7)
[2014-09-03 14:43:24.091551 THREAD(14983,140025756808960)] [INFORMATION]
[RESTART UNDO] begin
[2014-09-03 14:43:24.091598 THREAD(14983,140025756808960)] [INFORMATION]
[RESTART UNDO] end
```

After recovery, it executes checkpoint, reflects the recovery result to disk data file. Then, it creates index, moves to the transition to OPEN phase.

```
[2014-09-03 14:43:24.111878 THREAD(14983,140025633163008)] [INFORMATION]
[CHECKPOINT] begin
...
[2014-09-03 14:43:24.129995 THREAD(14983,140025633163008)] [INFORMATION]
[CHECKPOINT] save control file
[2014-09-03 14:43:24.135864 THREAD(14983,140025633163008)] [INFORMATION]
[CHECKPOINT] end
[2014-09-03 14:43:24.144525 THREAD(14983,140025756808960)] [INFORMATION]
[STARTUP-SM] PRE-OPEN PHASE
[2014-09-03 14:43:24.202782 THREAD(14983,140025756808960)] [INFORMATION]
[STARTUP-SM] RECOVER TABLESPACE AND DATAFILE STATE
[2014-09-03 14:43:24.210158 THREAD(14983,140025756808960)] [INFORMATION]
[STARTUP-SM] REFINE RELATIONS
[2014-09-03 14:43:24.210304 THREAD(14983,140025756808960)] [INFORMATION]
[STARTUP-SM] REBUILD INDEXES
[2014-09-03 14:43:24.210375 THREAD(14983,140025756808960)] [INFORMATION]
[STARTUP-SM] OPEN PHASE
[2014-09-03 14:43:24.332064 THREAD(14983,140025756808960)] [INFORMATION]
[EVENT] system startup : SUCCESS
[2014-09-03 14:43:24.340843 THREAD(14981,139649517561600)] [INFORMATION]
[STARTUP] OPEN PHASE
```

- GOLDILOCKS instance termination log

When it shuts down database instance, it reflects all data file to the disk, executes checkpoint, then termi nates the master process.

```
[2014-09-03 14:48:03.467293 THREAD(15416,139855812097792)] [INFORMATION]
[IO SLAVE] flush datafile ( tablespace : 0, datafile : 0 )
...
[2014-09-03 14:48:03.748958 THREAD(15416,139855908558592)] [INFORMATION]
[PAGE FLUSHER] flushed lsn(137496), flushed page count(9216)]
[2014-09-03 14:48:03.761055 THREAD(15416,139856376227584)] [INFORMATION]
[CHECKPOINT] begin
...
[2014-09-03 14:48:03.780011 THREAD(15416,139856376227584)] [INFORMATION]
[CHECKPOINT] save control file
[2014-09-03 14:48:03.786387 THREAD(15416,139856376227584)] [INFORMATION]
[CHECKPOINT] end
[2014-09-03 14:48:03.791251 THREAD(15416,139856430274304)] [INFORMATION]
[SHUTDOWN-SM] CLOSE
[2014-09-03 14:48:03.791383 THREAD(15416,139856430274304)] [INFORMATION]
[SHUTDOWN-SM] POST CLOSE
[2014-09-03 14:48:03.824445 THREAD(15416,139856430274304)] [INFORMATION]
[SHUTDOWN-SM] DISMOUNT
[2014-09-03 14:48:03.824518 THREAD(15416,139856430274304)] [INFORMATION]
[EVENT] system shutdown : SUCCESS
[2014-09-03 14:48:04.267130 THREAD(15416,139856430274304)] [INFORMATION]
[SHUTDOWN-SM] INIT
```

If \shutdown abort forcibly stopped the server, neither checkpoint, nor is normal server shutdown execute d.

```
[2014-09-03 14:51:45.353154 THREAD(8989,139949509089024)] [INFORMATION]
[SHUTDOWN] skip CLOSE phase
[2014-09-03 14:51:45.678461 THREAD(8989,139949509089024)] [INFORMATION]
[SHUTDOWN] skip DISMOUNT phase
[2014-09-03 14:51:45.678696 THREAD(8989,139949509089024)] [INFORMATION]
[EVENT] system shutdown : SUCCESS
[2014-09-03 14:51:45.678928 THREAD(8989,139949509089024)] [INFORMATION]
[SHUTDOWN-SM] INIT
```

- The logs of master process checkpoint, log flusher, log archiving, ager, parallel IO, cleanup thread dur ing database operation

A checkpoint reflects to disk all data files which are updated only in memory and not written to the disk y et at checkpoint time. If it uses the parallel IO, it executes parallel IO in data file units. One unit of checkp oint log is from '[CHECKPOINT] begin' to '[CHECKPOINT] end'.

[IO SLAVE] is a log which is recorded by IO thread dedicated to parallel IO. '[IO SLAVE] flush data file (tablespace: 0, datafile: 0)' log is recorded after the data file (whose datafile id is '0' and whose tablespace id is 0) is reflected to the disk. These data file flush logs are repeatedly recorded as many as the number of data file at checkpoint time.

'[PAGE FLUSHER] flushed lsn(139039), flushed page count(9216)]' means that the reflected minimum Lsn in the disk is 139039, and reflected pages are 9216. The last log Lsn archives redo log files which are smaller than 139039. They record checkpoint log and control file, then stores them in the disk.

If a database is large its checkpoint time will be longer. Check if the datafile is continuously being recorded by tracking [IO SLAVE] logs. And if disk IO does not operate and stops, then check if log archiving is in progress. If the space is insufficient , then spare the space and ensure the log archiving proceed normally.

If checkpoint fails '[CHECKPOINT] CHECKPOINT was failed' log will be recorded. The checkpoint can be specially omitted at checkpoint time due to the log file switch, where '[CHECKPOINT] CHECKPOINT was skipped' log might be recorded.

```
[2014-09-12 15:54:59.654427 THREAD(13780,140493515450112)] [INFORMATION]
[CHECKPOINT] begin
[2014-09-12 15:54:59.654798 THREAD(13780,140493029623552)] [INFORMATION]
[IO SLAVE] flush datafile ( tablespace : 0, datafile : 0 )
[2014-09-12 15:54:59.835173 THREAD(13780,140493029623552)] [INFORMATION]
[IO SLAVE] flush datafile ( tablespace : 1, datafile : 0 )
[2014-09-12 15:54:59.893991 THREAD(13780,140493029623552)] [INFORMATION]
[IO SLAVE] flush datafile ( tablespace : 2, datafile : 0 )
[2014-09-12 15:54:59.926753 THREAD(13780,140493050603264)] [INFORMATION]
[PAGE FLUSHER] flushed lsn(138895), flushed page count(9216)]
[2014-09-12 15:54:59.926989 THREAD(13780,140492777965312)] [INFORMATION]
[ARCHIVING] stable lsn(139039)
[2014-09-12 15:54:59.933780 THREAD(13780,140493515450112)] [INFORMATION]
[CHECKPOINT] begin - checkpoint lid(0,55527,13), checkpoint lsn(139040), oldest lsn(139040)
[2014-09-12 15:54:59.933825 THREAD(13780,140493515450112)] [INFORMATION]
[CHECKPOINT] body - checkpoint lid(0,55527,77), checkpoint lsn(139041), active transaction
count(1)
[2014-09-12 15:54:59.933844 THREAD(13780,140493515450112)] [INFORMATION]
[CHECKPOINT] end - checkpoint lid(0,55527,155), checkpoint lsn(139042)
[2014-09-12 15:54:59.933859 THREAD(13780,140493515450112)] [INFORMATION]
[CHECKPOINT] flush redo log
[2014-09-12 15:54:59.936154 THREAD(13780,140493515450112)] [INFORMATION]
[CHECKPOINT] save control file
[2014-09-12 15:54:59.942850 THREAD(13780,140493515450112)] [INFORMATION]
[CHECKPOINT] end
```

Log flusher records the system log when the log buffer stops flushing to disk, and when it restarts the sto pped flusher. If the next log group is not reusable log file, the logging stops until it becomes reusable. Th e following describes an example when the logging is stopped because the redo log file of which sequen ce number is 34 has not been archived.

```
[2014-09-12 16:01:57.514303 THREAD(13780,140573333325568)] [INFORMATION]
[LOG FLUSHER] disable logging - blocked lfsn(34)
```

If the logging stops, then the transaction stops, so an immediate action is needed. When checkpoint is ex ecuted and archiving is done, the logging will restarts.

```
[2014-09-12 16:01:58.079236 THREAD(13780,140380267869952)] [INFORMATION]
[ARCHIVING] enable logging - blocked lfsn(34), inactivated lfsn(34)
```

Log archiving thread archives the redo log file of ACTIVE state, and it records the system log. One unit is f rom '[ARCHIVING] stable lsn(...)' to '[ARCHIVING] inactivate group ...'. If database is operated in archive l og mode, the redo file archiving log(from'[ARCHIVELOG BEGIN] ...' to '[ARCHIVELOG END] ...') is record ed.

```
[2014-09-02 17:41:56.762950 THREAD(20800,140129584793344)] [INFORMATION]
[ARCHIVING] stable lsn(144143)
[2014-09-02 17:41:56.763549 THREAD(20800,140129584793344)] [INFORMATION]
[ARCHIVELOG BEGIN] LOG(/goldilocks_data/wal/redo_0_0.log(8)) =>
ARCHIVE(/goldilocks_data/archive_log/archive_8.log)
[2014-09-02 17:41:57.385936 THREAD(20800,140129584793344)] [INFORMATION]
[ARCHIVELOG END] (/goldilocks_data/archive_log/archive_8.log) : SUCCESS
[2014-09-02 17:41:57.385987 THREAD(20800,140129584793344)] [INFORMATION]
[ARCHIVING] inactivate group #0(8)
```

If the log *'Archiving was failed - ...'* outputs after the log *'[ARCHIVELOG BEGIN] ...'* ,, then log archiving is failed. This failure should be immediately resolved in order that the service will be enabled by reusing the redo log file in active state.

The table whose ager is dropped and the aging information for the tablespace will be recorded as follows. When dropping the table, the table lock is also deleted, and the table scn information, scn which is able t o aging at aging time and the aging information of table lock will be recorded. If a table has an index, it will be deleted when dropping the table.

```
[2014-09-03 12:13:56.539971 THREAD(5225,139821699815168)] [INFORMATION]
[AGER] aging index - object scn(224), type(0), physical id(22634477649920)
[2014-09-03 12:13:56.540388 THREAD(5225,139821699815168)] [INFORMATION]
[AGER] aging table - object scn(224), object view scn(225), type(0), physical
```

```
id(22630182682624)
[2014-09-03 12:13:56.540491 THREAD(5225,139821699815168)] [INFORMATION]
[AGER] aging lock item - object scn(226), agable stmt scn(228), physical id(22630182682624)
```

When deleting a tablespace, tablespace scn, aging available scn at aging time and dropped tablespace id are recorded.

```
[2014-09-03 12:13:56.540553 THREAD(5225,139821699815168)] [INFORMATION]
[AGER] aging tablespace - object scn(224), object view scn(227), tablespace id(5)
```

The cleanup thread records the information about abnormally terminated session as follows. Even when a user session is abnormally terminated, the resources of the session is cleaned up, so the database instan ce or other users continue to operate.

```
[2014-09-03 13:43:02.220139 THREAD(7768,140298504156928)] [WARNING]
[CLEANUP] snipe at zombie session - pid(7766), thread(139967223228160), program(gsql)
[2014-09-03 13:43:02.220211 THREAD(7768,140298504156928)] [WARNING]
[CLEANUP] cleaning session - env(3), session(4), transaction(FFFFFFFFFFFFFFFF), program(gsql),
pid(7766), thread(139967223228160)
[2014-09-03 13:43:02.220270 THREAD(7768,140298504156928)] [WARNING]
[CLEANUP] cleaning up 1 sessions
```

- Log of creating, dropping, altering tablespace by a user

The system log records the operations of creating, deleting, updating of a user tablespace. Tablespace rel ated to DDL is recorded by default, regardless of TRACE_DDL ON/OFF. DDL failure will not be recorded in the system log. Therefore, a user should operate the system by setting TRACE_DDL to *ON* in order to figu re out more details about DDL log and the causes of its failure.

```
[2014-09-15 10:26:41.649909 THREAD(24881,140468897289984)] [INFORMATION]
[TABLESPACE] Create Tablespace(7)
[2014-09-15 10:26:55.966385 THREAD(24881,140468897289984)] [INFORMATION]
[DATAFILE] add datafile(/home/zkyungoh/work/product/Gliese/home/db/TEST1.dbf)
[2014-09-15 10:27:11.325897 THREAD(24881,140468897289984)] [INFORMATION]
[DATAFILE] Drop Datafile(/home/zkyungoh/work/product/Gliese/home/db/TEST1.dbf)
...
[2014-09-15 10:32:00.669550 THREAD(24881,140468897289984)] [INFORMATION]
[TABLESPACE] drop tablespace ( 7 )
```

- System internal error and index creation failure log

An internal error occurs when GOLDILOCKS database system error occurs but the exact cause of the failu

re can not be defined. If an internal error occurs, the SQL statement which raised the error is rolled back. The service is continuously available because the error does not affect the system nor does other sessions.

By the way, if the error raising SQL statement is executed again, it could be failed for the same reason, or the cause of failure could disappear and the operation could succeed. Therefore, a user should not chang e the database at the point of failure, but should request cause analysis to find the cause.

The index creation fails if the same key index on a table exists when creating UNIQUE index. Even though the index creation fails, it does not affect the index tables which are already created. Therefore, it does n ot affect the service.

```
[2014-09-15 11:26:59.640345 THREAD(7819,140737354012416)] [INFORMATION]
Index creation failed ( physical id : 22638772617216, error code : 14016 )
```

XA Log

It records the success or failure log of start, close, end, rollback, prepare, commit, recover, forget operatio ns of XA transaction interface for processing distributed transactions. GOLDILOCKS database does not re cord XA trace log by default. TRACE_XA should be set to *ON* to record the XA trace log as follows. For m ore information, refer to **XA API References**.

```
gSQL> alter system set trace_xa = yes;
System altered.
```

XA trace log is recorded in 'xa.trc' as follows. At first, executed XA interface is recorded, and then the sta tus (*complete* or *failed*) is recorded. The information such as session id and transaction id is also recorded. If it fails, the error code defined in **XA API References** is recorded.

```
[2014-09-15 11:45:19.599018 THREAD(7966,139931504572160)] [INFORMATION]
xa_start() complete - session(4), xid(0.3231.00), flags(0)
[2014-09-15 11:45:19.599360 THREAD(7966,139931504572160)] [INFORMATION]
xa_end() complete - session(4), xid(0.3231.00), flags(4000000)
[2014-09-15 11:45:19.599418 THREAD(7966,139931504572160)] [INFORMATION]
xa_prepare() complete - session(4), xid(0.3231.00), flags(0)
[2014-09-15 11:45:22.864563 THREAD(7966,139931504572160)] [INFORMATION]
xa_recover() complete - session(4), xid(), flags(1000000)
[2014-09-15 11:45:22.864829 THREAD(7966,139931504572160)] [INFORMATION]
xa_commit() complete - session(4), xid(0.3231.00), flags(0)
[2014-09-15 11:45:22.864887 THREAD(7966,139931504572160)] [INFORMATION]
xa_rollback() complete - session(4), xid(0.3232.00), flags(0)
[2014-09-15 11:45:22.885951 THREAD(7966,139931504572160)] [INFORMATION]
xa_forget() complete - session(4), xid(0.3230.00), flags(0)
```

```
[2014-09-15 11:45:22.886017 THREAD(7966,139931504572160)] [INFORMATION]
xa_forget() failed - session(4), xid(0.3231.00), flags(0), xa_error(-4)
```

DDL Log

For all DDL (creating, dropping, altering) generated in GOLDILOCKS database, the DDL generating sessions, overall SQL statements, status of success or failure are added in system log. In GOLDILOCKS database, DDL log is not recorded by default. TRACE_DDL should be set to *ON* as follows to record DDL trace log.

```
gSQL> alter system set trace_ddl = yes;
System altered.
```

The followings describe an example of which DDL log is recorded when a tablespace is created using the DDL.

```
gSQL> CREATE TABLESPACE TEST_TBS1
DATAFILE 'TEST_TBS1_01.dbf' SIZE 10M,
                      'TEST_TBS1_02.dbf' SIZE 10M,
                      'TEST_TBS1_03.dbf' SIZE 10M;
Tablespace created.
```

```
[2014-09-15 12:26:29.209210 THREAD(8149,140267442067200)] [INFORMATION]
[SESSION:11][DDL success] CREATE TABLESPACE TEST_TBS1
DATAFILE 'TEST_TBS1_01.dbf' SIZE 10M,
                      'TEST_TBS1_02.dbf' SIZE 10M,
                      'TEST_TBS1_03.dbf' SIZE 10M
[2014-09-15 12:26:29.209277 THREAD(8149,140267442067200)] [INFORMATION]
[SESSION:11][COMMIT with DDL]
```

If DDL statement fails, 'DDL failure' is recorded as follows.

```
gSQL> ALTER TABLESPACE TEST_TBS1 ADD DATAFILE 'TEST_TBS1_04.dbf' SIZE 10M;
ERR-42000(16130): file is already exist -
'/home/zkyungoh/work/product/Gliese/home/db/TEST_TBS1_04.dbf' :
ALTER TABLESPACE TEST_TBS1 ADD DATAFILE 'TEST_TBS1_04.dbf' SIZE 10M
                                        *
ERROR at line 1:
```

```
[2014-09-15 12:45:08.598789 THREAD(8115,140191085913856)] [INFORMATION]
[SESSION:4][DDL failure] ALTER TABLESPACE TEST_TBS1 ADD DATAFILE 'TEST_TBS1_01.dbf' SIZE 10M
```

DDL log for table or index DDL statement is recorded in the same way. After creating a table or an index,

the committed DDL log is as follows.

```
gSQL> CREATE TABLE T1 ( I1 NATIVE_INTEGER ) TABLESPACE TEST_TBS1;
Table created.
gSQL> CREATE INDEX T1X ON T1 ( I1 );
Index created.
gSQL> COMMIT;
Commit complete.
```

```
[2014-09-15 12:40:37.887952 THREAD(8115,140191085913856)] [INFORMATION]
[SESSION:4][DDL success] CREATE TABLE T1 ( I1 NATIVE_INTEGER ) TABLESPACE TEST_TBS1
[2014-09-15 12:40:47.451806 THREAD(8115,140191085913856)] [INFORMATION]
[SESSION:4][DDL success] CREATE INDEX T1X ON T1 ( I1 )
[2014-09-15 12:40:51.017975 THREAD(8115,140191085913856)] [INFORMATION]
[SESSION:4][COMMIT with DDL]
```

The following is a rollback DDL log after creating a table or an index.

```
[2014-09-15 12:42:27.367722 THREAD(8115,140191085913856)] [INFORMATION]
[SESSION:4][DDL success] CREATE TABLE T1 ( I1 NATIVE_INTEGER ) TABLESPACE TEST_TBS1
[2014-09-15 12:42:31.317436 THREAD(8115,140191085913856)] [INFORMATION]
[SESSION:4][DDL success] CREATE INDEX T1X ON T1 ( I1 )
[2014-09-15 12:42:34.601738 THREAD(8115,140191085913856)] [INFORMATION]
[SESSION:4][ROLLBACK with DDL]
```

Trace Log Replication

Replication trace log is recorded in a separate file when using GOLDILOCKS' replication tool such as CYCL ONE and LOGMIRROR. For more information about replication trace log, refer to **operating CYCLONE** in CYCLONE chapter and **operating** in LOGMIRROR chapter.

Listener Log

The errors and information which occurs from the startup of listener process until the end of the process are recorded in the listener log.

Listener Log Format

The listener log is recorded in the following format.

```
['log recorded date and time' THREAD('process id', 'thread handle')]
['log prefix'] 'log body'
```

- 'log record date and time' is the date and time of the recorded log.
- THREAD ('process Id', 'thread handle') is the information of log process id and thread handle.
- 'log prefix' is the entity or feature which created the log, and 'log body' is the detailed information.

Monitoring Performance Using View

Concurrency control for multi-user is needed because database is accessed or updated by multiple users a t the same time. The concurrency should be provided for system data and shared resources as well as exp licit data by SQL statements. GOLDILOCKS controls the concurrency using latch.

The concurrency control using lock can cause a deadlock when same data is updated by multiple differen t transactions. The concurrency control using latch (supported by GOLDILOCKS) can cause a deadlock, to o. It is because the deadlock affects the performance, so a view is provided to handle the latch when a de adlock occurs.

The transactions generating the deadlock will be be found when using V$LOCK_WAIT. Then, the adminis trator should monitor them, and unlock the deadlock. For more information about V$LOCK_WAIT, refer t o **V$LOCK_WAIT**. For more information for monitoring latch item causing the deadlock, refer to **V$LATC H**.

6.

Structure and Storage Structure of GOLDILOCKS Database

6.1 Managing Control File

A database instance should be created to use GOLDILOCKS database, and the control file is created when creating a database instance. The information is written on the control file when GOLDILOCKS multilevel startup moves from nomount level to mount level. The absolute path and the size of the file to be used in database are recognized by using the recorded information on the control file. The control file is a binary file and database information is stored as follows.

Control File Contents

Table 6-1 GOLDILOCKS database system information

Item	Description
Data store mode	It stores mode which is set when database starts up. (TDS, CDS)
Server state	It is the database instance state. (NONE, RECOVERED, RECOVERING, SERVICE, SHUTDOWN)
Last checkpoint lsn	It is the LSN of the checkpoint which was executed last in the database.

Table 6-2 Log information

Item	Description
Checkpoint lid, lsn	It is the log information (LSN, log position) of the checkpoint which was executed last in the database.
Last inactivated log file sequence	It is the log file sequence which was changed to inactive last.
Archivelog mode	It is the archivelog mode of the database in operation.
Creation time	It is the database created time.

Database information stores information about database operating, all tablespace in use, and data. The database operating information stored in the control file is as follows.

Table 6-3 Database information

Item	Description
Transaction table size	It is the maximum number of using transaction table in database.
Undo relation count	It is the number of using undo relations in database.
Tablespace count	It is the number of tablespaces in use created in database.
New tablespace id	It is the tablespace ID to be created later.

Tablespace information is stored in the control file as follows.

Table 6-4 Tablespace information

Item	Description
Tablespace id	It is a unique tablespace ID.
Attributes	It is the attribute of a tablespace such as storage device (memory, disk), persistence at tribute (temporary, persistent), tablespace usage (dictionary, undo, data, temporary).
Page count in extent	It is the number of extent pages.
State	It is the tablespace status. (CREATING, CREATED, DROPPING, DROPPED, AGING)
Relation id	It is the relation ID to store pending operation of tablespace.
New data file id	It is the data file ID which is set when new data file is inserted to tablespace.
Is logging	It is the logging mode of a tablespace. (LOGGING, NOLOGGING)
Is online	It is the online or offline status of a tablespace. (ONLINE, OFFLINE)
Data file count	It is the number of data files used by a tablespace.
Offline lsn	It is the last LSN which executes recovery if needed, in order to shift offline tablespace to online.
Offline state	It is the status of offline tablespace. (CONSISTENT, INCONSISTENT) CONSISTENT offline tablespace does not have to be recovered when shifting the statu s to online. It is because the tablespace status is shifted to offline after the most recen t data in memory is stored to disk. On the other hand, INCONSISTENT offline tablespa ce executes recovery using the log when shifting to online.

Data file information is stored in the control file as follows.

Table 6-5 Data file information

Item	Description
Name	It is the data file name including the absolute path which stores data.
State	It is the data file status. (CREATING, CREATED, DROPPING, DROPPED, AGING)
Data file id	It is the unique data file ID in a tablespace.
Auto extend	It is whether to extended automatically or not when a data file is full.
Size	It is the data file size.
Next size	It is the size to be extend when a data file is full.
Max size	It is the maximum size of extendable data file.
Timestamp	It is the time when data file was created.
Checkpoint lsn, lid	It is the checkpoint log information when the last checkpoint is executed in the data file. (LS N, log position)
Creation lsn, lid	It is the checkpoint log information of when a data file is created. (LSN, log position)

It stores each incremental backup information operated in the database. GOLDILOCKS database supports the incremental backup for database and tablespaces, and the incremental backup information is stored i n the control file as follows.

Table 6-6 Incremental backup information

Item	Description
Backup lsn, lid	It is the checkpoint log information which was executed last at the beginning of back up. (LSN, log position)
Begin time	It is the incremental backup beginning time.
Completion time	It is the incremental backup completion time.
Tablespace id	It is the unique tablespace ID of which an incremental backup is executed. If incremental backup for a tablespace is executed, its tablespace ID is recorded. If the backup is not executed for a tablespace, then the maximum tablespace ID (65535) is recorded.
Level	It is the level of executed incremental backup.
Object type	It is the target of incremental backup. (database, control file, tablespace)
Backup file name	It is the name of the incremental backup file.
Backup option	It is the incremental backup option. (cumulative, differential)

Multiplexing Control File

The control file stores the physical structure of GOLDILOCKS database and the information on data consistency. If the control file is corrupted or deleted by mistake, it is not possible to operate database.

Therefore, GOLDILOCKS database recommends creating at least two or more control files and keep them in physically separated disks. GOLDILOCKS supports multiplexing up to 8. To add control files when creating a database, set the number of multiplexing control files in the property file, and set the path of each control file. To add control files when operating database, increase the property value for control file multiplexing, and add the control file paths.

Restoring Corrupted Control File

If a control file is corrupted due to database's abnormal termination, the corrupted control file is restored using a normal control file among multiplexed control files, and then it is restarted.

If all multiplexed control files are corrupted, the backup control file is restored, and the incomplete media recovery is executed for archive redo log files and redo log files, and then it is restarted. For more information about incomplete recovery using the backup control file, refer to **When all multiplexed control files are corrupted** in recovery examples.

Control File Information

A user can enquire the control file name by using V$CONTROLFILE which is the performance view to retri eve the location and the name of the control file as follows.

```
gSQL> SELECT CONTROLFILE_NAME FROM V$CONTROLFILE;
CONTROLFILE_NAME
------------------------------------------------------
/goldilocks_data/wal/control_0.ctl
/goldilocks_data/wal/control_1.ctl
2 rows selected.
```

A user can retrieve the correct information written in the control file using **gdump** the dump tool of GOL DILOCKS.

6.2 Managing Redo Log File

GOLDILOCKS uses the redo log files to ensure database persistency. In other words, the database can be restored to the state before shutdown database using the redo log files and data files when GOLDILOCKS database is abnormally terminated due to various reasons.

In order to do so, GOLDILOCKS database uses Write Ahead Logging (WAL) policy to store log of all updat e operations.

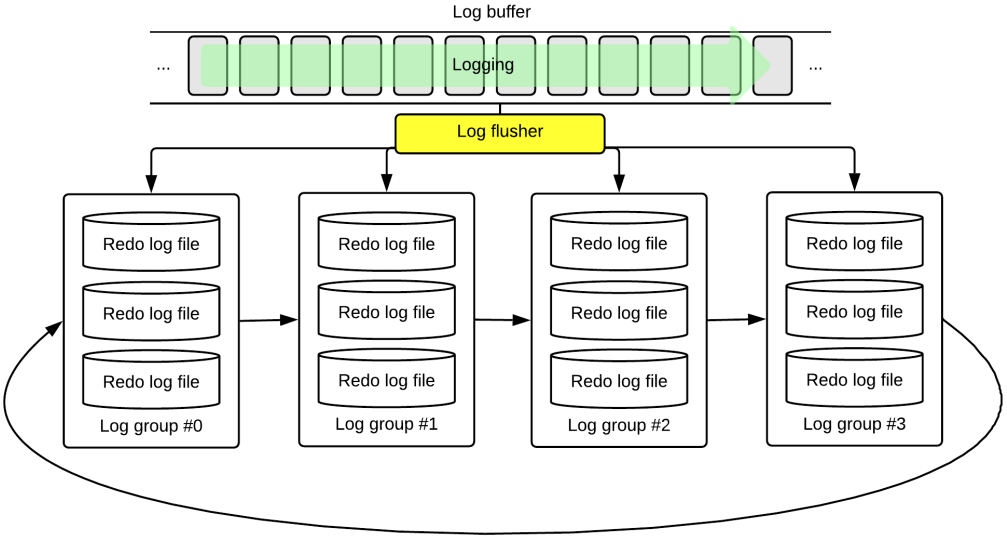
The updated data by the update operation is not recorded in the data file, but the log for the update ope ration is recorded in the redo log file. It is because it is much more efficient in the terms of database perfo rmance. Whenever each update operation is recorded in the data file, a random access occurs and the up date operations for the same file causes the excessive disk IO. However, the update log is smaller than th e updated data, and it performs the efficient disk IO in a way being appended at the end of log file.

GOLDILOCKS database uses log buffer in shared memory to record the updated logs to the log buffer, an d then records log buffer to the log file all together, and it leads to more efficient disk IO.

Redo Log File Structure

The redo log buffer and log file in GOLDILOCKS have a circular structure. The log files are created as man y as predefined number of log groups, and then they record logs. If a log file is full, the following log file i s used. When all log files are used up, the previously used log file is reused. The log file is a circular struct ure consisting of a single log group of several members. GOLDILOCKS performs the logging using minim um four log groups.

Figure 1 GOLDILOCKS redo log file, log buffer structure



Redo Log Group and Its Member

GOLDILOCKS database uses a log group and members to record logs to the disk log file during database operation. A single redo log file is a member of a log group, and a log group consists of several log members. Log group which consists of many members ensures high availability because other log members can be used when a particular disk fails or a particular log member is corrupted.

The system records logs to a log group, and if the log group is full, the following group is used. This is a circular log group. The currently used log group shifts to the following log group one and this is called as log switching.

A log group, the number of members and each of its position are set by the property when creating database. They can be altered by adding or dropping syntax when operating database.

Log Group State

The state of log group is initialized to UNUSED when creating it. It is changed to CURRENT, ACTIVE, and INACTIVE state by the system during operation.

Table 6-7 GOLDILOCKS log group state

Log group state	Description
UNUSED	The log group has never been used after creation.
CURRENT	The log group is being used by the current system.
ACTIVE	It is not yet ready to be reused after CURRENT log group is switched.
INACTIVE	ACTIVE state log group is ready to be reused.

ACTIVE log group can not be reused in the system, but it can be reused after it is changed to INACTIVE by archive log thread. Archive log thread wakes up by an event before the checkpoint is completed, and it changes ACTIVE log groups to INACTIVE state. In order to do so, log archiving thread archives log file in ACTIVE log group, then change it to INACTIVE when the system is operated in ARCHIVELOG mode. If the system is operated in NOARCHIVELOG mode, then the ACTIVE log group is immediately reusable after changing the log group state to INACTIVE.

Adding Log Group and Log Member

Adding Log Group

Adding log group is allowed only in mount phase of GOLDILOCKS database multilevel startup. The new log group is added next to the CURRENT state of the log group which is currently being used.
The following is an example to add a new log group whose file name is 'abc.log', and file size is 20 Mbytes to group ID 10.

```
ALTER DATABASE ADD LOGFILE GROUP 10 ('abc.log') SIZE 20M;
```

Adding Log Member

A new log member can be added for the stability of log group in use. It can be added in mount phase of GOLDILOCKS database multilevel startup in the same way of adding log group. The following is an exam ple to add the log file named as 'test log' to group ID 10. The log file size is not specified because the log member size in a log group is same.

```
ALTER DATABASE ADD LOGFILE MEMBER 'test.log' TO GROUP 10;
```

Altering Log Member Name

The position and the file name of the log member in use can be altered by executing RENAME in mount phase. RENAME is executed for log member when the log member's disk is physically failed or the log me mber should be transferred to another disk in terms of performance.
The following is an example to RENAME the log file '/disk1/goldilocks_data/wal/redo_0_0.log' to the log file '/disk2/goldilocks_data/wal/redo_0_0.log'.

```
ALTER DATABASE RENAME LOGFILE '/disk1/GOLDILOCKS_data/wal/redo_0_0.log' TO
'/disk2/GOLDILOCKS_data/wal/redo_0_0.log';
```

Dropping Log Group or Log Member

A log member or log group in use can be dropped in mount phase when a user wants to reduce log grou ps, log members or when log file disk is failed.
The following is an example to drop all members of log group ID 10.

```
ALTER DATABASE DROP LOGFILE GROUP 10;
```

Dropping log member is allowed only when at least two members exist in the log group.
The following is an example to drop the '/disk1/goldilocks_data/wal/redo_0_0.log' log member.

```
ALTER DATABASE DROP LOGFILE MEMBER '/disk1/GOLDILOCKS_data/wal/redo_0_0.log';
```

Restoring Corrupted Redo Log File

If the system fails and the redo log files are corrupted, then the restart recovery fails and the system can n ot be operated. If normal log members exist in the corrupted log group, the restart recovery and the syste

m operation can be executed by copying the log file of the normal log member to the corrupted log files.

If all log files in a log group are corrupted or there is only one log member, it can be restarted by executin g incomplete media recovery. The incomplete media recovery is restricted to the normal log file.
For more information about incomplete media recovery, refer to **Incomplete Recovery**.

Redo Log File Information

A user can enquire V$LOGFILE which is a performance view to retrieve the location and the name of the r edo log files. When inquiring V$LOGFILE, the log file name, its log group ID, its state, the file size are retri eved as follows.

```
gSQL> SELECT FILE_NAME, GROUP_ID, GROUP_STATE, FILE_SIZE FROM V$LOGFILE;
FILE_NAME                          GROUP_ID GROUP_STATE FILE_SIZE
---------------------------------- -------- ----------- ---------
/disk1/goldilocks_data/wal/redo_0_0.log        0 INACTIVE    104857600
/disk1/goldilocks_data/wal/redo_1_0.log        1 CURRENT     104857600
/disk1/goldilocks_data/wal/redo_2_0.log        2 UNUSED      104857600
/disk1/goldilocks_data/wal/redo_3_0.log        3 UNUSED      104857600
4 rows selected.
```

6.3 Managing Archive Redo Log File

The database should be operated in archive log mode for media recovery using backup, because GOLDILOCKS redo log file reuses log group which is used when *circular* structure log group is used. Then the written completed redo log files are copied to the archive redo log files.
For more information about GOLDILOCKS database's archive log mode, refer to **ARCHIVELOG Mode**.

Creating Archive Redo Log File

The redo log file is copied to archive redo log file directory by log archiving thread which is GOLDILOCKS database's system thread. Log archiving thread is activated by checkpoint thread during checkpoint, and it archives the proper object among redo log files.

Archive redo log file is created in the directory set by the ARCHIVELOG_DIR_ 1 property. The name of archive redo log file is composed of its prefix set by the ARCHIVELOG_FILE property and of the sequence number of each redo log file.

Maintaining and Dropping Archive Redo Log File

Media recovery using backup can fail if archive redo log files, similarly to the redo log files, are arbitrarily dropped. Therefore, archive redo log files should be saved with backup files, and the archive redo log files for media backup can be dropped when the backup file is not needed any more.

Backup is copying the data file which is downloaded to the disk by the most recent checkpoint. Therefore, the archive log files of the oldest LSN and later are needed to recover media using backup. Checkpoint is executed by the system when redo log file is switched, so one or more checkpoint logs exist in every log file except for the redo file in CURRENT state.

The followings describe how to get the archive redo log file required for media recovery using backup file.

1. Get the checkpoint LSN which is recorded in the file header of backup data files.
2. Dump the archive redo log file, and get the archive redo log file including checkpoint LSN.
3. The archive redo log file needed for the media recovery using backup begins just before the archive redo log file of No. 2.

Dump the control file and get the checkpoint LSN to get the archive redo log file needed for incremental backup. Subsequent process is as same as the entire backup process.

If backup files are not needed any more, the archive redo log file required for the media recovery using ba

ckup can be removed.

Multiplexing Archive Redo Log File Directory

If the archive redo log file is moved from the directory set in ARCHIVELOG_DIR_1 to the other directory o r media to preserve the archive redo log file, the media recovery fails because it can not find the required archive redo log file.

Move the archive redo log file back to the directory set in ARCHIVELOG_DIR_1 to resolve this problem. Or set the archive log files existing directory to ARCHIVELOG_DIR_2 ~ ARCHIVELOG_DIR_10, and add archiv e redo file directory for media recovery. READABLE_ARCHIVELOG_DIR_COUNT should be set as many as the number of directory when using ARCHIVELOG_DIR_2 ~ ARCHIVELOG_DIR_10 for the media recovery.

6.4 Managing Tablespace

All the data used in database are stored in physical disk file, and use the database's logical structure for ef ficient data management and performance. GOLDILOCKS manages the disk space efficiently by using the logical structure such as tablespace, segment, extent, and page.

A tablespace can include multiple data files, and each tablespace is set to online/offline to support high d ata availability. Distributing data storing disks improves the IO performance, and reduces the contention of physical disk IO.

Tablespace Type

GOLDILOCKS tablespace is divided into SYSTEM tablespace and non-SYSTEM tablespace. SYSTEM tables pace is created when database is created, and it is used and controlled only by the GOLDILOCKS system. Non-SYSTEM tablespace is created and used by a user.

SYSTEM Tablespace

It is created when GOLDILOCKS database is created, and it is essential for database operation. There are dictionary tablespace, undo tablespace and system temporary tablespace.

Non-SYSTEM Tablespace

It is a tablespace in which tables and indexes are stored for data storing, and a user can arbitrarily creates or drops.

Managing Tablespace and Data File

Managing Tablespace

Managing Tablespace State

The GOLDILOCKS database tablespace state is divided into online and offline. The offline tablespace is no t accessible. A user can arbitrarily set the tablespace state to offline. Or, an abnormal tablespace can be s et to offline by the system. The system tablespace should not be set to offline.

- Offline tablespace

GOLDILOCKS database flushes all data files in the tablespace to disk before the tablespace is set to offline. All related logs should be flushed to disk to flush data file. Therefore, a recovery is not needed when swit ching to online later. However, it is applied when DDLs occurred in the offline tablespace state is changed to online state.

A tablespace can be immediately set to offline using IMMEDIATE mode, without flushing data file. In this case, the tablespace state is changed to online after media recovery when setting the tablespace to onlin e.

Table 6-8 Tablespace offline option

Option	Description	Media recovery when setting the tablespace to online
NORMAL	Flushing all the data file related logs in tablespace to the disk, then set it to offline	Media recovery is not required.
IMMEDIATE	Immediately setting a tablespace to offline	Media recovery is required.

GOLDILOCKS can set tablespace to offline in mount phase. To do so, the service should be normally termi nated, or the server should be operated in ARCHIVELOG mode. This method provides high availability by performing service exclusing the unrecoverable tablespaces when starts up database.

```
gSQL> ALTER TABLESPACE TEST_TBS OFFLINE;

Tablespace altered.

gSQL> ALTER TABLESPACE TEST_TBS ONLINE;

Tablespace altered.
```

```
gSQL> ALTER TABLESPACE TEST_TBS OFFLINE IMMEDIATE;

Tablespace altered.

gSQL> ALTER TABLESPACE TEST_TBS ONLINE;

ERR-42000(14051): media recovery required - 'TEST_TBS'

gSQL> ALTER DATABASE RECOVER TABLESPACE TEST_TBS;

Database altered.

gSQL> ALTER TABLESPACE TEST_TBS ONLINE;
```

```
Tablespace altered.
```

Attributes of Tablespace

The followings are attributes of tablespace in GOLDILOCKS database. PERSISTENT or TEMPORARY indicates whether persistence is ensured or not. DATA or UNDO indicates the the type of tablespace and stored data.

Table 6-9 Tablespace attribute of GOLDILOCKS database

Attribute		Description
Persistence	PERSISTENT	It supports persistence of data stored in a tablespace. (A recovery is required.)
	TEMPORARY	It does not support persistence of data stored in a tablespace.
Type of stored data	DATA	It stores the user input data.
	UNDO	It stores data required for MVCC of database.
	DICT	It stores dictionary information for database operation.
	TEMPORARY	It stores data for SQL processing.
Type of stored media	DISK	Pages of the disk table space should be read from the disk data file by using the buffer cache. If it is cached, then it is accessible in the buffer cache.
	MEMORY	When creating a tablespace, the exclusive shared memory whose size is as same as that of the data file is created. Therefore, the instant access to the desired page is available in the memory.

Managing Tablespace

- Creating user tablespace

It creates a new user tablespace. The name of tablespace being used in the database instance should be unique when creating tablespace. A tablespace can have up to 1024 data files. The data file in the memory tablespace can store up to 30 GBytes, and the data file in the disk tablespace can store as much as the disk physically allows. The database can create tablespaces up to 65,535, including system tablespace.

The data file name including the absolute path in the data file should be unique. Use 'REUSE' option to reuse the existing data file which is not being used in database. An extent size of a tablespace is selectable among 64 Kbytes, 128 Kbytes, 256 Kbytes, 512 Kbytes and 1 Mbyte, and the default extent size is 256 Kbytes.

```
gSQL> CREATE TABLESPACE TEST_TBS DATAFILE
     '/goldilocks1/db/TEST_TBS1.dbf' SIZE 20M,
     '/goldilocks2/db/TEST_TBS2.dbf' SIZE 50M,
     '/goldilocks3/db/TEST_TBS3.dbf' SIZE 100M REUSE;
Tablespace created.
```

The following is an example of creating a disk tablespace.

```
gSQL> CREATE DISK TABLESPACE TEST_TBS DATAFILE
     '/goldilocks1/db/TEST_DISK_TBS1.dbf' AUTOEXTEND OFF MAXSIZE 20M,
     '/goldilocks2/db/TEST_DISK_TBS2.dbf' AUTOEXTEND ON NEXT 20M MAXSIZE UNLIMITED REUSE;
Tablespace created.
```

The state can be set to ONLINE/OFFLINE when creating tablespace, and the LOGGING/NOLOGGING prop erty also can be set.

- Dropping tablespace

The tablespace and data file can be dropped when the tablespace is not needed any more. The unused ta blespace should be dropped not to waste the resources. It is because once tablespace is created, the add ed disk data file and memory is created and remains.

```
gSQL> DROP TABLESPACE TEST_TBS;
Tablespace dropped.
```

Dropping the tablespace does not drop its table index in use by default. Therefore, *INCLUDING CONTENT S* option should be used together when dropping tablespace including tables or indexes in use.

```
gSQL> DROP TABLESPACE TEST_TBS;
ERR-42000(16148): tablespace not empty, use INCLUDING CONTENTS option :
drop tablespace TEST_TBS
                *
ERROR at line 1:
gSQL> DROP TABLESPACE TEST_TBS INCLUDING CONTENTS;
Tablespace dropped.
```

Use *AND DATAFILES* option to drop the data files added in the tablespace.

```
gSQL> DROP TABLESPACE TEST_TBS INCLUDING CONTENTS AND DATAFILES;
Tablespace dropped.
```

- Altering tablespace size

Add datafiles to the tablespace or drop datafiles from the tablespace to alter the tablespace size. Add ne w datafiles to tablespace to spare space when the storing space is not enough during database operation. Or, drop unused datafile from tablespace not to waste the space.

```
gSQL> ALTER TABLESPACE TEST_TBS ADD DATAFILE 'TEST_TBS2.dbf' SIZE 20M;
Tablespace altered.
gSQL> ALTER TABLESPACE TEST_TBS DROP DATAFILE 'TEST_TBS2.dbf;
Tablespace altered.
```

The data file could be dropped from tablespace only when is has never been used since its creation. The d ata file can not be dropped once it has been used even when all data was dropped.

```
ALTER TABLESPACE TEST_TBS DROP DATAFILE 'TEST_TBS2.dbf';
ERR-42000(14044): datafile not empty
```

- Managing temporary tablespace

Temporary tablespace does not store data file, but it allocates memory of specified size. Create the tempo rary tablespace as follows.

```
gSQL> CREATE TEMPORARY TABLESPACE TEST_TBS MEMORY 'TEST_TEMP_TBS' SIZE 10M EXTSIZE 256K;
Tablespace created.
```

Add memory to the temporary tablespace as follows.

```
gSQL> ALTER TABLESPACE TEST_TBS ADD MEMORY 'TEST_TBS2' SIZE 10M;
Tablespace altered.
```

Drop the unused memory from the temporary tablespace as follows.

```
gSQL> ALTER TABLESPACE TEST_TBS DROP MEMORY 'TEST_TBS2';
Tablespace altered.
```

Drop the temporary tablespace as follows.

```
gSQL> DROP TABLESPACE TEST_TBS INCLUDING CONTENTS AND DATAFILES;
Tablespace dropped.
```

Transferring Data File

The data file storage path stored in the database should be modified when altering the datafile storage di sk, or directory.

The following is an example to describe how to modify the path when transferring the datafile '/goldilock s1/db/TEST_TBS1.dbf' in tablespace TEST_TBS to '/goldilocks4/db/TEST_TBS1.dbf'.

```
gSQL> ALTER TABLESPACE TEST_TBS RENAME DATAFILE
    '/goldilocks1/db/TEST_TBS1.dbf' TO '/goldilocks4/db/TEST_TBS1.dbf';
Tablespace altered.
```

Tablespace Information

For more information about the tablespaces created in database, refer to **V$TABLESPACE**.

```
gSQL> \DESC V$TABLESPACE
COLUMN_NAME   TYPE                   IS_NULLABLE
------------- ---------------------- -----------
TBS_NAME      CHARACTER VARYING(128) FALSE
TBS_ID        NUMBER                 FALSE
TBS_ATTR      CHARACTER VARYING(128) FALSE
IS_LOGGING    BOOLEAN                FALSE
IS_ONLINE     BOOLEAN                FALSE
OFFLINE_STATE CHARACTER VARYING(32)  FALSE
EXTENT_SIZE   NUMBER                 FALSE
PAGE_SIZE     NUMBER                 FALSE
```

6.5 Managing Data File

Data File Matching

Data file can be corrupted due to disk failure, database defects or human error. If the corrupted data file i s used in database, serious problems occurs.

GOLDILOCKS database ensures the matching of the data file using a checksum for each page of the data file. GOLDILOCKS database's page checksum is generated using LSN and CRC value, and it is stored in ea ch page. A user sets the page checksum type using the value in **PAGE_CHECKSUM_TYPE**, and uses LSN f or the default value.

The page checksum is checked when loading the data file into the memory at database startup. If an erro r occurs concerning the checksum value, the database can not start up the service.

The following is an example to describe the database startup failure when the datafile TEST_TBS.dbf in th e tablespace TEST_TBS created by a user is not matching.

```
gSQL> \STARTUP MOUNT
Startup success
gSQL> ALTER SYSTEM OPEN DATABASE;
ERR-HY000(14094): datafile recovery required - datafile(/goldilocks/db/TEST_TBS.dbf) of
tablespace(TEST_TBS) corrupted
```

In case when the data file is not matching, set its tablespace having the data file to *OFFLINE*, or recover th e data file to restart the database.

The following describe how to start up the database after setting the tablespace to *OFFLINE*.

```
gSQL> \STARTUP MOUNT
Startup success
gSQL> ALTER SYSTEM OPEN DATABASE;
ERR-HY000(14094): datafile recovery required - datafile(/goldilocks/db/TEST_TBS.dbf) of
tablespace(TEST_TBS) corrupted
gSQL> ALTER TABLESPACE TEST_TBS OFFLINE IMMEDIATE;
Tablespace altered.
gSQL> ALTER SYSTEM OPEN DATABASE;
System altered.
```

Data file full backup or incremental backup should exist when recovering the data file.
The following describes how to recover the data file when backup exists.

```
gSQL> \STARTUP MOUNT
Startup success
gSQL> ALTER SYSTEM OPEN DATABASE;
ERR-HY000(14094): datafile recovery required - datafile(/goldilocks/db/TEST_TBS.dbf) of
tablespace(TEST_TBS) corrupted
gSQL> ALTER DATABASE RECOVER DATAFILE 'TEST_TBS.dbf' CORRUPTION;
Database altered.
gSQL> ALTER SYSTEM OPEN DATABASE;
System altered.
```

Data File Information

For more information about data files being used in database, refer to **V$DATAFILE**.

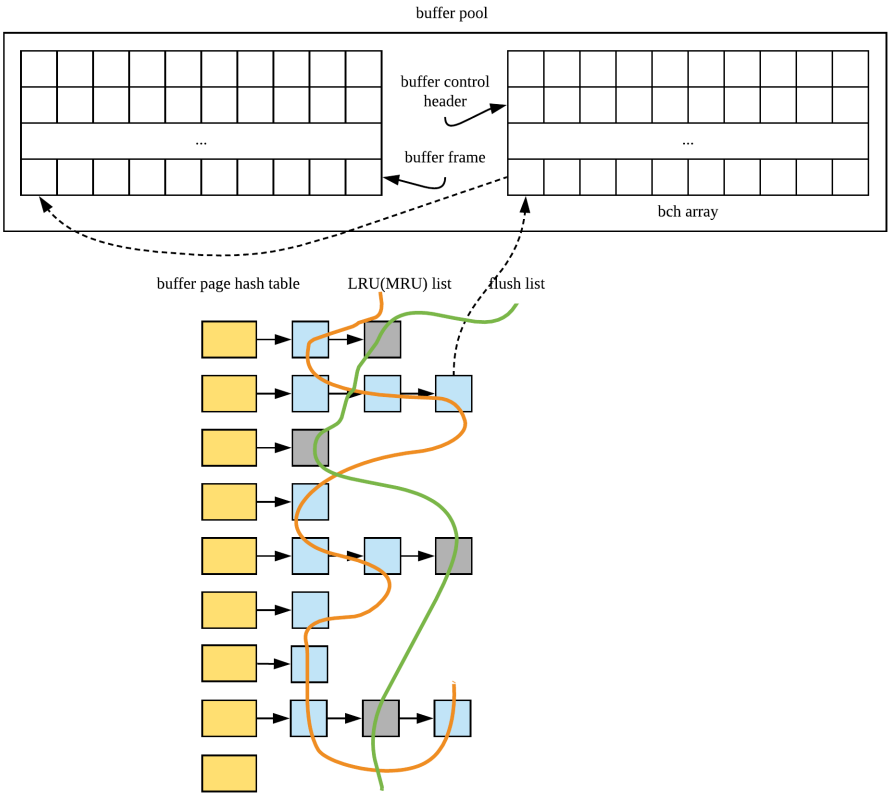
```
gSQL> \DESC V$DATAFILE
COLUMN_NAME    TYPE                           IS_NULLABLE
-------------- ------------------------------ -----------
TBS_NAME       CHARACTER VARYING(128)         FALSE
DATAFILE_NAME  CHARACTER VARYING(1024)        FALSE
CHECKPOINT_LSN NUMBER                         FALSE
CREATION_TIME  TIMESTAMP(6) WITHOUT TIME ZONE FALSE
FILE_SIZE      NUMBER                         FALSE
```

6.6 Buffer Cache

Structure of GOLDILOCKS Buffer Cache

It caches pages required from the disk data file to access index pages and tables stored in the disk table s pace to the buffer. GOLDILOCKS allocates the buffer cache as big as the size set in **BUFFER_CACHE_SIZE** property, and it retrieves the pages requested from the page cached in the buffer cache by using the has hash table as big as the size set in **BUFFER_HASH_BUCKETS** property. It increases the touch count whene ver accessing to the page. If the available space does not exist in the buffer cache, then it uses LRU strate gy which replaces the page whose touch count value is small.

Figure 2 Structure of GOLDILOCKS buffer cache



Buffer Cache List

The followings are lists which are used for the buffer cache in GOLDILOCKS.

Table 6-10 Buffer cache list

List	Description	Property
Buffer free list	It is a list of buffer frames which are immediately a vailable.	BUFFER_FREE_LIST_COUNT
Buffer LRU list	It is a list to retrieve the pages which is reusable ba sed on the touch count.	BUFFER_LRU_LIST_COUNT
Buffer flush list	It is a list of buffer frames to reflect the pages upd ated to reuse in the buffer LRU list to the data file.	BUFFER_FLUSH_THREADS
Buffer checkpoi nt list	It is a list of updated page frames which should be flushed to the data file at the time of checkpoint.	BUFFER_CHECKPOINT_LIST_COUNT

Page frames which were not used after the system startup are linked to the buffer free list, and used pag e frames are linked to the buffer LRU list. The page frame updated in the buffer cache is linked to the buf fer checkpoint list, but it is not dropped from the buffer LRU list.

After all page frames are used in the buffer free list and a page frame to cache a new fage does not exist, then it looks for an available page frame by retrieving the buffer LRU list. The page frame which were up dated but not in use is transferred to the buffer flush list at this moment.

A page frame can not be linked dundantly to the buffer free/ LRU/ flush list except for the buffer checkpo int list. In other words, a page frame in the buffer free list can not simultaneously exist in the buffer LRU/ flush list, nor can a page frame in the buffer LRU list exist in the buffer free/ flush list.

The buffer LRU list is divided into hot/ cold area, and if the touch count of the page frame in the cold are a is same or larger than the specified value (**BUFFER_HOT_REGION_CRITERIA**), then it is transferred to th e hot area. The touch count increases whenever the page frames in the buffer LRU list access, and if all of them belong to the hot area, then the page frame to replace is not available. Therefore, the hot area size is restricted to the **BUFFER_HOT_REGION_PERCENT** property value.

Page Frame Status

All page frames of the buffer cache have the following status.

Table 6-11 Buffer frame status

Status	Description	Page frame access
FREE	It is the page frame which is not currently used.	Inaccessible
PREPARED	It is allocated for the page caching, but is not completely read from t he disk.	Inaccessible
CLEAN	A page was cached to the buffer cache and the page frame never h as been updated.	Accessible
DIRTY	A page was cached to the buffer cache and the page frame is updat	Accessible

Status	Description	Page frame access
	ed.	
FLUSHING	A page was cached to the buffer cache and the page frame is updated., and it is being flushed to the disk data file.	Accessible
INCONSISTENT	The page frame status is abnormal in the buffer cache.	Inaccessible

7.

Backup and Recovery of GOLDILOCKS Database

This chapter describes GOLDILOCKS database backup and recovery, database ARCHIVELOG mode for the backup and recovery.

7.1 ARCHIVELOG Mode

GOLDILOCKS database executes logging using circular log group. A circular log group consists of at least four log groups, and if all log allocated to a log group are run out, then following log group is used. If all created log groups are run out, the first log group is reused. In this case, the new log file is not created b ut the previously recorded log file is reused.

The previously written logs are lost if a log group is reused in NOARCHIVELOG mode. Therefore, if an ad ministrator does not manage the completed logging group, the completed transactions logs disappears while operating in NOARCHIVELOG mode.

On the other hand, in ARCHIVELOG mode, the system archives log files prior to reusing it when using the following log groups after the completion of recording on log file in a log group. Then the completed log s are permanently preserved unless they are deliberately deleted.

ACHIVELOG Mode

Log files should be archived before they are reused. It is because all log files after backup moment is requi red for the recovery using backup, and the backup could be needed anytime. Therefore, the backup is su pported only in ARCHIVELOG mode.

The service can be interrupted in busy system due to archiving when operating in ARCHIVELOG mode. Al so, an additional space is needed to store files.

NOARCHIVELOG Mode

Backup is not supported in NOARCHIVELOG mode, because it can not be determined if the log files of be fore reusing exist.

However, system does not archive log files. Therefore, the interruption due to an archiving at checkpoint when bulk logs are continuously being recorded does not occur. Also, it does not need a storage space fo r archive log files.

NOARCHIIVELOG mode is set by the 'ARCHIVELOG_MODE' property value when creating database. Data

base is created in NOARCHIIVELOG mode if the 'ARCHIVELOG_MODE' value is 0, and it is created in ARC HIIVELOG mode when the 'ARCHIVELOG_MODE' value is 1. This property is valid only when creating dat abase, and it is not referenced when operating database.

Execute the following syntax in mount phase of GOLDILOCKS database startup level to change ARCHIVEL OG mode during database operation.

```
gSQL> ALTER DATABASE ARCHIVELOG;

Database altered.

gSQL> ALTER DATABASE NOARCHIVELOG;

Database altered.
```

Enquire *ARCHIVELOG_MODE* of *V$ARCHIVELOG* which is the performance view to retrieve archive log m ode set in database.

```
gSQL> SELECT ARCHIVELOG_MODE FROM V$ARCHIVELOG;
ARCHIVELOG_MODE
---------------
NOARCHIVELOG
1 row selected.
```

7.2 Backup and Recovery

Backup

The Purpose of Backup and Recovery

Database can protect and recover data when various failures or data loss occurs. There are many reasons for failures. The duplicated copy is required especially when the database is physically corrupted or dama ged by disaster, and this is called as backup.

When database service is not available due to various failures, it becomes available again by using current database or backup, and this is called as recovery. The restart recovery is to recover by using the current d atabase. The media recovery is to recover by using the backup. GOLDILOCKS automatically or manually p erforms the media recovery to recover the backup data file, then recover and restarts by the restart recov ery.

Backup

Database backup is divided into physical backup and logical backup. Generally, backup means making co py of data files online. This chapter describes the online physical backup.

Table 7-1 Database backup type

Backup type	Backup form	Database state	Description
Physical backup	Cold backup	Offline	Creating the copy of data file Stopping service to execute backup
	Hot backup	Online	Creating the copy of data file Executing backup during service operation Available only in ARCHIVELOG mode
Logical backup	Export backup	Online	Backup/recovery in table unit Exporting regardless of HW/OS

GOLDILOCKS uses data files and control files for executing service, and they should be recovered when th ey were corrupted from a failure. A control file is created when creating database, and stores necessary in formation to operate database. A data file stores actual data, and they are data files in system tablespace s created when database is created and datafiles in user made tablespace. Use backup files for recovery w hen some of those control files or data files are corrupted.

In other words, control files and data files should be backed up for recovery. In order to do so, GOLDILOC KS supports control file backup, database backup, and tablespace backup.

Depending on the backup method, it is divided into full backups and incremental backups. Full backup co

pies data files in time of backup, and an incremental backup copies only modified parts since the previous backup. The full backup copies data files, so the copy as big as the data file is created everytime of backup. Therefore, it consumes the storage space as much as the size of database or tablespace.

On the other hand, the size of the incremental backup is relatively small because it copies only the modified part after the previous backup.

Table 7-2 Full backup vs. incremental backup

Item	Full backup	Incremental backup
Backup object	• Database: Entire data file which is being used by database • Tablespace: Data files in database's specific tablespace	
Description	• Backup entire data file which is being used by database or tablespace • Creating a backup file per a data file • Restoring the required data file after failure by using appropriate backup method, then recover it	• Backup the modified part of the data file being used by database or tablespace after the previous backup • Creating an incremental backup file in which the modified part is recorded • Recovery by using multiple incremental backups after failure

Full Backup

Use full backup to execute database backup and tablespace backup. Database backup is to backup control files and data files.

Control File Backup

Backup the control file as follows. Specify the backup control file name including the absolute path, or specify the backup control file name only. If only the backup control file name is specified, the backup file is created in the path set in the 'LOG_DIR' property.

```
gSQL> ALTER DATABASE BACKUP CONTROLFILE TO '/goldilocks_data/backup/backup.ctl';
Database altered.
```

Database Backup

Database backup can backup entire data file being used in database. When backup data file, recording on the file should be prevented while coping the data file. If the file is used during copying, the data file becomes inconsistent, and even worse it will be inconsistent within a page. Set the database to the state which enables backup to prevent those inconsistencies.

```
gSQL> ALTER DATABASE BEGIN BACKUP;
Database altered.
```

Use operating system's file copy feature to create the copy of data file on database backup enabled state. Then set it as follows, then the database backup is completed, and it is writable.

```
gSQL> ALTER DATABASE END BACKUP;
Database altered.
```

Tablespace Backup

Tablespace backup can backup data files being used by a specified tablespace. Set it to backup enabled st ate by using the table space name (tablespace_name) as follows for the same reason of database backup.

```
gSQL> ALTER TABLESPACE TEST_TBS BEGIN BACKUP;
Tablespace altered.
```

Use operating system's file copy feature to create the copy of tablespace's data file on tablespace backup enabled state. Then, complete the tablespace backup as follows.

```
gSQL> ALTER TABLESPACE TEST_TBS END BACKUP;
Tablespace altered.
```

Incremental Backup

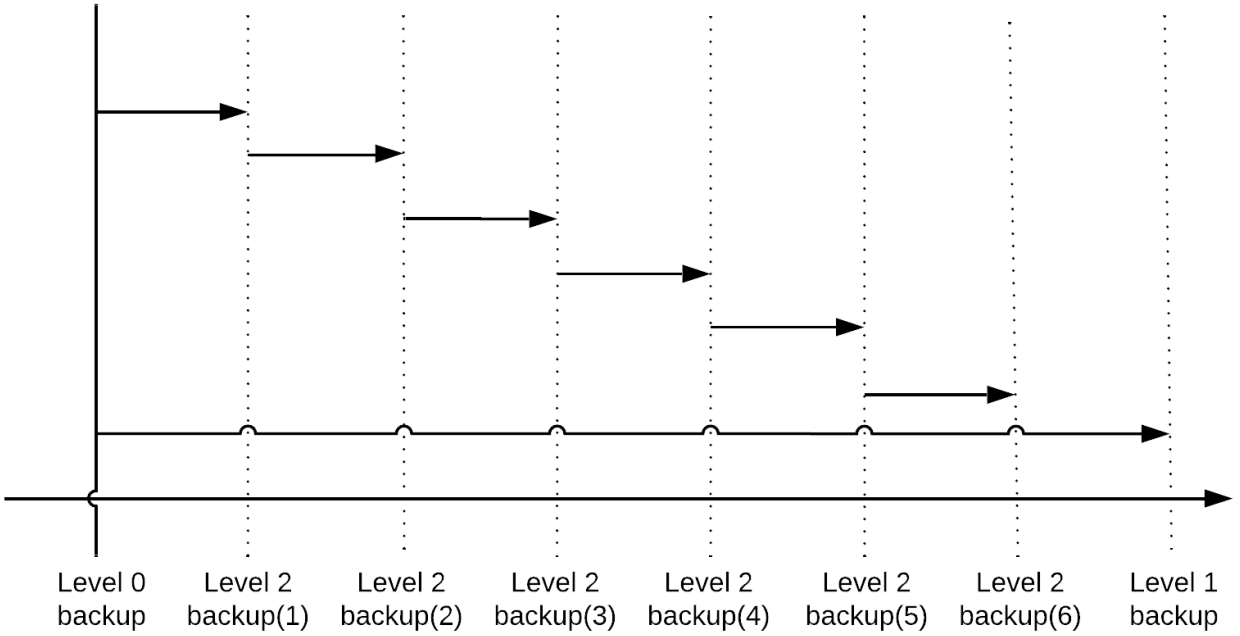
An incremental backup supports database unit backup and tablespace unit as same as full backup. An inc remental backup does not backup control files separately, but the control file is backed up together when executing the database incremental backup.

GOLDILOCKS supports incremental level from 0 to 4 for an incremental backup. When an incremental ba ckup is executed for the first time, the level should be set to 0, and backup the entire data file. Set the inc remental backup level to 1 or higher to back up only the modified parts since the last backup when execu ting the incremental backup later.

The given level of incremental backup searches for the time when the same level or lower level was execu ted. Then, it backups only the modified parts after the previous backup.

For example, after performing level 0 backup, level 2 backup(1) backs up only the modified parts after th e level 0 backup, and level 2 backup(2) backs up the modified part after the level 2(1) backup. In the sam e way, level 2 backup(3), (4), (5), (6) backs up the modified part after the level 2 backup. The level 1 back up which was executed lastly backs up all modified parts after the level 0 backup.

Figure 1 Incremental backup



Database Incremental Backup

Execute an incremental backup on the entire data file of database as follows. At first, the entire data file of database is backed up at level 0.

```
gSQL> ALTER DATABASE BACKUP INCREMENTAL LEVEL 0;
Database altered.
```

And then, the modified part after level 0 is backed up at level 1.

```
gSQL> ALTER DATABASE BACKUP INCREMENTAL LEVEL 1;
Database altered.
```

Tablespace Incremental Backup

At first, the entire data file of tablespace is backed up at level 0 as follows in the same way as the databas e backup.

```
gSQL> ALTER TABLESPACE TEST_TBS BACKUP INCREMENTAL LEVEL 0;
Tablespace altered.
```

And then, the modified part after level 0 is backed up at level 1.

```
gSQL> ALTER TABLESPACE TEST_TBS BACKUP INCREMENTAL LEVEL 1;
Tablespace altered.
```

Change Tracking

It should scan the entire data file to figure whether any page is updated after the previous backup to perform the incremental backup for the disk tablespace. Therefore, if the data file size is big, then the incremental backup takes a long time even when the number of actually updated pages is small because it should scan the entire data file.

Change tracking stores only updated pages after the incremental backup, then performs the next incremental backup by selecting updated pages only without full scan. Therefore, the backup time is decreased.

However, if most of the pages in the data file were updated, then it should backup the most part of the data file, so the the change tracking is not efficient. The change tracking is available when the database is in ARCHIVELOG mode only, but it is not available in NOARCHIVELOG mode.

The following is an example of enable/ disable the change tracking in the database.

```
gSQL> ALTER DATABASE ENABLE CHANGE TRACKING;
Database altered.
gSQL> ALTER DATABASE DISABLE CHANGE TRACKING;
Database altered.
```

If the change tracking is enabled, then it creates the tracking file and the shared memory. The structure of the change tracking file and that of the shared memory are same. It consists of blocks storing update flags for pages bound in a number specified in **CHANGE_TRACKING_EXTENT_SIZE** property.

The update flags are initialized when the incremental backup is performed for the first time after the change tracking is enabled. If a page is updated, then it is recorded on the flag of the page, so it scans only the pages which were recorded on the update flags in the change tracking file and backs up only updated pages at the next incremental backup.

If the change tracking is enabled, then it creates a file in the location specified in **CHANGE_TRACKING_FILE** property. If the file name and storing location for change tracking are specified when enabling it, then the file is created in the specified location. If the storing location is not specified, then the change tracking file is created in the location specified in BACKUP_DIR property.

```
gSQL> ALTER DATABASE ENABLE CHANGE TRACKING USING FILE '/tmp/change_tracking.ctf';
Database altered.
```

The change tracking file size is 10 M, and it increases by 10 M when the change tracking file is full due to the increased number of disk data files.

Recovery

Database guarantees data consistency by executing recovery when a failure occurs or database is corrupted.
The types of database failure are as follows.

Table 7-3 Database failure type

Failure type	Causes and symtoms	Solution
Transaction failure	Transaction failure and deadlock due to the logical error (bad input, overflow, data not found)	Abort transaction
System crash	Corruption of volatile storage device due to an abnormal termination (blackout) of DBMS or OS	Restart recovery
Media failure	Corruption of non-volatile storage device	Restore, restart recovery

Abort the executing transactions, rollback all database updates and release obtained lock items to solve transaction failure.

Database processes are abnormally terminated at system crash, so the recent information stored in volatile storage are not reflected in the non-volatile storage, and they are lost. Startup database, and recover the database to the state when it was consistent before abnormal termination. This process is called as restart recovery. To recover the database, the restart recovery uses control files, data files, and log files which were used by database before the failure.

The recovery using database file before failure is not possible if non-volatile storage device is corrupted. It is because control files, datafiles and log files are corrupted so that they can not be used for recovery. In this case, recover the database file by using previously archived backup files and log files, then execute the recovery.

GOLDILOCKS supports both the complete recovery and the incomplete recovery. The complete recovery recovers the datafile to the latest and consistent state by using the log file. The complete recovery targets database, tablespace and data file. For the tablespace and data file, the complete recovery is available for the offline tablespace even during the database service. The complete recovery is divided into the automatic recovery and the manual recovery. The automatic recovery is performed when restaring the database, and the manual recovery uses the recovery statement supported by GOLDILOCKS.

The incomplete recovery is available only for the database, and it recovers to the consistent state of a specific point. The incomplete recovery is performed only manually. It incompletely recovers at once up to the specific point, or it performs the user selective incomplete recovery. The user selective incomplete recovery is a method of which a user selects a log file available to recover and recovers up to that user selected log file.

If both the complete recovery and the incomplete recovery is required, then use redo log and archive log f

iles.

Table 7-4 Database recovery

Recovery	Target	Description
Complete recovery	Database, tablespace, datafile	• Automatic recovery (Recovers at the restart) • Manual recovery (Manually recovers the database, tablespace and data file.)
Incomplete recovery	Database	• Manual recovery only ◦ Incomplete recovery at once ◦ User selective incomplete recovery

Automatic Recovery

The automatic recovery is executed when restarting after a normal or abnormal termination of database. It uses the control files, data files and log files which were used just before the termination.
Especially when the latest database file is corrupted, the backed up database file is recovered and the automatic recovery is executed by using the archive log file.
The recovery is executed in three phases, and they are analysis, redo and undo.

Analysis

Two operations are executed in analysis phase.
First, it searches for the first log to perform the restart recovery. For that, it refers to the most recently executed checkpoint log, and looks for the most recent checkpoint log from the log information recorded in control file.
Second, it initialize the transaction table of the system. It uses transaction information which was executed at checkpoint written on checkpoint log to initialize system transaction table.

Restart Redo

All logs, from the first log obtained in the analysis phase for restart recovery to the last log recorded in the redo log files, execute restart redo. The transaction table is updated when a transaction is completed or a new transaction is started during this process.

Restart Undo

After restart redo is completed, it performs the transaction rollback by performing undo all incomplete transactions remained in the transaction table.

Recovery Using Backup

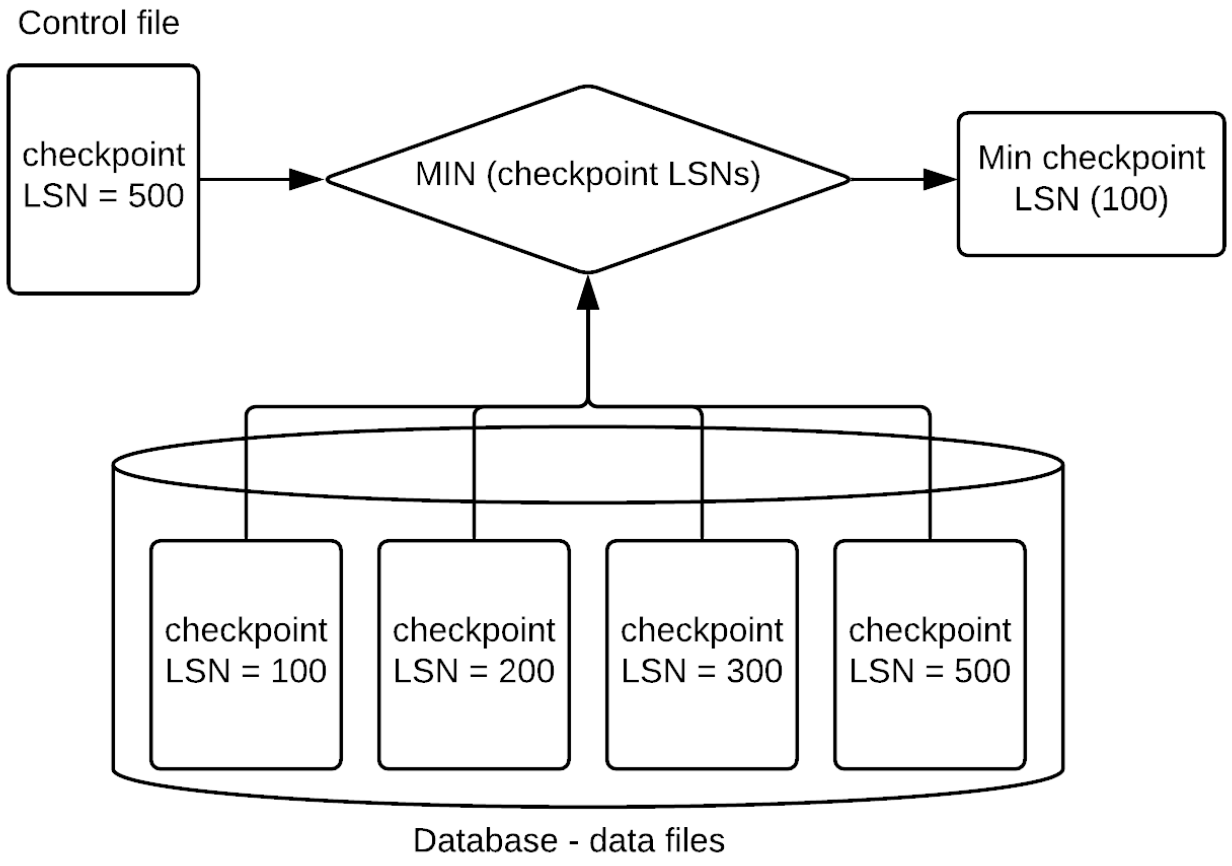
If control files, data files and log files are corrupted or does not exist, the recovery should be performed a fter it is recovered by using the backup file. It is complicated to find the log on which the recovery starts when performing the recovery by using the backed up control file or the data file.

Analysis for Recovery Using Backup

In analysis phase for the recovery using backup, like as the automatic recovery, it searches for the first log for recovery and initializes the transaction table. It searches for the oldest LSN among the checkpoint LSN recorded in all the data file's file header, then selects the minimum value by comparing it with checkpoint LSN recorded in control file to find the first log for recovery.

The checkpoint LSN recorded in data file header stores the checkpointed LSN of the corresponding data fi le. Therefore, when using the backup datafile, select the oldest checkpoint LSN, and then select the mini mum value comparing to the checkpoint LSN in the control file, then the minimum checkpoint LSN for th e recovery is determined.

Figure 2 Procedure to determine the minimum checkpoint LSN for the recovery



Recovery Using Archive Log Files

The recovery using archive log files uses not only redo log files but also archive log files, when the minimum checkpoint LSN defined for the recovery is in an archive log file. The recovery using the backup is executed in a unit of database, tablespace and data file. Database recovery is executed only on MOUNT phase, and the recovery in tablespace and data file unit is executed on MOUNT phase or OPEN phase.

- Restoring the backup data files

Restoring data files is executed by using the full backup or the incremental backup. A user directly executes the full backup by using operating system's file copy command to restore data. On the other hand, the incremental backup is executed by using restoring syntax of GOLDILOCKS. The tablespace should be OFF LINE to restore data files on OPEN phase.

The followings describe how to restore data files using incremental backup.

```
gSQL> ALTER DATABASE RESTORE;

Database altered.

gSQL> ALTER DATABASE RESTORE TABLESPACE TEST_TBS;

Database altered.
```

- Manual recovery after restoring data files

The followings describe how to execute recovery using the syntax of recovery after restoring data files.

```
gSQL> ALTER DATABASE RECOVER;

Database altered.

gSQL> ALTER DATABASE RECOVER TABLESPACE TEST_TBS;

Database altered.
```

Incomplete Recovery

If the restart recovery is not available due to a user mistake during operation, corrupted control files, or corrupted redo log files and archive log files nor can the recovery restore consistency of database, then execute the incomplete recovery. The incomplete recovery restores data only until the point-in-time.
The followings are when the incomplete recovery is required.

Corrupted Control File

Control files are multiplexed, so they can be recovered using uncorrupted files if not all of the multiplexed files are corrupted.
However, if all control files are corrupted, recovery should be executed using the backup control files. In t his case, the complete recovery is impossible because the log information of the control files can be chan ged. The recovery restores only until point-in-time.

Restoring Backup Control File

When control files are corrupted, copy other multiplexed control files to keep control files up-to-date. However, when all multiplexed control files are corrupted, restore the backup control files, then perform the recovery. The log information which is changed after the backup can not be recovered when executin g recovery using the backup data files.

Corrupted Redo Log File

GOLDILOCKS consists redo log files with several log members in a log group to prevent log file corruption. However, if all log members in a log group are corrupted, it can not be recovered using redo log file. In th is case, an incomplete recovery should be executed until uncorrupted log file.

Corrupted Archive Log File

In recovery, if archive log file is corrupted, then incomplete recovery should be executed until uncorrupte d log file. The process is as same as when redo log file is corrupted.

User's Mistake

When a user dropped an important table by mistake, or inserted, updated, deleted wrong data, it should be recovered back to the point before the mistake.

Incomplete Recovery of GOLDILOCKS Database

GOLDILOCKS supports two types of incomplete recovery. One of the recovery is executed until the point which an operator specified. The other recovery is executed in log file unit interactively between an opera tor and system.

The incomplete recovery until a specified point can specify the point-in-time by using a specified log's LSN, specified time, or a specified SCN.

Incomplete recovery is executed for the entire database only on MOUNT phase. Incomplete recovery in th e specific tablespace unit is not supported due to the database consistency problem.

- Incomplete recovery until specified LSN

It searches for log which will complete the incomplete recovery, then executes the recovery until the log's LSN.
The following is an example of executing the incomplete recovery until log LSN 1000.

```
gSQL> ALTER DATABASE RECOVER UNTIL CHANGE 1000;
Database altered;
gSQL> ALTER DATABASE RECOVER UNTIL CHANGE LSN 1000;
Database altered;
```

- Incomplete recovery until specified SCN

It searches for SCN which will complete the incomplete recovery, then executes the recovery until the log 's SCN.
The following is an example of executing the incomplete recovery until SCN 300.

```
gSQL> ALTER DATABASE RECOVER UNTIL CHANGE SCN 300;
Database altered;
```

SCN is not sequentially recorded so even when the incomplete recovery is executed until SCN 300, it may be recovered beyond SCN 300. The following is an example of the recovery when SCN is reversed.

Log: --- LSN 90 (SCN 3) -- LSN 91 (SCN 5) -- LSN 92 (SCN 4)

gSQL> ALTER DATABASE RECOVER UNTIL CHANGE SCN 3;
→ It is recovered until LSN 90.

gSQL> ALTER DATABASE RECOVER UNTIL CHANGE SCN 4;
→ It is recovered until LSN 92. (It is recovered until LSN 92 in which SCN 4 is.)

gSQL> ALTER DATABASE RECOVER UNTIL CHANGE SCN 5;
→ It is recovered until LSN 92. (Both SCN 4, SCN 5 are arbitrarily recovered until SCN5)

- Incomplete recovery until specified time

It searches for the time which will complete the incomplete recovery, then executes the recovery until the specified time.
The following is an example of executing the incomplete recovery until '2017-05-18 16:10:10.00000'.

```
gSQL> ALTER DATABASE RECOVER UNTIL TIME '2017-05-18 16:10:10.000000';
Database altered;
```

- Interactive incomplete recovery

If the log file is corrupted, it executes the recovery until just before the corrupted log file. For that, GOLDI LOCKS suggests an operator the required log files, and the operator executes the incomplete recovery by using the GOLDILOCKS' recommended log file or a new log file.

The following is an example of executing the interactive incomplete recovery of GOLDILOCKS. GOLDILOC KS suggests log files required for the recovery when executing BEGIN for the incomplete recovery. The op erator executes the incomplete recovery by using the GOLDILOCKS' recommended log file, or may descri be the log file for the recovery.

```
gSQL> ALTER DATABASE BEGIN INCOMPLETE RECOVERY;
ERR-01000(14104): Warning: suggestion '/goldilocks/archive_log/archive_0.log'
ERR-01000(14103): Warning: media recovery needs a logfile including log (Lsn 139992)
Database altered.
gSQL> ALTER DATABASE RECOVER AUTOMATICALLY;
ERR-01000(14104): Warning: suggestion '/goldilocks/archive_log/archive_1.log'
ERR-01000(14103): Warning: media recovery needs a logfile including log (Lsn 144143)
Database altered.
gSQL> ALTER DATABASE END INCOMPLETE RECOVERY;
Database altered.
```

Restarting Database after Incomplete Recovery

After the incomplete recovery is completed, a user can not restart database in a normal way. It is because the recovered GOLDILOCKS database by the incomplete recovery has nothing to do with the current redo log files. A user should reset the redo log file to restart database because database is at the previous poin t, and the current redo log file is about the log after then. Use *RESETLOGS* option when restarting GOLDI LOCKS database after the incomplete recovery.

```
gSQL> ALTER SYSTEM OPEN DATABASE;
ERR-HY000(14083): must use RESETLOGS option for database open
gSQL> ALTER SYSTEM OPEN DATABASE RESETLOGS;
System altered.
```

Cautions for Incomplete Recovery

Incomplete recovery is executed until the specific point to create consistent database, but it is not easy to find the specific point. All redo log files are reset after the incomplete recovery. Therefore, all the control files, data files, redo log files in database should be backed up offline before the incomplete recovery. Th en the incomplete recovery should be executed several times to find the correct point.

Archive log files are needed during the incomplete recovery. However, the newly recovered database is di fferent from the previous database, so drop the archive redo log file created by the previous database.

Recovery Examples

Corrupted Control File

GOLDILOCKS database control files store the important information about the physical structure of datab ase and the database consistency. If it is corrupted or dropped by mistake, the database can not be opera ted.

GOLDILOCKS database multiplexes at least 2 up to 8 control files. If there is at least one valid control file, the remaining control files are restored, then the database can be restarted.

When a valid multiplexed control file exists

If the multiplexed control file *'/goldilocks_data/wal/control_1.ctl'* is corrupted, restarting database fails as follows.

```
gSQL> \STARTUP
ERR-HY000(14097): control file is corrupted - '/goldilocks_data/wal/control_1.ctl'
```

Copy the valid control file */goldilocks_data/wal/control_0.ctl* to */goldilocks_data/wal/control_1.ctl*, and dr op the shared memory which failed to restart. Then restart the database.

```
$ cp /goldilocks_data/wal/control_0.ctl /goldilocks_data/wal/control_1.ctl
gSQL> \SHUTDOWN
Shutdown success
gSQL> \STARTUP
Startup success
```

When all multiplexed control files are corrupted

If all multiplexed control files are corrupted, a user can restart the database using backup control files afte r incomplete recovery. The database's physical structure can be changed after backing up control files. Th erefore, the incomplete recovery should be executed when restoring control files using backup control fil es. Archive log files and redo log files still exist even after incomplete recovery. Therefore, an administrato r executes GOLDILOCKS interactive incomplete recovery to manually restore until 'CURRENT' state redo lo g file.

The backup control file can be copied to multiplexed control files by operating system's file copy feature, or they can be restored by GOLDILOCKS database's recovery feature as follows. The control file recovery c an be executed only in NOMOUNT phase of GOLDILOCKS multilevel startup.

```
gSQL> \STARTUP NOMOUNT
Startup success
gSQL> ALTER DATABASE RESTORE CONTROLFILE FROM '/goldilocks_data/backup/backup.ctl';
Database altered.
```

After restoring the backup control files, execute the incomplete recovery in MOUNT phase as follows.

```
gSQL> ALTER SYSTEM MOUNT DATABASE;
System altered.
gSQL> ALTER DATABASE BEGIN INCOMPLETE RECOVERY;
ERR-01000(14104): Warning: suggestion '/goldilocks/archive_log/archive_0.log'
ERR-01000(14103): Warning: media recovery needs a logfile including log (Lsn 137499)
Database altered.
gSQL> ALTER DATABASE RECOVER AUTOMATICALLY;
ERR-01000(14104): Warning: suggestion '/goldilocks/archive_log/archive_1.log'
ERR-01000(14103): Warning: media recovery needs a logfile including log (Lsn 137667)
Database altered.
gSQL> ALTER DATABASE RECOVER '/goldilocks/wal/redo_1_0.log';
ERR-01000(14104): Warning: suggestion '/goldilocks/archive_log/archive_2.log'
ERR-01000(14103): Warning: media recovery needs a logfile including log (Lsn 137672)
Database altered.
gSQL> ALTER DATABASE END INCOMPLETE RECOVERY;
Database altered.
gSQL> ALTER SYSTEM OPEN DATABASE RESETLOGS;
System altered.
```

Corrupted Data File

If a data file is corrupted or dropped, the complete recovery is executed by using the backup data files. Th e full backup restores the data files by copying the backup files, and the incremental backup restores the data files by using the GOLDILOCKS' restoring syntax.
The restoration and recovery of data files are executed in database unit or in tablespace unit. It can also b e executed in the tablespace unit of the corresponding data file. The recovery in tablespace unit can be ex ecuted in MOUNT phase or OPEN phase. The tablespace should be in OFFLINE state to restore and recove r data files on OPEN phase.

- Recovering the corrupted data files using full backup in MOUNT phase

Execute the complete recovery after copying the backup data file */goldilocks/backup/test.dbf* to */goldiloc ks/db/test.dbf*.

```
$ cp /goldilocks/backup/test.dbf /goldilocks/db/test.dbf
gSQL> \STARTUP MOUNT
System altered.
gSQL> ALTER DATABASE RECOVER;
Database altered.
gSQL> ALTER SYSTEM OPEN DATABASE;
```

```
System altered.
```

- Recovering the corrupted data files using full backup in OPEN phase

```
gSQL> SELECT IS_ONLINE FROM V$TABLESPACE WHERE TBS_NAME = 'TEST_TBS';
IS_ONLINE
---------
FALSE
1 row selected.
$ cp /goldilocks/backup/test.dbf /goldilocks/db/test.dbf
gSQL> ALTER DATABASE RECOVER TABLESPACE TEST_TBS;
Database altered.
gSQL> ALTER TABLESPACE TEST_TBS ONLINE;
Tablespace altered.
```

- Recovering the corrupted data files using incremental backup in MOUNT phase

```
gSQL> \STARTUP MOUNT
System altered.
gSQL> ALTER DATABASE RESTORE;
Database altered.
gSQL> ALTER SYSTEM OPEN DATABASE;
System altered.
```

- Recovering the corrupted data files using incremental backup in OPEN phase

```
gSQL> SELECT IS_ONLINE FROM V$TABLESPACE WHERE TBS_NAME = 'TEST_TBS';
IS_ONLINE
---------
FALSE
1 row selected.
gSQL> ALTER DATABASE RESTORE TABLESPACE TEST_TBS;
Database altered.
gSQL> ALTER TABLESPACE TEST_TBS ONLINE;
Tablespace altered.
```

User's Mistake (Table Dropping or Wrong Insert/drop/update)

GOLDILOCKS database supports DDL rollback of table and index if the table *TEST* is dropped by mistake. Namely, a user can rollback to cancel the table dropping instead of committing as follows even if a user d ropped the table.

```
gSQL> DROP TABLE TEST;
Table dropped.
gSQL> ROLLBACK;
Rollback complete.
gSQL> \DESC TEST
COLUMN_NAME TYPE          IS_NULLABLE
----------- ------------- -----------
I1          NUMBER(10,0)  TRUE
I2          CHARACTER(10) TRUE
gSQL> DROP TABLE TEST;
Table dropped.
gSQL> COMMIT;
Commit complete.
gSQL> \DESC TEST
ERR-42000(16040): table or view does not exist :
SELECT *   FROM TEST  WHERE 1 = 0
               *
ERROR at line 1:
```

If table dropping is committed it can not be rolled back. Therefore, execute GOLDILOCKS incomplete rec overy to recover until the specific point of the database using backup. Then, the data is recovered until th e time before the table dropping. Restart the database after that.

The backup file at the point before the table dropping is used to restore the table in incomplete media re covery. The correct point can be found, as described above, by repeating the recovery several times to fin d the time of table dropped. At that time, the gdump tool is used to dump the log file and analyze it.

Assuming LSN is 1000 at the time after table dropping, the incomplete recovery is executed as follows.

```
gSQL> \STARTUP MOUNT
System altered.
gSQL> ALTER DATABASE RECOVER UNTIL CHANGE 1000;
Database altered.
gSQL> ALTER SYSTEM OPEN DATABASE RESETLOGS;
System altered
gSQL> \DESC TEST
COLUMN_NAME TYPE          IS_NULLABLE
----------- ------------- -----------
I1          NUMBER(10,0)  TRUE
I2          CHARACTER(10) TRUE
```

Corrupted Log File (Archive File, Redo Log File)

- Corrupted archive log file during recovery

Assume that the data files are corrupted and a user are executing recovery using the backup data files. Al so, assume that the specific archive log file is corrupted during the recovery so that the recovery can not b e completed.

For example, there are archive log files such as 'archive_0.log', 'archive_1.log', 'archive_2.log', 'archive_3 .log'. An 'archive_3.log' is corrupted, and the recovery can not be executed. In this case, incomplete reco very is executed until 'archive_2.log', and the database is restarted.

```
gSQL> \STARTUP MOUNT
System altered
gSQL> ALTER DATABASE BEGIN INCOMPLETE RECOVERY;
ERR-01000(14104): Warning: suggestion '/goldilocks/archive_log/archive_0.log'
ERR-01000(14103): Warning: media recovery needs a logfile including log (Lsn 139992)
Database altered.
gSQL> ALTER DATABASE RECOVER AUTOMATICALLY;
ERR-01000(14104): Warning: suggestion '/goldilocks/archive_log/archive_3.log'
ERR-01000(14103): Warning: media recovery needs a logfile including log (Lsn 194143)
Database altered.
gSQL> ALTER DATABASE END INCOMPLETE RECOVERY;
Database altered.
gSQL> ALTER SYSTEM OPEN DATABASE RESETLOGS;
System altered
```

- Corrupted redo log file

Assume that a failure occurs when operating database so the CURRENT log group in which logs are flush ed is corrupted.

For example, when an abnormal termination occurs in the state of the following log groups, the manual r ecovery is executed and completes the incomplete recovery. It is because the log group 3, 0 is not archive d yet.

Table 7-5 Log group state

Log group	Log group state	Log file sequence no.	Prev last LSN
Log group 0	ACTIVE	8	80000
Log group 1	CURRENT	9	90000
Log group 2	INACTIVE	6	60000
Log group 3	ACTIVE	7	70000

```
gSQL> \STARTUP MOUNT
System altered
gSQL> ALTER DATABASE BEGIN INCOMPLETE RECOVERY;
ERR-01000(14104): Warning: suggestion '/goldilocks/archive_log/archive_0.log'
ERR-01000(14103): Warning: media recovery needs a logfile including log (Lsn 1000)
Database altered.
gSQL> ALTER DATABASE RECOVER AUTOMATICALLY;
ERR-01000(14104): Warning: suggestion '/goldilocks/archive_log/archive_7.log'
ERR-01000(14103): Warning: media recovery needs a logfile including log (Lsn 70001)
Database altered.
gSQL> ALTER DATABASE RECOVER '/goldilocks/wal/redo_3_0.log';
ERR-01000(14104): Warning: suggestion '/goldilocks/archive_log/archive_8.log'
ERR-01000(14103): Warning: media recovery needs a logfile including log (Lsn 80001)
Database altered.
gSQL> ALTER DATABASE RECOVER '/goldilocks/wal/redo_0_0.log';
ERR-01000(14104): Warning: suggestion '/goldilocks/archive_log/archive_9.log'
ERR-01000(14103): Warning: media recovery needs a logfile including log (Lsn 90001)
Database altered.
gSQL> ALTER DATABASE END INCOMPLETE RECOVERY;
Database altered.
gSQL> ALTER SYSTEM OPEN DATABASE RESETLOGS;
System altered
```

More Recent Datafile Than Log

All tablespaces created in GOLDILOCKS database consists of pages, and each page set the log LSN which was recorded by a transaction having updated that page last is set as the page LSN. Therefore, all page LSNs in the datafile have the same or smaller value than the LSN of the latest log recorded in the redo log file of the log group.

When restartig the database, if a specific page's LSN of the data file has a bigger value than the latest log's LSN, then it corrupts the database consistency and the normal service is not available. GOLDILOCKS database checks the data file and log when restarting so that this abnormal situation does not happen.

If any page whose LSN has bigger value than the latest log's LSN is in the data file, then restarting the database fails as follows.

```
gSQL> \STARTUP
ERR-HY000(14114): exist inconsistent datafiles; need to restore more older backup datafiles or
more recent redo logfiles
```

To solve this problem, restore the backup data file consisting of LSNs smaller than the latest log LSN. Or, restart the database after restoring the log file on which the LSN log bigger than the data file is recorded.

Check the trace file to find the data file to restore.

For example, if the following messages are output on the trace file when restarting fails, then the page w hose LSN is '126787' in '/data/db/system_dic.dbf' data file, and this value is bigger than the latest log LS N of the log file '126652'. Therefore, for the restart, restore the previous backup data file, or restore the l og file on which the log LSN same or bigger than '126787' is recorded. Also, if several pages of the data f ile has a LSN value bigger than the log file LSN, then restore the log file bigger than the maximum value a mong them to restart and provide the service.

```
[2016-01-15 12:41:14.045679 THREAD(10581,139799401453312)] [INFORMATION]
[STARTUP_SM] the max page lsn '126787' of datafile '/data/db/system_dict.dbf' is more recent
than the latest redo log lsn '126652'.
[2016-01-15 12:41:14.045705 THREAD(10581,139799401453312)] [INFORMATION]
[STARTUP_SM] the max page lsn '126830' of datafile '/data/db/system_undo.dbf' is more recent
than the latest redo log lsn '126652'.
[2016-01-15 12:41:14.045729 THREAD(10581,139799401453312)] [INFORMATION]
[STARTUP_SM] the max page lsn '126829' of datafile '/data/db/test_log.dbf' is more recent than
the latest redo log lsn '126652'.
```

in doubt Transaction Recovery in Cluster Environment

The transactions in the cluster environment are divided into global transaction, domain transaction and lo cal transaction. The global transaction is performed in two or more cluster groups, the domain transactio n is performed in a single cluster group and the local transaction is performed in a single cluter member.

The local transaction uses only the local member's log when performing the recovery. The domain transa ction uses the local member's log when performing the recovery, and performs the rollback when an abn ormal termination occurs without completing the transaction, and performs the synchronization with a gr oup member through the rebalance if needed

The global transaction uses 2 phase commit protocol to commit. 2 phase commit is performed as follows for GOLDILOCKS global transaction.

- PREPARE phase
  - It transfers PREPARE messages from the driver member to all members, and waits for the respon d message.
  - It moves on to COMMIT phase when it receives the respond message of PREPARE from all memb ers, or it rolls back when at least one member fails or does not respond.
- COMMIT phase
  - It transfers COMMIT messages from the driver member to all members, and waits for the respon d message.
  - It commits even when a member fails.

If a member is abnormally terminated without recording the commit log on COMMIT phase, it should obt ain the state information from other members when restarting. It is because it is unable to know if the gl obal transaction in 'PREPARE' state was committed or rolled back, when restarting.

For that, GOLDILOCKS records the committed global transaction information in transaction record form i n MEM_TRANS_TBS. Also, when recording a new record in that record, the previous record is recorded in the global transaction log file if needed. Later, the abnormally terminated member performs the restart re covery or the manual recovery by using the log. It performs the recovery by obtaining the transaction CO MMIT/ROLLBACK information from members in service when *in doubt* transaction in 'PREPARE' state is r emained.

GOLDILOCKS creates the global transaction log file consisting of two log groups in advance and records *i n doubt* transaction information in the log file when creating the database. In this case, if the log file is fu ll, then the next log file is used. Filled log file is reused after it is archived by the archive system thread. Th e global transaction log file is always archived in cluster environment regardless of ARCHIVELOG mode. It is because it is required to allow it to refer to the previous *in doubt* transaction information.

8.

GOLDILOCKS Database Replication

8.1 Overview

This chapter describes CYCLONE, LOGMIRROR and CYFILE.

GOLDILOCKS supports two types of replication, and they are CYCLONE and LOGMIRROR. CYCLONE repli cates transactions using CDC, and LOGMIRROR replicates redo log files in the source database.

CYFILE uses CDC method to store/ record the transaction which was applied to the original database in C omma-Separated Values (CSV) format file.

Table 8-1 Replication tool

Tool	Replication target	Description
CYCLONE	Transaction	It uses CDC method, and replicates the transaction reflected in mast er, then reflect it to slave.
LOGMIRROR	Redo log file	It identically replicates redo file in master database to slave.
CYFILE	Transaction	It uses CDC method to store/ record the transaction in CSV format fi le.

- CYCLONE
  - It uses Change Data Capture (CDC) method to analyze and treat redo log files of the source data base, then applies them to a remote database.
  - It supports only asynchronous (async) method because it analyzes contents stored in database's r edo log file.
- LOGMIRROR
  - It replicates redo log files stored in the source database to a remote database.
  - It is used to avoid the data loss of CYCLONE which is executed in async method.
- CYFILE
  - It analyzes the redo log file of the database in CDC method, and stores it in CSV format data file.
  - A user can perform the replication by using the corresponding file or perform ETL according to th e purpose of the development.

8.2 Operating Method

CYCLONE

For more information about general operating method and option, refer to **CYCLONE**.

Adding and Deleting Nodes

CYCLONE is performed in a group unit, and it is as same as the replication nodes. Add a group when add ing nodes, drop a group when deleting nodes.

Examples of Adding Nodes

- Record the group 2 to be added in master configuration file.
  - Set a unique port per each group.
  - The default master configuration file is $GOLDILOCKS_DATA/conf/cyclone.master.conf.
  - The following is an example of group 1 node in operation.

```
...
...
GROUP_NAME = Group1
{
    PORT = 21102
    CAPTURE_TABLE =
    (
        testTable1,
        testTable2
    )
}
```

  - The following is an example of group 2 node to be added.

```
GROUP_NAME = Group2
{
    PORT = 21103
    CAPTURE_TABLE =
    (
        testTable5,
```

```
        testTable6
    )
}
```

- Record the group 2 to be added in slave configuration file.
    - The port of group 2 should be as same as the port added to the existing master.
    - The default slave configuration file is $GOLDILOCKS_DATA/conf/cyclone.slave.conf.

    - The following is an example of group 1 node in operation.

```
...
...
GROUP_NAME = Group1
{
    PORT = 21102
    APPLY_TABLE =
    (
        testTable1 To testTable3,
        testTable2 To testTable4
    )
}
```

    - The following is an example of group 2 node to be added.

```
GROUP_NAME = Group2
{
    PORT = 21103
    APPLY_TABLE =
    (
        testTable5 To testTable7,
        testTable6 To testTable8
    )
}
```

- Execute and check the added group 2 node in the master device as follows.

```
prompt> cyclone --master --start --group Group2
[GROUP2] Startup done as Master.
prompt> cyclone --master --status
=====================================
¦       CYCLONE STATUS - MASTER      ¦
=====================================
```

```
 GROUP1 Running...
 GROUP2 Running...
-------------------------------------
```

- Execute and check the added group 2 node in the slave device as follows.

```
prompt> cyclone --slave --start --group Group2
[GROUP2] Startup done as Slave.
prompt> cyclone --slave --status
=====================================
        CYCLONE STATUS - SLAVE
=====================================
 GROUP1 Running...
 GROUP2 Running...
-------------------------------------
```

Examples of Deleting Nodes

- Terminate the group 2 node to be deleted in the slave device as follows.

```
prompt> cyclone --slave --stop --group Group2
stop done.
prompt> cyclone --slave --status
=====================================
CYCLONE STATUS - SLAVE
=====================================
GROUP1 Running...
-------------------------------------
```

- Terminate the group 2 node to be deleted in the master device as follows.

```
prompt> cyclone --master --stop --group Group2
stop done.
prompt> cyclone --master --status
=====================================
CYCLONE STATUS - MASTER
=====================================
GROUP1 Running...
-------------------------------------
```

- Drop the group 2 node to be deleted from the master configuration file.
  - The default master configuration file is $GOLDILOCKS_DATA/conf/cyclone.master.conf.

    - The following is an example of group 1 node in operation.

```
...
...
GROUP_NAME = Group1
{
    PORT = 21102
    CAPTURE_TABLE =
    (
        testTable1,
        testTable2
    )
}
```

    - The following is an example of deleting group 2 node.

```
~~GROUP_NAME = Group2~~
~~{~~
~~    PORT = 21103~~
~~    CAPTURE_TABLE =~~
~~    (~~
~~        testTable5,~~
~~        testTable6~~
~~    )~~
~~}~~
```

- Drop the group 2 node to be deleted from the slave configuration file as follows.
    - The default slave configuration file is $GOLDILOCKS_DATA/conf/cyclone.slave.conf.

    - The following is an example of group 1 node in operation.

```
...
...
GROUP_NAME = Group1
{
    PORT = 21102
    APPLY_TABLE =
    (
        testTable1 To testTable3,
        testTable2 To testTable4
    )
}
```

- The following is an example of deleting group 2 node.

```
GROUP_NAME = Group2
{
    PORT = 21103
    APPLY_TABLE =
    (
        testTable5 To testTable7,
        testTable6 To testTable8
    )
}
```

Initializing Replication

Initializing replication is executed when existing replication nodes or a group's table gives up execution due to DDL operation. A specific node or entire node can be initialized.

The initializing replication is performed by restarting replication being operated in slave using --reset option.
On the other hand, master does not require any operation.

Examples of Initializing Replication on a Specific Node

- Terminate the group 2 node to be initialized in the slave device as follows.

```
prompt> cyclone --slave --stop --group Group2
stop done.
prompt> cyclone --slave --status
=====================================
        CYCLONE STATUS - SLAVE
=====================================
 GROUP1 Running...
-------------------------------------
```

- Restart group 2 node in slave device using --reset option.
  - Master device does not require any operation.
  - Replication restarts from the current point.

```
prompt> cyclone --slave --start --reset --group Group2
[GROUP2] Startup done as Slave.
prompt> cyclone --slave --status
=====================================
```

```
        CYCLONE STATUS - SLAVE
=====================================
 GROUP1 Running...
 GROUP2 Running...
-------------------------------------
```

Examples of Initializing Replication of All Nodes

- Terminate all CYCLONE in operation in the slave device as follows.

```
prompt> cyclone --stop --slave
```

- Restart CYCLONE in slave device using --reset option.
  - Master device does not require any operation.
  - Replication starts from the current point.

```
prompt> cyclone --slave --start --reset
[GROUP1] Startup done as Slave.
[GROUP2] Startup done as Slave.
prompt> cyclone --slave --status
=====================================
        CYCLONE STATUS - SLAVE
=====================================
 GROUP1 Running...
 GROUP2 Running...
-------------------------------------
```

LOGMIRROR

For more information about general operation method and options, refer to **LOGMIRROR**.

Retrieving LOGMIRROR State

GOLDILOCKS includes the response waiting procedure of LOGMIRROR when GOLDILOCKS interworks wi th LOGMIRROR. If LOGMIRROR is waiting for response, GOLDILOCKS is also waiting in a blocked state. T his state can be retrieved in v$system_stat.

```
gSQL> SELECT * FROM V$SYSTEM_STAT WHERE STAT_NAME='LOG_MIRROR_SYNC_STATE';
STAT_NAME             STAT_VALUE COMMENTS
--------------------- ---------- ----------------------------------------------
```

```
LOG_MIRROR_SYNC_STATE      0 logmirror sync state( 0 : sync, 1 : blocked )
1 row selected.
```

If STAT_VALUE is 0, it is not a standby state but an ordinary state. If STAT_VALUE is 1, it is a blocked state waiting for a response. To restart GOLDILOCKS service while LOGMIRROR is waiting for a response, LOG MIRROR service can be stopped by modifying LOG_MIRROR_TIMEOUT as follows.

```
gSQL> ALTER SYSTEM SET LOG_MIRROR_TIMEOUT = 20;
System altered.
```

Initializing Replication

Initializing replication of LOGMIRROR should be done manually. This is to prevent data dropping or the u nrecoverable situation driven by the user's incorrect option usage.

Control files and redo log files are stored in LOGMIRROR slave. The required information for the operation is stored and updated in the control files.

Examples of Initializing Replication

- Terminate LOGMIRROR in operation in the slave device as follows.

```
prompt> logmirror --slave --stop
stop done.
```

- Terminate LOGMIRROR in operation in the master device as follows.

```
prompt> logmirror --master --stop
stop done.
```

- Drop the replicated control files and redo log files from the slave device.
  - For more information about the path, refer to 'LOG_PATH' option described in the slave configur ation file.
  - The default LOGMIRROR slave configuration file is $GOLDILOCKS_DATA/conf/logmirror.slave.co nf.

CYFILE

For more information about how to operate and its options, refer to **CYFILE**.

Starting, Stopping and Status Checking of cyfile

Example

- Start cyfile from the current moment by using the default configuration file.

```
prompt> cyfile --start --reset all
Startup done.
```

- Stop cyfile.

```
prompt> cyfile --stop
Stop done.
```

- CYFILE is operated in groups and the data file is created in groups.

```
prompt> cyfile --status
cyfile --status
=====================================
¦          CYFILE STATUS            ¦
=====================================
 GROUP1 Running...
=====================================
```

8.3 Trace Log

The followings are detailed information about the trace log.

Table 8-2 Trace log

Name	Category	File name
CYCLONE	Master	cyclone_master_GROUP_NAME.trc
	Slave	cyclone_slave_GROUP_NAME.trc
LOGMIRROR	Master	LogMirror_master.trc
	Slave	LogMirror_slave.trc
CYFILE	-	cyfile_GROUP_NAME.trc

Troubleshooting of CYCLONE

The followings are error messages and troubleshooting of CYCLONE.

Table 8-3 Troubleshooting of CYCLONE

Error message	Solution
Service is not available	Ensure that GOLDILOCKS is normally running.
table does not exist	Check the table name described in the configuration file.
schema does not exist	Check the schema name described in the configuration file.
previously added. Maybe duplicated	Check if a table is duplicated in the configuration file.
table must have a primary key	Ensure that a table in the configuration file has the primary key.
internal error occurred.	Check the error details.
table must set supplemental log	Ensure that supplemental logging is executed in GOLDILOCKS.
group XXX is already running	Check if the corresponding group is already running.
GOLDILOCKS_DATA system environment is invalid	Ensure that GOLDILOCKS_DATA environment variable is set.
log file reused or invalid. restart cyclone with '--reset' option	This error occurs when the redo log file is reused or archived redo log file does not exist. In this case, initialize CYCLONE and restart it.
fail to analyze flow	This error occurs when analyzing an abnormal redo log file. Ensure that the release version between master and slave are same.
Communication link failure	Check the network state. Restart CYCLONE.
Master disconnect abnormally	Check the network state. Restart CYCLONE.
Protocol error occurred	Check the error details.
Already slave connected	Check if the slave is already running.
Invalid group name	Check the specified group name at the startup/termination. The group name

Error message	Solution
	should be same as described in the configuration file.
Invalid capture information	This error occurs when the existing operating information is abnormal. In this case, initialize CYCLONE and restart it.
Redo log file read timeout	Ensure that the archived redo log files normally exist.
Invalid archive log file	The archived redo log file is not normal. In this case, initialize CYCLONE and r estart it.
Fail to write file	Check the available space in the disk, and restart CYCLONE.
Invalid Meta File	This error occurs when the meta files which are managed by CYCLONE are co rrupted. In this case, initialize CYCLONE and restart it.
Redo log file does not exist	Ensure that GOLDILOCKS which is operated as master is normally running.
[APPLIER-INSERT] XXX	INSERT is failed due to XXX.
[APPLIER-DELETE] XXX	DELETE is failed due to XXX.
[APPLIER-UPDATE] XXX	UPDATE is failed due to XXX.

Troubleshooting of LOGMIRROR

The followings are error messages and troubleshooting of LOGMIRROR.

Table 8-4 Troubleshooting of LOGMIRROR

Error message	Solution
Service is not available	Ensure that GOLDILOCKS is normally running.
Invalid Protocol value	Check the error details.
file does not exist	Ensure that the corresponding file normally exists.
invalid Control file	The control file is corrupted. Initialize and restart LOGMIRROR.
Communication link failure	Check the network state. Restart LOGMIRROR.
GOLDILOCKS_DATA system en vironment is invalid	Ensure that GOLDILOCKS_DATA environment variable is set.
There is no Shared Memory Are a for LogMirror	Ensure that LOG_MIRROR_MODE property is normally set to 'enabled' in properti es of GOLDILOCKS which is operated as master.
Master disconnect abnormally	Check the network state. Restart LOGMIRROR.
Invalid Log File	Ensure that the corresponding file normally exists.
Connection Information does n ot exist	Ensure that GOLDILOCKS connecting information in configuration files is normal.
Archive Log File does not exist	Ensure that ARCHIVELOG_MODE in GOLDILOCKS which is operated as master is normally set.

Troubleshooting of CYFILE

The followings are error messages and troubleshooting of CYFILE.

Error message	Solution
Service is not available	Ensure that GOLDILOCKS is normally running.
table does not exist	Check the table name described in the configuration file.
schema does not exist	Check the schema name described in the configuration file.
previously added. Maybe duplicated	Check if a table is duplicated in the configuration file.
table must have a primary key	Ensure that a table in the configuration file has the primary key.
internal error occurred.	Check the error details.
table must set supplemental log	Ensure that supplemental logging is executed in GOLDILOCKS.
group XXX is already running	Check if the corresponding group is already running.
GOLDILOCKS_DATA system environment is invalid	Ensure that GOLDILOCKS_DATA environment variable is set.
log file reused or invalid. restart cyfile with '--reset' option	This error occurs when the redo log file is reused or archived redo log file does not exist. In this case, initialize cyfile and restart it.
fail to analyze flow	This error occurs when analyzing an abnormal redo log file. Ensure that the release version between master and slave are same.
Invalid group name	Check the specified group name at the startup/termination. The group name should be same as described in the configuration file.
Invalid capture information	This error occurs when the existing operating information is abnormal. In this case, initialize cyfile and restart it.
Redo log file read timeout	Ensure that the archived redo log files normally exist.
Invalid archive log file	The archived redo log file is not normal. In this case, initialize cyfile and restart it.
Fail to write file	Check the available space in the disk, and restart cyfile.
Invalid Meta File	This error occurs when the meta files which are managed by cyfile are corrupted. In this case, initialize cyfile and restart it.
Redo log file does not exist	Ensure that GOLDILOCKS is normally running.
Control has broken(CRC Error)	The control file of cyfile is damaged. Retry it by using the mirror file.

9.

Database Information

9.1 DICTIONARY_SCHEMA

DICTIONARY_SCHEMA contains views or tables to get SQL objects and their information in the system.


Views and tables of DICTIONARY_SCHEMA can be retrieved only in open phase.

Execute *DictionarySchema.sql* as follows to use the views.

- For standalone

```
% gsql sys gliese --as sysdba --import $GOLDILOCKS_HOME/admin/standalone/DictionarySchema.sql
```

- For cluster

```
% gsql sys gliese --as sysdba --import $GOLDILOCKS_HOME/admin/cluster/DictionarySchema.sql
```

Information is retrieved as follows according to the names of views or tables.

- ALL-family view
  - The view name begins with ALL_
  - Information about accessible objects by a current user
- DBA-family view
  - The view name begins with DBA_
  - Information about all objects whose current user has DBA privileges (ACCESS CONTROL ON DAT ABASE).
- USER-family view
  - The view name begins with USER_
  - Information about objects which are owned by the current user

Views of ALL_family

It retrieves information about objects accessible by a current user.

ALL_ALL_TABLES

ALL_ALL_TABLES describes the object tables and relational tables accessible to the current user.

Table 9-1 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the table
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the table
CLUSTER_NAME	VARCHAR(128)	• reserved Name of the cluster
IOT_NAME	VARCHAR(128)	• reserved Name of the index-organized table
STATUS	VARCHAR(32)	If a previous DROP TABLE operation failed, indicates whether the table is un usable (UNUSABLE) or valid (VALID)
PCT_FREE	NUMBER	Minimum percentage of free space in a block
PCT_USED	NUMBER	Minimum percentage of used space in a block
INI_TRANS	NUMBER	Initial number of transactions
MAX_TRANS	NUMBER	Maximum number of transactions
INITIAL_EXTENT	NUMBER	Size of the initial extent (in bytes)
NEXT_EXTENT	NUMBER	Size of secondary extents (in bytes)
MIN_EXTENTS	NUMBER	Minimum number of extents allowed in the segment
MAX_EXTENTS	NUMBER	Maximum number of extents allowed in the segment
PCT_INCREASE	NUMBER	• reserved Percentage increase in extent size
FREELISTS	NUMBER	• reserved Number of process freelists allocated to the segment
FREELIST_GROUPS	NUMBER	• reserved Number of freelist groups allocated to the segment
LOGGING	VARCHAR(3)	Indicates whether or not changes to the table are logged
BACKED_UP	VARCHAR(1)	• reserved Indicates whether the table has been backed up since the last modification (Y) or not (N)
NUM_ROWS	NUMBER	Number of rows in the table

Column name	Data type	Description
BLOCKS	NUMBER	Number of used blocks in the table
ANAL_BLOCKS	NUMBER	Number of used blocks in the table when most recently analyzed
EMPTY_BLOCKS	NUMBER	• reserved Number of empty (never used) blocks in the table
AVG_SPACE	NUMBER	• reserved Average available free space in the table
CHAIN_CNT	NUMBER	• reserved Number of rows in the table that are chained from one data block to another or that have migrated to a new block, requiring a link to preserve the old rowid
AVG_ROW_LEN	NUMBER	• reserved Average row length, including row overhead
AVG_SPACE_FREELIST_BLOCKS	NUMBER	• reserved Average freespace of all blocks on a freelist
NUM_FREELIST_BLOCKS	NUMBER	• reserved Number of blocks on the freelist
DEGREE	VARCHAR(32)	• reserved Number of threads per instance for scanning the table, or DEFAULT
INSTANCES	VARCHAR(32)	• reserved Number of instances across which the table is to be scanned, or DEFAULT
CACHE	VARCHAR(1)	• reserved Indicates whether the table is to be cached in the buffer cache (Y) or not (N)
TABLE_LOCK	VARCHAR(32)	Indicates whether table locking is enabled (ENABLED) or disabled (DISABLED)
SAMPLE_SIZE	NUMBER	Sample size used in analyzing the table
LAST_ANALYZED	TIMESTAMP(6) WITHOUT TIME ZONE	Date on which the table was most recently analyzed
PARTITIONED	VARCHAR(3)	• reserved Indicates whether the table is partitioned (YES) or not (NO)
IOT_TYPE	VARCHAR(32)	• reserved If the table is an index-organized table, then IOT_TYPE is IOT, IOT_OVERFLOW, or IOT_MAPPING.
OBJECT_ID_TYPE	VARCHAR(32)	• reserved Indicates whether the object ID (OID) is USER-DEFINED or SYSTEM GENERATED
TABLE_TYPE_OWNER	VARCHAR(128)	• reserved If an object table, owner of the type from which the table is created
TABLE_TYPE	VARCHAR(128)	• reserved If an object table, type of the table

Column name	Data type	Description
TEMPORARY	VARCHAR(1)	Indicates whether the table is temporary (Y) or not (N)
SECONDARY	VARCHAR(1)	• reserved Indicates whether the table is a secondary object created by cartridge
NESTED	VARCHAR(3)	• reserved Indicates whether the table is a nested table (YES) or not (NO)
BUFFER_POOL	VARCHAR(32)	• reserved Buffer pool to be used for table blocks
FLASH_CACHE	VARCHAR(32)	• reserved Database Smart Flash Cache hint to be used for table blocks
CELL_FLASH_CACHE	VARCHAR(32)	Cell flash cache hint to be used for table blocks
ROW_MOVEMENT	VARCHAR(32)	• reserved If a partitioned table, indicates whether row movement is enabled (ENABLED) or disabled (DISABLED)
GLOBAL_STATS	VARCHAR(3)	• reserved For partitioned tables, indicates whether statistics for the table as a whole (global statistics) are accurate (YES)
USER_STATS	VARCHAR(3)	• reserved Indicates whether statistics were entered directly by the user (YES) or not (NO)
DURATION	VARCHAR(32)	Indicates the duration of a temporary table, the value is in (TRANSACTION, SESSION)
SKIP_CORRUPT	VARCHAR(32)	• reserved Indicates whether Database ignores blocks marked corrupt during table and index scans (ENABLED) or raises an error (DISABLED)
MONITORING	VARCHAR(3)	• reserved Indicates whether the table has the MONITORING attribute set (YES) or not (NO)
CLUSTER_OWNER	VARCHAR(128)	• reserved Owner of the cluster, if any
DEPENDENCIES	VARCHAR(32)	• reserved Indicates whether row-level dependency tracking is enabled (ENABLED) or disabled (DISABLED)
COMPRESSION	VARCHAR(32)	• reserved Indicates whether table compression is enabled (ENABLED) or not (DISABLED)
COMPRESS_FOR	VARCHAR(32)	• reserved Default compression for what kind of operations
DROPPED	VARCHAR(3)	Indicates whether the table has been dropped and is in the recycle bin (YES) or not (NO)
READ_ONLY	VARCHAR(3)	Indicates whether the table IS READ-ONLY (YES) or not (NO)

Column name	Data type	Description
SEGMENT_CREATED	VARCHAR(3)	Indicates whether the table segment has been created (YES) or not (NO)

ALL_ARGUMENTS

ALL_ARGUMENTS lists all arguments of functions, procedures.

Table 9-2 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of function, procedures or pacakage
SCHEMA_NAME	VARCHAR(128)	Schema Name of function, procedures or package
OBJECT_NAME	VARCHAR(128)	Name of function, procedures
PACKAGE_NAME	VARCHAR(128)	Package Name of function, procedures
OBJECT_ID	NUMBER	ID of a function, procedures
SUBPROGRAM_ID	NUMBER	ID of procedures in pacakage
ARGUMENT_NAME	VARCHAR(128)	Name of argument or attribute name of record type argument
POSITION	NUMBER	Position of argument or position of attribute in record type
SEQUENCE	NUMBER	Sequential order of argument and its attributes
DATA_LEVEL	NUMBER	Nesting depth of the argument for composite types
DATA_TYPE	VARCHAR(128)	Data type of the argument
DEFAULTED	VARCHAR(1)	Whether or not the argument is defaulted
DEFAULT_VALUE	VARCHAR(1)	Reserved for future use
DEFAULT_LENGTH	VARCHAR(1)	Reserved for future use
IN_OUT	VARCHAR(32)	Direction of the argument (IN, OUT, IN/OUT)
DATA_LENGTH	NUMBER	Length of the column(in bytes)
DATA_PRECISION	NUMBER	Length in decimal digits(NUMBER) or binary digits(FLOAT)
DATA_SCALE	NUMBER	Digits to the right of the decimal point in a number
RADIX	NUMBER	Argument radix for a number
CHARACTER_SET_NAME	VARCHAR(128)	Character set name for the argument
TYPE_OWNER	VARCHAR(128)	Owner of the type of the argument
TYPE_NAME	VARCHAR(128)	Name of the type of the argument
TYPE_SUBNAME	VARCHAR(128)	Name of the type of the argument declared in package
TYPE_LINK	VARCHAR(128)	Name of the type of the argument declared in a remote package
PLS_TYPE	VARCHAR(128)	Name of the type of the argument at PSM
CHAR_LENGTH	NUMBER	Character limit for string datatypes
CHAR_USED	VARCHAR(1)	Whether the byte limit(B) or char limit(C) is official for the string
ORIGIN_CON_ID	VARCHAR(256)	ID of the container where the data originates

ALL_CATALOG

ALL_CATALOG displays the tables, views, synonyms, and sequences accessible to the current user.

Table 9-3 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the TABLE, VIEW, SYNONYM, SEQUENCE, or UNDEFINED
TABLE_SCHEMA	VARCHAR(128)	Schema of the TABLE, VIEW, SYNONYM, SEQUENCE, or UNDEFINED
TABLE_NAME	VARCHAR(128)	Name of the TABLE, VIEW, SYNONYM, SEQUENCE, or UNDEFINED
TABLE_TYPE	VARCHAR(32)	Type of the TABLE, VIEW, SYNONYM, SEQUENCE, or UNDEFINED

ALL_CLUSTER_TABLES

ALL_CLUSTER_TABLES describes all cluster tables accessible to the current user in the cluster system.

It is available only on a cluster.

Table 9-4 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the table
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
SHARD_STRATEGY	VARCHAR(32)	Sharding strategy of the table: the value in (CLONED, HASH SHARDING, RANGE SHARDING, LIST SHARDING)
SHARD_PLACEMENT	VARCHAR(32)	Shard placement of the table: the value in (AT CLUSTER WIDE or AT CLUSTER GROUP)
SHARD_COUNT	NUMBER	Shard count of the table (if cloned table, the value is null)
SHARD_KEY_COUNT	NUMBER	Shard key column count of the table (if cloned table, the value is null)
HAS_GSI	VARCHAR(3)	Indicate whether the table has global secondary index: (YES) or (NO)
DROPPED	VARCHAR(3)	Indicates whether the table has been dropped and is in the recycle bin (YES) or not (NO)

ALL_COL_COMMENTS

ALL_COL_COMMENTS displays comments on the columns of the tables and views accessible to the curre nt user.

Table 9-5 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Name of the column
COMMENTS	VARCHAR(1024)	Comment on the column

ALL_COL_PRIVS

ALL_COL_PRIVS describes the object grants, for which the current user is the object owner, grantor, or gr antee, or for which an enabled role or PUBLIC is the grantee.

Table 9-6 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Name of the column
PRIVILEGE	VARCHAR(32)	Privilege on the column
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTIO N (YES) or not (NO)

ALL_COL_PRIVS_MADE

ALL_COL_PRIVS_MADE describes the column object grants for which the current user is the object owner or grantor.

Table 9-7 Column information

Column name	Data type	Description
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Name of the column
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
PRIVILEGE	VARCHAR(32)	Privilege on the column
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_COL_PRIVS_RECD

ALL_COL_PRIVS_RECD describes the column object grants, for which the current user is the grantee, or f or which an enabled role or PUBLIC is the grantee.

Table 9-8 Column information

Column name	Data type	Description
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Name of the column
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
PRIVILEGE	VARCHAR(32)	Privilege on the column
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTIO N (YES) or not (NO)

ALL_CONSTRAINTS

ALL_CONSTRAINTS describes constraint definitions on tables accessible to the current user.

Table 9-9 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the constraint definition
CONSTRAINT_SCHEMA	VARCHAR(128)	Schema of the constraint definition
CONSTRAINT_NAME	VARCHAR(128)	Name of the constraint definition
CONSTRAINT_TYPE	VARCHAR(1)	Type of the constraint definition: the value in (C: check constraint, P: Primary key, U: Unique Key, R: Referential intgrity)
TABLE_OWNER	VARCHAR(128)	Owner of the table (or view) associated with the constraint definition
TABLE_SCHEMA	VARCHAR(128)	Schema of the table (or view) associated with the constraint definition
TABLE_NAME	VARCHAR(128)	Name of the table (or view) associated with the constraint definition
SEARCH_CONDITION	LONG VARCHAR	Text of search condition for a check constraint
R_OWNER	VARCHAR(128)	Owner of the unique constraint definition for the referenced table
R_SCHEMA	VARCHAR(128)	Schema of the unique constraint definition for the referenced table
R_CONSTRAINT_NAME	VARCHAR(128)	Name of the unique constraint definition for the referenced table
DELETE_RULE	VARCHAR(32)	Delete rule for a referential constraint: the value in (NO ACTION, RESTRICT, CASCADE, SET NULL, SET DEFAULT)
UPDATE_RULE	VARCHAR(32)	Update rule for a referential constraint: the value in (NO ACTION, RESTRICT, CASCADE, SET NULL, SET DEFAULT)
STATUS	VARCHAR(32)	Enforcement status of the constraint: the value in (ENABLED, DISABLE)
DEFERRABLE	VARCHAR(32)	Indicates whether the constraint is deferrable (DEFERRABLE) or not (NOT DEFERRABLE)
DEFERRED	VARCHAR(32)	Indicates whether the constraint was initially deferred (DEFERRED) or not (IMMEDIATE)
VALIDATED	VARCHAR(32)	Indicates whether all data may obey the constraint or not: the value in (VALIDATED, NOT VALIDATED)
GENERATED	VARCHAR(32)	Indicates whether the name of the constraint is user-generated (USER NAME) or system-generated (GENERATED NAME)
BAD	VARCHAR(32)	• reserved Indicates whether this constraint specifies a century in an ambiguous manner (BAD) or not (NULL)
RELY	VARCHAR(32)	• reserved When NOT VALIDATED, indicates whether the constraint is to be taken in to account for query rewrite (RELY) or not (NULL)
LAST_CHANGE	TIMESTAMP(6) WITHOUT TIME ZONE	When the constraint was last enabled or disabled

Column name	Data type	Description
INDEX_OWNER	VARCHAR(128)	Owner of the index associated with the key constraint
INDEX_SCHEMA	VARCHAR(128)	Schema of the index associated with the key constraint
INDEX_NAME	VARCHAR(128)	Name of the index associated with the key constraint
INVALID	VARCHAR(32)	Indicates whether the constraint is invalid (INVALID) or not (NULL)
VIEW_RELATED	VARCHAR(32)	Indicates whether the constraint depends on a view (DEPEND ON VIEW) or not (NULL)
DROPPED	VARCHAR(3)	Indicates whether the constraint has been dropped and is in the recycle bi n (YES) or not (NO)
COMMENTS	VARCHAR(1024)	Comments of the constraint definition

ALL_CONS_COLUMNS

ALL_CONS_COLUMNS describes columns that are accessible to the current user and that are specified in constraints.

Table 9-10 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the constraint definition
CONSTRAINT_SCHEMA	VARCHAR(128)	Schema of the constraint definition
CONSTRAINT_NAME	VARCHAR(128)	Name of the constraint definition
TABLE_OWNER	VARCHAR(128)	Owner of the table with the constraint definition
TABLE_SCHEMA	VARCHAR(128)	Schema of the table with the constraint definition
TABLE_NAME	VARCHAR(128)	Name of the table with the constraint definition
COLUMN_NAME	VARCHAR(128)	Name of the column or attribute of the object type column specified in the constraint definition
POSITION	NUMBER	Original position of the column or attribute in the definition of the object

ALL_DB_PRIVS

ALL_DB_PRIVS describes the database grants, for which the current user is the grantor, or grantee, or for which an enabled role or PUBLIC is the grantee.

Table 9-11 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
PRIVILEGE	VARCHAR(32)	Privilege on the database
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_DB_PRIVS_MADE

ALL_DB_PRIVS_MADE describes the database grants for which the current user is the grantor.

Table 9-12 Column information

Column name	Data type	Description
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
PRIVILEGE	VARCHAR(32)	Privilege on the database
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_DB_PRIVS_RECD

ALL_DB_PRIVS_RECD describes the database grants, for which the current user is the grantee, or for whic h an enabled role or PUBLIC is the grantee.

Table 9-13 Column information

Column name	Data type	Description
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
PRIVILEGE	VARCHAR(32)	Privilege on the database
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTIO N (YES) or not (NO)

ALL_DEPENDENCIES

ALL_DEPENDENCIES describes dependencies between objects accessible to the current user

Table 9-14 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of object
SCHEMA_NAME	VARCHAR(128)	Schema Name of object
NAME	VARCHAR(128)	Name of object
TYPE	VARCHAR(32)	Type of object: FUNCTION, PROCEDURE, VIEW, PACKAGE, PACKAGE BODY, TRIGGER
REFERENCED_OWNER	VARCHAR(128)	Owner of the referenced object
REFERENCED_SCHEMA_NAME	VARCHAR(128)	Schema Name of the referenced object
REFERENCED_TYPE	VARCHAR(32)	Type of the referenced object: FUNCTION, PROCEDURE, TABLE, VIEW, SEQUENCE, PACKAGE, PACKAGE BODY, TRIGGER
REFERENCED_LINK_NAME	VARCHAR(128)	Name of the link to the parent object
REFERENCED_NAME	VARCHAR(128)	Name of the referenced object
DEPENDENCY_TYPE	VARCHAR(32)	Indicates whether the dependency is a REF dependency (REF) or not (HARD)

ALL_GLOBAL_SECONDARY_INDEXES

ALL_GLOBAL_SECONDARY_INDEXES describes the global secondary indexes on the tables accessible to t he current user.


It is available only on a cluster.

Table 9-15 Column information

Column name	Data type	Description
TABLE_OWNER	VARCHAR(128)	Owner of the global secondary indexed object
TABLE_SCHEMA	VARCHAR(128)	Schema of the global secondary indexed object
TABLE_NAME	VARCHAR(128)	Name of the global secondary indexed object
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the global secondary index
INI_TRANS	NUMBER	Initial number of transactions
MAX_TRANS	NUMBER	Maximum number of transactions
INITIAL_EXTENT	NUMBER	Size of the initial extent
NEXT_EXTENT	NUMBER	Size of secondary extents
MIN_EXTENTS	NUMBER	Minimum number of extents allowed in the segment
MAX_EXTENTS	NUMBER	Maximum number of extents allowed in the segment
PCT_FREE	NUMBER	Minimum percentage of free space in a block
LOGGING	VARCHAR(3)	Indicates whether or not changes to the global secondary index are l ogged: (YES) or (NO)
BLOCKS	NUMBER	Number of used blocks in the global secondary index
EMPTY_BLOCKS	NUMBER	Number of empty blocks in the global secondary index
DROPPED	VARCHAR(3)	Indicates whether the global secondary index has been dropped and is in the recycle bin (YES) or not (NO)

ALL_GSI_PLACE

ALL_GSI_PLACE describes node placement of all global secondary indexes on the tables accessible to the current user in the cluster system.


It is available only on a cluster.

Table 9-16 Column information

Column name	Data type	Description
TABLE_OWNER	VARCHAR(128)	Owner of the global secondary indexed object
TABLE_SCHEMA	VARCHAR(128)	Schema of the global secondary indexed object
TABLE_NAME	VARCHAR(128)	Name of the global secondary indexed object
GROUP_ID	NUMBER	Group identifier of the node where the global secondary index placed
GROUP_NAME	VARCHAR(128)	Group name of the node where the global secondary index placed
MEMBER_ID	NUMBER	Member identifier of the node where the global secondary index placed
MEMBER_NAME	VARCHAR(128)	Member name of the node where the global secondary index placed
MEMBER_OFFLINE	BOOLEAN	data of the cluster member is offline or not
DROPPED	VARCHAR(3)	Indicates whether the global secondary index has been dropped and is in the recycle bin (YES) or not (NO)
BLOCKS	NUMBER	Number of used blocks of the node where the global secondary index placed

ALL_INDEXES

ALL_INDEXES describes the indexes on the tables accessible to the current user.

Table 9-17 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the index
INDEX_SCHEMA	VARCHAR(128)	Schema of the index
INDEX_NAME	VARCHAR(128)	Name of the index
INDEX_TYPE	VARCHAR(32)	Type of the index: the value in (NORMAL, NORMAL/REV, BITMAP, FUNCTION-BASED NORMAL, FUNCTION-BASED NORMAL/REV, FUNCTION-BASED BITMAP, IOT - TOP, DOMAIN)
TABLE_OWNER	VARCHAR(128)	Owner of the indexed object
TABLE_SCHEMA	VARCHAR(128)	Schema of the indexed object
TABLE_NAME	VARCHAR(128)	Name of the indexed object
TABLE_TYPE	VARCHAR(32)	Type of the indexed object: the value in (NEXT OBJECT, INDEX, TABLE, VIEW, SYNONYM, SEQUENCE)
UNIQUENESS	VARCHAR(32)	Indicates whether the index is unique (UNIQUE) or nonunique (NONUNIQUE)
COMPRESSION	VARCHAR(32)	• reserved Indicates whether index compression is enabled (ENABLED) or not (DISABLED)
PREFIX_LENGTH	NUMBER	• reserved Number of columns in the prefix of the compression key
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the index
INI_TRANS	NUMBER	Initial number of transactions
MAX_TRANS	NUMBER	Maximum number of transactions
INITIAL_EXTENT	NUMBER	Size of the initial extent
NEXT_EXTENT	NUMBER	Size of secondary extents
MIN_EXTENTS	NUMBER	Minimum number of extents allowed in the segment
MAX_EXTENTS	NUMBER	Maximum number of extents allowed in the segment
PCT_INCREASE	NUMBER	• reserved Percentage increase in extent size
PCT_THRESHOLD	NUMBER	• reserved Threshold percentage of block space allowed per index entry
INCLUDE_COLUMN	NUMBER	• reserved Column ID of the last column to be included in index-organized table primary key (non-overflow) index
FREELISTS	NUMBER	• reserved Number of process freelists allocated to this segment
FREELIST_GROUPS	NUMBER	• reserved Number of freelist groups allocated to this segment

Column name	Data type	Description
PCT_FREE	NUMBER	Minimum percentage of free space in a block
LOGGING	VARCHAR(3)	Indicates whether or not changes to the index are logged: (YES) or (NO)
BLOCKS	NUMBER	Number of used blocks in the index
ANAL_BLOCKS	NUMBER	Number of used blocks in the index when most recently analyzed
EMPTY_BLOCKS	NUMBER	Number of empty blocks in the index
BLEVEL	NUMBER	B-Tree level (depth of the index from its root block to its leaf blocks)
LEAF_BLOCKS	NUMBER	Number of leaf blocks in the index
DISTINCT_KEYS	NUMBER	Number of distinct indexed values.
AVG_LEAF_BLOCKS_PER_KEY	NUMBER	• reserved Average number of leaf blocks in which each distinct value in the in dex appears, rounded to the nearest integer
AVG_DATA_BLOCKS_PER_KEY	NUMBER	• reserved Average number of data blocks in the table that are pointed to by a distinct value in the index rounded to the nearest integer
CLUSTERING_FACTOR	NUMBER	Indicates the amount of order of the rows in the table based on the values of the index
STATUS	VARCHAR(32)	Indicates whether a nonpartitioned index is VALID or UNUSABLE
NUM_ROWS	NUMBER	• reserved Number of rows in the index
SAMPLE_SIZE	NUMBER	Size of the sample used to analyze the index
LAST_ANALYZED	TIMESTAMP(6) WITH OUT TIME ZONE	Date on which this index was most recently analyzed
DEGREE	VARCHAR(32)	• reserved Number of threads per instance for scanning the index, or DEFAULT
INSTANCES	VARCHAR(32)	• reserved Number of instances across which the indexes to be scanned, or DE FAULT
PARTITIONED	VARCHAR(3)	• reserved Indicates whether the index is partitioned (YES) or not (NO)
TEMPORARY	VARCHAR(1)	Indicates whether the index is on a temporary table (Y) or not (N)
GENERATED	VARCHAR(1)	Indicates whether the name of the index is system-generated (Y) or not (N)
SECONDARY	VARCHAR(1)	Indicates whether the index is a secondary object created by the me thod of the Data Cartridge (Y) or not (N)
BUFFER_POOL	VARCHAR(32)	• reserved Buffer pool to be used for index blocks
FLASH_CACHE	VARCHAR(32)	• reserved Database Smart Flash Cache hint to be used for index blocks
CELL_FLASH_CACHE	VARCHAR(32)	• reserved Cell flash cache hint to be used for index blocks

Column name	Data type	Description
USER_STATS	VARCHAR(3)	• reserved Indicates whether statistics were entered directly by the user (YES) o r not (NO)
DURATION	VARCHAR(32)	Indicates the duration of a temporary table, the value is in (TRANSA CTION, SESSION)
PCT_DIRECT_ACCES S	NUMBER	• reserved For a secondary index on an index-organized table, the percentage of rows with VALID guess
ITYP_OWNER	VARCHAR(128)	• reserved For a domain index, the owner of the indextype
ITYP_NAME	VARCHAR(128)	• reserved For a domain index, the name of the indextype
PARAMETERS	VARCHAR(1024)	• reserved For a domain index, the parameter string
GLOBAL_STATS	VARCHAR(3)	• reserved For partitioned indexes, indicates whether statistics were collected b y analyzing the index as a whole (YES) or were estimated from statis tics on underlying index partitions and subpartitions (NO)
DOMIDX_STATUS	VARCHAR(32)	• reserved Status of a domain index
DOMIDX_OPSTATUS	VARCHAR(32)	• reserved Status of the operation on a domain index
FUNCIDX_STATUS	VARCHAR(32)	• reserved Status of a function-based index
JOIN_INDEX	VARCHAR(3)	• reserved Indicates whether the index is a join index (YES) or not (NO)
IOT_REDUNDANT_P KEY_ELIM	VARCHAR(3)	• reserved Indicates whether redundant primary key columns are eliminated fro m secondary indexes on index-organized tables (YES) or not (NO)
DROPPED	VARCHAR(3)	Indicates whether the index has been dropped and is in the recycle bin (YES) or not (NO)
VISIBILITY	VARCHAR(3)	• reserved Indicates whether the index is VISIBLE or INVISIBLE to the optimizer
DOMIDX_MANAGE MENT	VARCHAR(32)	• reserved If this is a domain index, indicates whether the domain index is syste m-managed (SYSTEM_MANAGED) or user-managed (USER_MANA GED)
SEGMENT_CREATED	VARCHAR(3)	Indicates whether the index segment has been created (YES) or not (NO)
COMMENTS	VARCHAR(1024)	Comments of the index
EMPTY_BLOCKS	NUMBER	Number of empty blocks in the index

ALL_IND_COLUMNS

ALL_IND_COLUMNS describes the columns of indexes on all tables accessible to the current user.

Table 9-18 Column information

Column name	Data type	Description
INDEX_OWNER	VARCHAR(128)	Owner of the index
INDEX_SCHEMA	VARCHAR(128)	Schema of the index
INDEX_NAME	VARCHAR(128)	Name of the index
TABLE_OWNER	VARCHAR(128)	Owner of the table or cluster
TABLE_SCHEMA	VARCHAR(128)	Schema of the table or cluster
TABLE_NAME	VARCHAR(128)	Name of the table or cluster
COLUMN_NAME	VARCHAR(128)	Column name or attribute of the object type column
COLUMN_POSITION	NUMBER	Position of the column or attribute within the index
COLUMN_LENGTH	NUMBER	Indexed length of the column
CHAR_LENGTH	NUMBER	Maximum codepoint length of the column
DESCEND	VARCHAR(32)	Indicates whether the column is sorted in descending order (DESC) or ascending order (ASC)
NULL_ORDER	VARCHAR(32)	Indicates whether the null value of the column is sorted in nulls first order (NULLS FIRST) or nulls last order (NULLS LAST)

ALL_IND_PLACE

ALL_IND_PLACE describes node placement of the indexes on the tables accessible to the current user in t he cluster system.


It is available only on a cluster.

Table 9-19 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the index
INDEX_SCHEMA	VARCHAR(128)	Schema of the index
INDEX_NAME	VARCHAR(128)	Name of the index
TABLE_OWNER	VARCHAR(128)	Owner of the indexed object
TABLE_SCHEMA	VARCHAR(128)	Schema of the indexed object
TABLE_NAME	VARCHAR(128)	Name of the indexed object
GROUP_ID	NUMBER	Group identifier of the node where the index placed
GROUP_NAME	VARCHAR(128)	Group name of the node where the index placed
MEMBER_ID	NUMBER	Member identifier of the node where the index placed
MEMBER_NAME	VARCHAR(128)	Member name of the node where the index placed
MEMBER_OFFLINE	BOOLEAN	data of the cluster member is offline or not
DROPPED	VARCHAR(3)	Indicates whether the index has been dropped and is in the recycle bin (YES) or not (NO)
DISTINCT_KEYS	NUMBER	(deprecated)
SAMPLE_SIZE	NUMBER	(deprecated)
BLOCKS	NUMBER	Number of used blocks of the node where the index placed
LAST_ANALYZED	TIMESTAMP(6) WITH OUT TIME ZONE	(deprecated)

ALL_NONSCHEMA_COMMENTS

ALL_NONSCHEMA_COMMENTS displays comments on all non-schema objects (database, authorizations, schemas, tablespaces) accessible to the current user.

Table 9-20 Column information

Column name	Data type	Description
OBJECT_NAME	VARCHAR(128)	Name of the non-schema object
OBJECT_TYPE	VARCHAR(32)	Type of the non-schema object: DATABASE, AUTHORIZATION, SCHEMA, TABLESPACE
COMMENTS	VARCHAR(1024)	Comments of the non-schema object

ALL_OBJECTS

ALL_OBJECTS describes all objects accessible to the current user.

Table 9-21 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the object
SCHEMA_NAME	VARCHAR(128)	Schema of the object
OBJECT_NAME	VARCHAR(128)	Name of the object
SUBOBJECT_NAME	VARCHAR(128)	Name of the subobject (for example, partition)
OBJECT_ID	NUMBER	Dictionary object number of the object
DATA_OBJECT_ID	NUMBER	Dictionary object number of the segment that contains the object
OBJECT_TYPE	VARCHAR(32)	Type of the object (such as TABLE, INDEX)
CREATED	TIMESTAMP(6) WITHOUT TIME ZONE	Timestamp for the creation of the object
LAST_DDL_TIME	TIMESTAMP(6) WITHOUT TIME ZONE	Timestamp for the last modification of the object resulting from a DDL statement
TIMESTAMP	VARCHAR(32)	Timestamp for the specification of the object (character data)
STATUS	VARCHAR(32)	Status of the object: the value in (VALID, INVALID, N/A)
TEMPORARY	VARCHAR(1)	Indicates whether the object is temporary (the current session can see only data that it placed in this object itself) (Y) or not (N)
GENERATED	VARCHAR(1)	• reserved Indicates whether the name of this object was system-generated (Y) or not (N)
SECONDARY	VARCHAR(1)	• reserved Indicates whether this is a secondary object created by the ODCIIndexCreate method of the Oracle Data Cartridge (Y) or not (N)
NAMESPACE	NUMBER	Namespace for the object
EDITION_NAME	VARCHAR(128)	• reserved Name of the edition in which the object is actual
DROPPED	VARCHAR(3)	Indicates whether the object has been dropped and is in the recycle bin (YES) or not (NO)

ALL_PACKAGE_PRIVS

ALL_PACKAGE_PRIVS describes the package grants, for which the current user is the package owner, gra ntor, or grantee, or for which an enabled role or PUBLIC is the grantee.

Table 9-22 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
PROCEDURE_OWNER	VARCHAR(128)	Owner of the package
PROCEDURE_SCHEMA	VARCHAR(128)	Schema of the package
PROCEDURE_NAME	VARCHAR(128)	Name of the package
PRIVILEGE	VARCHAR(32)	Privilege on the package
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTIO N (YES) or not (NO)

ALL_PACKAGE_PRIVS_MADE

ALL_PACKAGE_PRIVS_MADE describes the package grants for which the current user is the package owner or grantor.

Table 9-23 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
PROCEDURE_OWNER	VARCHAR(128)	Owner of the package
PROCEDURE_SCHEMA	VARCHAR(128)	Schema of the package
PROCEDURE_NAME	VARCHAR(128)	Name of the package
PRIVILEGE	VARCHAR(32)	Privilege on the package
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_PACKAGE_PRIVS_RECD

ALL_PACKAGE_PRIVS_RECD describes the package grants, for which the current user is the grantee, or f or which an enabled role or PUBLIC is the grantee.

Table 9-24 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
PROCEDURE_OWNER	VARCHAR(128)	Owner of the package
PROCEDURE_SCHEMA	VARCHAR(128)	Schema of the package
PROCEDURE_NAME	VARCHAR(128)	Name of the package
PRIVILEGE	VARCHAR(32)	Privilege on the package
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTIO N (YES) or not (NO)

ALL_PROCEDURES

ALL_PROCEDURES lists all function, procedures or pacakage

Table 9-25 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of function, procedures or pacakage
SCHEMA_NAME	VARCHAR(128)	Schema Name of function, procedures or package
OBJECT_NAME	VARCHAR(128)	Name of function, procedures or pacakage
PROCEDURE_NAME	VARCHAR(128)	Name when a procedures in pacakage
OBJECT_ID	NUMBER	ID of a function, procedures or pacakage
SUBPROGRAM_ID	NUMBER	ID of procedures in pacakage
OVERLOAD	VARCHAR(32)	ID of overloading procedure in pacakage
OBJECT_TYPE	VARCHAR(32)	Type of function, procedures or package
AGGREGATE	VARCHAR(3)	Indicate whether the procedure is an aggreage function(YES) or not (NO)
PIPELINED	VARCHAR(3)	Indicate whether the procedure is a pipelined table function(YES) or not(NO)
IMPLTYPEOWNER	VARCHAR(128)	Name of the owner of the implementation type, if any
IMPLTYPENAME	VARCHAR(128)	Name of the implementation type, if any
PARALLEL	VARCHAR(3)	Indicates whether the procedure or function is parallel-enabled (YES) or not (NO)
INTERFACE	VARCHAR(3)	YES, if the procedure/function is a table function implemented usin g the SQLCLI interface; otherwise NO
DETERMINISTIC	VARCHAR(3)	YES, if the procedure/function is declared to be deterministic; other wise NO
AUTHID	VARCHAR(32)	Indicates whether the procedure/function is declared to execute as DEFINER or CURRENT_USER (invoker)

ALL_PROC_PRIVS

ALL_PROC_PRIVS describes the procedure grants, for which the current user is the procedure owner, gra ntor, or grantee, or for which an enabled role or PUBLIC is the grantee.

Table 9-26 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
PROCEDURE_OWNER	VARCHAR(128)	Owner of the procedure and function
PROCEDURE_SCHEMA	VARCHAR(128)	Schema of the procedure and function
PROCEDURE_NAME	VARCHAR(128)	Name of the procedure and function
PRIVILEGE	VARCHAR(32)	Privilege on the procedure and function
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTIO N (YES) or not (NO)

ALL_PROC_PRIVS_MADE

ALL_PROC_PRIVS_MADE describes the procedure grants for which the current user is the procedure owner or grantor.

Table 9-27 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
PROCEDURE_OWNER	VARCHAR(128)	Owner of the procedure and function
PROCEDURE_SCHEMA	VARCHAR(128)	Schema of the procedure and function
PROCEDURE_NAME	VARCHAR(128)	Name of the procedure and function
PRIVILEGE	VARCHAR(32)	Privilege on the procedure and function
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_PROC_PRIVS_RECD

ALL_PROC_PRIVS_RECD describes the procedure grants, for which the current user is the grantee, or for which an enabled role or PUBLIC is the grantee.

Table 9-28 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
PROCEDURE_OWNER	VARCHAR(128)	Owner of the procedure and function
PROCEDURE_SCHEMA	VARCHAR(128)	Schema of the procedure and function
PROCEDURE_NAME	VARCHAR(128)	Name of the procedure and function
PRIVILEGE	VARCHAR(32)	Privilege on the procedure and function
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_SCHEMAS

Identify the schemata in a catalog that are owned by given user or accessible to given user or role.

Table 9-29 Column information

Column name	Data type	Description
SCHEMA_OWNER	VARCHAR(128)	Owner of the schema
SCHEMA_NAME	VARCHAR(128)	Name of the schema
CREATED_TIME	TIMESTAMP(6) WITHOUT TIME ZONE	Created time of the schema
MODIFIED_TIME	TIMESTAMP(6) WITHOUT TIME ZONE	Last modified time of the schema
COMMENTS	VARCHAR(1024)	Comments of the schema

ALL_SCHEMA_PATH

ALL_SCHEMA_PATH describes the schema search order of the current user and PUBLIC, for naming resol ution of unqualified SQL schema objects.

Table 9-30 Column information

Column name	Data type	Description
AUTH_NAME	VARCHAR(128)	Name of the authorization
SCHEMA_NAME	VARCHAR(128)	Name of the schema
SEARCH_ORDER	NUMBER	Schema search order of the authorization

ALL_SCHEMA_PRIVS

ALL_SCHEMA_PRIVS describes the schema grants, for which the current user is the schema owner, grant or, or grantee, or for which an enabled role or PUBLIC is the grantee.

Table 9-31 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the schema
SCHEMA_NAME	VARCHAR(128)	Name of the schema
PRIVILEGE	VARCHAR(32)	Privilege on the schema
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_SCHEMA_PRIVS_MADE

ALL_SCHEMA_PRIVS_MADE describes the schema grants, for which the current user is the grantor.

Table 9-32 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the schema
SCHEMA_NAME	VARCHAR(128)	Name of the schema
PRIVILEGE	VARCHAR(32)	Privilege on the schema
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_SCHEMA_PRIVS_RECD

ALL_SCHEMA_PRIVS_RECD describes the schema grants, for which the current user is the grantee, or for which an enabled role or PUBLIC is the grantee.

Table 9-33 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the schema
SCHEMA_NAME	VARCHAR(128)	Name of the schema
PRIVILEGE	VARCHAR(32)	Privilege on the schema
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_SEQUENCES

ALL_SEQUENCES describes all sequences accessible to the current user.

Table 9-34 Column information

Column name	Data type	Description
SEQUENCE_OWNER	VARCHAR(128)	Owner of the sequence
SEQUENCE_SCHEMA	VARCHAR(128)	Schema of the sequence
SEQUENCE_NAME	VARCHAR(128)	Sequence name
MIN_VALUE	NUMBER	Minimum value of the sequence
MAX_VALUE	NUMBER	Maximum value of the sequence
INCREMENT_BY	NUMBER	Value by which sequence is incremented
CYCLE_FLAG	VARCHAR(1)	Indicates whether the sequence wraps around on reaching the limit (Y) or not (N)
ORDER_FLAG	VARCHAR(1)	Indicates whether sequence numbers are generated in order (Y) or not (N)
CACHE_SIZE	NUMBER	Number of sequence numbers to cache
LAST_NUMBER	NUMBER	Last sequence number written to database. If a sequence uses caching, the number written to database is the last number placed in the sequence cache.
COMMENTS	VARCHAR(1024)	Comments of the sequence

ALL_SEQ_PRIVS

ALL_SEQ_PRIVS describes the sequence grants, for which the current user is the sequence owner, grantor, or grantee, or for which an enabled role or PUBLIC is the grantee.

Table 9-35 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
SEQUENCE_OWNER	VARCHAR(128)	Owner of the sequence
SEQUENCE_SCHEMA	VARCHAR(128)	Schema of the sequence
SEQUENCE_NAME	VARCHAR(128)	Name of the sequence
PRIVILEGE	VARCHAR(32)	Privilege on the sequence
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_SEQ_PRIVS_MADE

ALL_SEQ_PRIVS_MADE describes the sequence grants for which the current user is the sequence owner or grantor.

Table 9-36 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
SEQUENCE_OWNER	VARCHAR(128)	Owner of the sequence
SEQUENCE_SCHEMA	VARCHAR(128)	Schema of the sequence
SEQUENCE_NAME	VARCHAR(128)	Name of the sequence
PRIVILEGE	VARCHAR(32)	Privilege on the sequence
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_SEQ_PRIVS_RECD

ALL_SEQ_PRIVS_RECD describes the sequence grants, for which the current user is the grantee, or for wh ich an enabled role or PUBLIC is the grantee.

Table 9-37 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
SEQUENCE_OWNER	VARCHAR(128)	Owner of the sequence
SEQUENCE_SCHEM A	VARCHAR(128)	Schema of the sequence
SEQUENCE_NAME	VARCHAR(128)	Name of the sequence
PRIVILEGE	VARCHAR(32)	Privilege on the sequence
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTIO N (YES) or not (NO)

ALL_SHARD_KEY_COLUMNS

ALL_SHARD_KEY_COLUMNS describes shard key columns of all shareded tables accessible to the current user in the cluster system.


It is available only on a cluster.

Table 9-38 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the table
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
COLUMN_NAME	VARCHAR(128)	Column name of the shard key
COLUMN_POSITION	NUMBER	Position of the column within the shard key

ALL_SOURCE

ALL_SOURCE describes the text source of the stored objects accessible to the current user.

Table 9-39 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of object
SCHEMA_NAME	VARCHAR(128)	Schema Name of object
NAME	VARCHAR(128)	Name of object
TYPE	VARCHAR(32)	Type of object: FUNCTION, PROCEDURE, PACKAGE, PACKAGE BODY, TRIGGER
LINE	NUMBER	Line number of this line of source
TEXT	LONG VARCHAR	Text source of the strored object
ORIGIN_CON_ID	VARCHAR(256)	ID of the container where the data originates

ALL_SYNONYMS

ALL_SYNONYMS describes all synonyms.

Table 9-40 Column information

Column name	Data type	Description
SYNONYM_OWNER	VARCHAR(128)	Owner of the synonym
SYNONYM_SCHEMA	VARCHAR(128)	Schema of the synonym
SYNONYM_NAME	VARCHAR(128)	Synonym name
OBJECT_SCHEMA_NAME	VARCHAR(128)	Object schema name
OBJECT_NAME	VARCHAR(128)	Object name
DB_LINK	VARCHAR(128)	Reserved for future use

ALL_TABLES

ALL_TABLES describes the relational tables accessible to the current user.

Table 9-41 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the table
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the table
CLUSTER_NAME	VARCHAR(128)	• reserved Name of the cluster
IOT_NAME	VARCHAR(128)	• reserved Name of the index-organized table
STATUS	VARCHAR(32)	• reserved If a previous DROP TABLE operation failed, indicates whether the table is unusable (UNUSABLE) or valid (VALID)
PCT_FREE	NUMBER	Minimum percentage of free space in a block
PCT_USED	NUMBER	Minimum percentage of used space in a block
INI_TRANS	NUMBER	Initial number of transactions
MAX_TRANS	NUMBER	Maximum number of transactions
INITIAL_EXTENT	NUMBER	Size of the initial extent (in bytes)
NEXT_EXTENT	NUMBER	Size of secondary extents (in bytes)
MIN_EXTENTS	NUMBER	Minimum number of extents allowed in the segment
MAX_EXTENTS	NUMBER	Maximum number of extents allowed in the segment
PCT_INCREASE	NUMBER	• reserved Percentage increase in extent size
FREELISTS	NUMBER	• reserved Number of process freelists allocated to the segment
FREELIST_GROUPS	NUMBER	• reserved Number of freelist groups allocated to the segment
LOGGING	VARCHAR(3)	Indicates whether or not changes to the table are logged
BACKED_UP	VARCHAR(1)	• reserved Indicates whether the table has been backed up since the last modification (Y) or not (N)
NUM_ROWS	NUMBER	Number of rows in the table
BLOCKS	NUMBER	Number of used blocks in the table
ANAL_BLOCKS	NUMBER	Number of used blocks in the table when most recently analyzed
EMPTY_BLOCKS	NUMBER	• reserved Number of empty (never used) blocks in the table
AVG_SPACE	NUMBER	• reserved Average available free space in the table

Column name	Data type	Description
CHAIN_CNT	NUMBER	• reserved Number of rows in the table that are chained from one data block to another or that have migrated to a new block, requiring a link to preserve the old rowid
AVG_ROW_LEN	NUMBER	• reserved Average row length, including row overhead
AVG_SPACE_FREELIST_BLOCKS	NUMBER	• reserved Average freespace of all blocks on a freelist
NUM_FREELIST_BLOCKS	NUMBER	• reserved Number of blocks on the freelist
DEGREE	VARCHAR(32)	• reserved Number of threads per instance for scanning the table, or DEFAULT
INSTANCES	VARCHAR(32)	• reserved Number of instances across which the table is to be scanned, or DEFAULT
CACHE	VARCHAR(1)	• reserved Indicates whether the table is to be cached in the buffer cache (Y) or not (N)
TABLE_LOCK	VARCHAR(32)	Indicates whether table locking is enabled (ENABLED) or disabled (DISABLED)
SAMPLE_SIZE	NUMBER	Sample size used in analyzing the table
LAST_ANALYZED	TIMESTAMP(6) WITHOUT TIME ZONE	Date on which the table was most recently analyzed
PARTITIONED	VARCHAR(3)	• reserved Indicates whether the table is partitioned (YES) or not (NO)
IOT_TYPE	VARCHAR(32)	• reserved If the table is an index-organized table, then IOT_TYPE is IOT, IOT_OVERFLOW, or IOT_MAPPING.
TEMPORARY	VARCHAR(1)	Indicates whether the table is temporary (Y) or not (N)
SECONDARY	VARCHAR(1)	• reserved Indicates whether the table is a secondary object created by cartridge
NESTED	VARCHAR(3)	• reserved Indicates whether the table is a nested table (YES) or not (NO)
BUFFER_POOL	VARCHAR(32)	• reserved Buffer pool to be used for table blocks
FLASH_CACHE	VARCHAR(32)	• reserved Database Smart Flash Cache hint to be used for table blocks
CELL_FLASH_CACHE	VARCHAR(32)	• reserved Cell flash cache hint to be used for table blocks
		• reserved

Column name	Data type	Description
ROW_MOVEMENT	VARCHAR(32)	If a partitioned table, indicates whether row movement is enabled (ENABLED) or disabled (DISABLED)
GLOBAL_STATS	VARCHAR(3)	• reserved For partitioned tables, indicates whether statistics for the table as a whole (global statistics) are accurate (YES)
USER_STATS	VARCHAR(3)	• reserved Indicates whether statistics were entered directly by the user (YES) o r not (NO)
DURATION	VARCHAR(32)	Indicates the duration of a temporary table, the value is in (TRANSA CTION, SESSION)
SKIP_CORRUPT	VARCHAR(32)	• reserved Indicates whether Database ignores blocks marked corrupt during ta ble and index scans (ENABLED) or raises an error (DISABLED)
MONITORING	VARCHAR(3)	• reserved Indicates whether the table has the MONITORING attribute set (YES) or not (NO)
CLUSTER_OWNER	VARCHAR(128)	• reserved Owner of the cluster, if any
DEPENDENCIES	VARCHAR(32)	• reserved Indicates whether row-level dependency tracking is enabled (ENABL ED) or disabled (DISABLED)
COMPRESSION	VARCHAR(32)	• reserved Indicates whether table compression is enabled (ENABLED) or not (DISABLED)
COMPRESS_FOR	VARCHAR(32)	• reserved Default compression for what kind of operations
DROPPED	VARCHAR(3)	Indicates whether the table has been dropped and is in the recycle b in (YES) or not (NO)
READ_ONLY	VARCHAR(3)	Indicates whether the table IS READ-ONLY (YES) or not (NO)
SEGMENT_CREATED	VARCHAR(3)	Indicates whether the table segment has been created (YES) or not (NO)
RESULT_CACHE	VARCHAR(32)	• reserved Result cache mode annotation for the table: the value in (NULL, DE FAULT, FORCE, MANUAL)

ALL_TAB_COLS

ALL_TAB_COLS describes the columns(including hidden columns) of the tables, views, and clusters access ible to the current user.

Table 9-42 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Column name
DATA_TYPE	VARCHAR(128)	Datatype of the column
DATA_TYPE_MOD	VARCHAR(3)	• reserved Datatype modifier of the column
DATA_TYPE_OWNER	VARCHAR(128)	• reserved Owner of the datatype of the column
DATA_LENGTH	NUMBER	Length of the column (in bytes)
DATA_PRECISION	NUMBER	Decimal precision for NUMBER datatype; binary precision for FLOAT datatype; NULL for all other datatypes
DATA_SCALE	NUMBER	Digits to the right of the decimal point in a number
NULLABLE	VARCHAR(1)	Indicates whether a column allows NULLs.
COLUMN_ID	NUMBER	Sequence number of the column as created
DEFAULT_LENGTH	NUMBER	Length of the default value for the column
DATA_DEFAULT	LONG VARCHAR	Default value for the column
NUM_DISTINCT	NUMBER	Number of distinct values in the column
LOW_VALUE	VARBINARY(32)	Low value in the column
HIGH_VALUE	VARBINARY(32)	High value in the column
DENSITY	NUMBER	• reserved If a histogram is available on COLUMN_NAME, then this column dis plays the selectivity of a value that spans fewer than 2 endpoints in t he histogram.
NUM_NULLS	NUMBER	Number of NULLs in the column
NUM_BUCKETS	NUMBER	• reserved Number of buckets in the histogram for the column
LAST_ANALYZED	TIMESTAMP(6) WITH OUT TIME ZONE	Date on which this column was most recently analyzed
SAMPLE_SIZE	NUMBER	Sample size used in analyzing this column
CHARACTER_SET_N AME	VARCHAR(128)	• reserved Name of the character set
CHAR_COL_DECL_LE NGTH	NUMBER	• reserved Declaration length of the character type column
		• reserved

Column name	Data type	Description
GLOBAL_STATS	VARCHAR(3)	For partitioned tables, indicates whether column statistics were collected for the table
USER_STATS	VARCHAR(3)	• reserved Indicates whether statistics were entered directly by the user (YES) or not (NO)
AVG_COL_LEN	NUMBER	Average length of the column (in bytes)
CHAR_LENGTH	NUMBER	Displays the length of the column in characters.
CHAR_USED	VARCHAR(1)	Indicates that the column uses BYTE length semantics (B) or CHAR length semantics (C)
V80_FMT_IMAGE	VARCHAR(3)	• reserved Indicates whether the column data is in release older image format (YES) or not (NO)
DATA_UPGRADED	VARCHAR(3)	• reserved Indicates whether the column data has been upgraded to the latest type version format (YES) or not (NO)
HIDDEN_COLUMN	VARCHAR(3)	Indicates whether the column is a hidden column (YES) or not (NO)
VIRTUAL_COLUMN	VARCHAR(3)	• reserved Indicates whether the column is a virtual column (YES) or not (NO)
SEGMENT_COLUMN_ID	NUMBER	Sequence number of the column in the segment
INTERNAL_COLUMN_ID	NUMBER	Internal sequence number of the column
HISTOGRAM	VARCHAR(32)	• reserved Indicates existence/type of histogram
QUALIFIED_COL_NAME	VARCHAR(4000)	Qualified column name
IDENTITY_COLUMN	VARCHAR(3)	Indicates whether this is an identity column (YES) or not (NO)

ALL_TAB_COLUMNS

ALL_TAB_COLUMNS describes the columns of the tables, views, and clusters accessible to the current use r.

Table 9-43 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Column name
DATA_TYPE	VARCHAR(128)	Datatype of the column
DATA_TYPE_MOD	VARCHAR(3)	• reserved Datatype modifier of the column
DATA_TYPE_OWNER	VARCHAR(128)	• reserved Owner of the datatype of the column
DATA_LENGTH	NUMBER	Length of the column (in bytes)
DATA_PRECISION	NUMBER	Decimal precision for NUMBER datatype; binary precision for FLOAT datatype; NULL for all other datatypes
DATA_SCALE	NUMBER	Digits to the right of the decimal point in a number
NULLABLE	VARCHAR(1)	Indicates whether a column allows NULLs.
COLUMN_ID	NUMBER	Sequence number of the column as created
DEFAULT_LENGTH	NUMBER	Length of the default value for the column
DATA_DEFAULT	LONG VARCHAR	Default value for the column
NUM_DISTINCT	NUMBER	Number of distinct values in the column
LOW_VALUE	VARBINARY(32)	Low value in the column
HIGH_VALUE	VARBINARY(32)	High value in the column
DENSITY	NUMBER	• reserved If a histogram is available on COLUMN_NAME, then this column dis plays the selectivity of a value that spans fewer than 2 endpoints in t he histogram.
NUM_NULLS	NUMBER	Number of NULLs in the column
NUM_BUCKETS	NUMBER	• reserved Number of buckets in the histogram for the column
LAST_ANALYZED	TIMESTAMP(6) WITH OUT TIME ZONE	Date on which this column was most recently analyzed
SAMPLE_SIZE	NUMBER	Sample size used in analyzing this column
CHARACTER_SET_N AME	VARCHAR(128)	• reserved Name of the character set
CHAR_COL_DECL_LE NGTH	NUMBER	Declaration length of the character type column
		• reserved

Column name	Data type	Description
GLOBAL_STATS	VARCHAR(3)	For partitioned tables, indicates whether column statistics were collected for the table
USER_STATS	VARCHAR(3)	• reserved Indicates whether statistics were entered directly by the user (YES) or not (NO)
AVG_COL_LEN	NUMBER	Average length of the column (in bytes)
CHAR_LENGTH	NUMBER	Displays the length of the column in characters.
CHAR_USED	VARCHAR(1)	Indicates that the column uses BYTE length semantics (B) or CHAR length semantics (C)
V80_FMT_IMAGE	VARCHAR(3)	• reserved Indicates whether the column data is in release older image format (YES) or not (NO)
DATA_UPGRADED	VARCHAR(3)	• reserved Indicates whether the column data has been upgraded to the latest type version format (YES) or not (NO)
HISTOGRAM	VARCHAR(32)	• reserved Indicates existence/type of histogram
IDENTITY_COLUMN	VARCHAR(3)	Indicates whether this is an identity column (YES) or not (NO)

ALL_TAB_COMMENTS

ALL_TAB_COMMENTS displays comments on the tables and views accessible to the current user.

Table 9-44 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
TABLE_TYPE	VARCHAR(32)	Type of the object
COMMENTS	VARCHAR(1024)	Comment on the object

ALL_TAB_IDENTITY_COLS

ALL_TAB_IDENTITY_COLS describes all table identity columns.

Table 9-45 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the table
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
COLUMN_NAME	VARCHAR(128)	Name of the identity column
GENERATION_TYPE	VARCHAR(32)	Generation type of the identity column. Possible values are ALWAYS or BY DEFAULT
IDENTITY_OPTIONS	VARCHAR(1024)	Options for the identity column sequence generator

ALL_TAB_PLACE

ALL_TAB_PLACE describes node placement of all cluster tables accessible to the current user in the cluste r system.

It is available only on a cluster.

Table 9-46 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the table
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
GROUP_ID	NUMBER	Group identifier of the node where the table placed
GROUP_NAME	VARCHAR(128)	Group name of the node where the table placed
MEMBER_ID	NUMBER	Member identifier of the node where the table placed
MEMBER_NAME	VARCHAR(128)	Member name of the node where the table placed
MEMBER_POSITION	NUMBER	Member position of the node where the table placed
MEMBER_OFFLINE	BOOLEAN	data of the cluster member is offline or not
IS_UPDATE_MASTER	BOOLEAN	whether the cluster member is update master or not
DROPPED	VARCHAR(3)	Indicates whether the table has been dropped and is in the recycle b in (YES) or not (NO)
SCN	VARCHAR(64)	table scn of the node where the table placed
NUM_ROWS	NUMBER	Number of rows in the table
BLOCKS	NUMBER	Number of used blocks of the node where the table placed
LAST_ANALYZED	TIMESTAMP(6) WITH OUT TIME ZONE	Date on which the table was most recently analyzed

ALL_TAB_SHARDS

ALL_TAB_SHARDS describes shard information of sharded tables accessible to the current user in the clus ter system.


It is available only on a cluster.

Table 9-47 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the table
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
SHARD_STRATEGY	VARCHAR(32)	Sharding strategy of the table: the value in (HASH SHARDING, RANGE SHARDING, LIST SHARDING)
SHARD_NAME	VARCHAR(128)	Shard name
SHARD_NUMBER	NUMBER	Shard number
SHARD_DEFINITION	LONG VARCHAR	Shard definition (if hash sharded, the value is null)
GROUP_ID	NUMBER	Group identifier where the shard placed
GROUP_NAME	VARCHAR(128)	Group Name where the shard placed
DROPPED	VARCHAR(3)	Indicates whether the table has been dropped and is in the recycle bin (YES) or not (NO)

ALL_TAB_PRIVS

ALL_TAB_PRIVS describes the object grants, for which the current user is the object owner, grantor, or gr antee, or for which an enabled role or PUBLIC is the grantee.

Table 9-48 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
PRIVILEGE	VARCHAR(32)	Privilege on the object
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTIO N (YES) or not (NO)
HIERARCHY	VARCHAR(3)	Indicates whether the privilege was granted with the HIERARCHY O PTION (YES) or not (NO)

ALL_TAB_PRIVS_MADE

ALL_TAB_PRIVS_MADE describes the object grants for which the current user is the object owner or gran tor.

Table 9-49 Column information

Column name	Data type	Description
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
PRIVILEGE	VARCHAR(32)	Privilege on the object
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTIO N (YES) or not (NO)
HIERARCHY	VARCHAR(3)	Indicates whether the privilege was granted with the HIERARCHY O PTION (YES) or not (NO)

ALL_TAB_PRIVS_RECD

ALL_TAB_PRIVS_RECD describes object grants, for which the current user is the grantee, or for which an enabled role or PUBLIC is the grantee.

Table 9-50 Column information

Column name	Data type	Description
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
PRIVILEGE	VARCHAR(32)	Privilege on the object
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)
HIERARCHY	VARCHAR(3)	Indicates whether the privilege was granted with the HIERARCHY OPTION (YES) or not (NO)

ALL_TBS_PRIVS

ALL_TBS_PRIVS describes the tablespace grants, for which the current user is the grantor, or grantee, or f or which an enabled role or PUBLIC is the grantee.

Table 9-51 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace
PRIVILEGE	VARCHAR(32)	Privilege on the tablespace
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTIO N (YES) or not (NO)

ALL_TBS_PRIVS_MADE

ALL_TBS_PRIVS_MADE describes the tablespace grants for which the current user is the grantor.

Table 9-52 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace
PRIVILEGE	VARCHAR(32)	Privilege on the tablespace
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

ALL_TBS_PRIVS_RECD

ALL_TBS_PRIVS_RECD describes the tablespace grants, for which the current user is the grantee, or for w hich an enabled role or PUBLIC is the grantee.

Table 9-53 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace
PRIVILEGE	VARCHAR(32)	Privilege on the tablespace
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTIO N (YES) or not (NO)

ALL_USERS

ALL_USERS lists all users of the database visible to the current user.

Table 9-54 Column information

Column name	Data type	Description
USERNAME	VARCHAR(128)	Name of the user
USER_ID	NUMBER	ID number of the user
CREATED	TIMESTAMP(6) WITHOUT TIME ZONE	User creation timestamp

ALL_VIEWS

ALL_VIEWS describes the views accessible to the current user.

Table 9-55 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the view
VIEW_SCHEMA	VARCHAR(128)	Schema of the view
VIEW_NAME	VARCHAR(128)	Name of the view
TEXT_LENGTH	NUMBER	Length of the view text
TEXT	LONG VARCHAR	View text
TYPE_TEXT_LENGTH	NUMBER	• reserved Length of the type clause of the typed view
TYPE_TEXT	VARCHAR(4000)	• reserved Type clause of the typed view
OID_TEXT_LENGTH	NUMBER	• reserved Length of the WITH OID clause of the typed view
OID_TEXT	VARCHAR(4000)	• reserved WITH OID clause of the typed view
VIEW_TYPE_OWNER	VARCHAR(128)	• reserved Owner of the type of the view if the view is a typed view
VIEW_TYPE	VARCHAR(32)	• reserved Type of the view if the view is a typed view
SUPERVIEW_NAME	VARCHAR(128)	• reserved Name of the superview
EDITIONING_VIEW	VARCHAR(1)	• reserved Reserved for future use
READ_ONLY	VARCHAR(1)	Indicates whether the view is read-only (Y) or not (N)

Views of DBA_family

The current user has DBA privileges (ACCESS CONTROL ON DATABASE), and the user can retrieve inform ation about all objects.

DBA_ALL_TABLES

DBA_ALL_TABLES describes all object tables and relational tables in the database.

Table 9-56 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the table
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the table
CLUSTER_NAME	VARCHAR(128)	• reserved Name of the cluster
IOT_NAME	VARCHAR(128)	• reserved Name of the index-organized table
STATUS	VARCHAR(32)	• reserved If a previous DROP TABLE operation failed, indicates whether the table is u nusable (UNUSABLE) or valid (VALID)
PCT_FREE	NUMBER	Minimum percentage of free space in a block
PCT_USED	NUMBER	Minimum percentage of used space in a block
INI_TRANS	NUMBER	Initial number of transactions
MAX_TRANS	NUMBER	Maximum number of transactions
INITIAL_EXTENT	NUMBER	Size of the initial extent (in bytes)
NEXT_EXTENT	NUMBER	Size of secondary extents (in bytes)
MIN_EXTENTS	NUMBER	Minimum number of extents allowed in the segment
MAX_EXTENTS	NUMBER	Maximum number of extents allowed in the segment
PCT_INCREASE	NUMBER	• reserved Percentage increase in extent size
FREELISTS	NUMBER	• reserved Number of process freelists allocated to the segment
FREELIST_GROUPS	NUMBER	• reserved Number of freelist groups allocated to the segment
LOGGING	VARCHAR(3)	Indicates whether or not changes to the table are logged
BACKED_UP	VARCHAR(1)	• reserved Indicates whether the table has been backed up since the last modification

Column name	Data type	Description
		(Y) or not (N)
NUM_ROWS	NUMBER	Number of rows in the table
BLOCKS	NUMBER	Number of used blocks in the table
ANAL_BLOCKS	NUMBER	Number of used blocks in the table when most recently analyzed
EMPTY_BLOCKS	NUMBER	• reserved Number of empty (never used) blocks in the table
AVG_SPACE	NUMBER	• reserved Average available free space in the table
CHAIN_CNT	NUMBER	• reserved Number of rows in the table that are chained from one data block to another or that have migrated to a new block, requiring a link to preserve the old rowid
AVG_ROW_LEN	NUMBER	• reserved Average row length, including row overhead
AVG_SPACE_FREELIST_BLOCKS	NUMBER	• reserved Average freespace of all blocks on a freelist
NUM_FREELIST_BLOCKS	NUMBER	• reserved Number of blocks on the freelist
DEGREE	VARCHAR(32)	• reserved Number of threads per instance for scanning the table, or DEFAULT
INSTANCES	VARCHAR(32)	• reserved Number of instances across which the table is to be scanned, or DEFAULT
CACHE	VARCHAR(1)	• reserved Indicates whether the table is to be cached in the buffer cache (Y) or not (N)
TABLE_LOCK	VARCHAR(32)	Indicates whether table locking is enabled (ENABLED) or disabled (DISABLED)
SAMPLE_SIZE	NUMBER	Sample size used in analyzing the table
LAST_ANALYZED	TIMESTAMP(6) WITHOUT TIME ZONE	Date on which the table was most recently analyzed
PARTITIONED	VARCHAR(3)	• reserved Indicates whether the table is partitioned (YES) or not (NO)
IOT_TYPE	VARCHAR(32)	• reserved If the table is an index-organized table, then IOT_TYPE is IOT, IOT_OVERFLOW, or IOT_MAPPING.
OBJECT_ID_TYPE	VARCHAR(32)	• reserved Indicates whether the object ID (OID) is USER-DEFINED or SYSTEM GENERATED
TABLE_TYPE_OWNER	VARCHAR(128)	• reserved If an object table, owner of the type from which the table is created

Column name	Data type	Description
TABLE_TYPE	VARCHAR(128)	• reserved If an object table, type of the table
TEMPORARY	VARCHAR(1)	Indicates whether the table is temporary (Y) or not (N)
SECONDARY	VARCHAR(1)	• reserved Indicates whether the table is a secondary object created by cartridge
NESTED	VARCHAR(3)	• reserved Indicates whether the table is a nested table (YES) or not (NO)
BUFFER_POOL	VARCHAR(32)	• reserved Buffer pool to be used for table blocks
FLASH_CACHE	VARCHAR(32)	• reserved Database Smart Flash Cache hint to be used for table blocks
CELL_FLASH_CACHE	VARCHAR(32)	• reserved Cell flash cache hint to be used for table blocks
ROW_MOVEMENT	VARCHAR(32)	• reserved If a partitioned table, indicates whether row movement is enabled (ENABLED) or disabled (DISABLED)
GLOBAL_STATS	VARCHAR(3)	• reserved For partitioned tables, indicates whether statistics for the table as a whole (global statistics) are accurate (YES)
USER_STATS	VARCHAR(3)	• reserved Indicates whether statistics were entered directly by the user (YES) or not (NO)
DURATION	VARCHAR(32)	Indicates the duration of a temporary table, the value is in (TRANSACTION, SESSION)
SKIP_CORRUPT	VARCHAR(32)	• reserved Indicates whether Database ignores blocks marked corrupt during table and index scans (ENABLED) or raises an error (DISABLED)
MONITORING	VARCHAR(3)	• reserved Indicates whether the table has the MONITORING attribute set (YES) or not (NO)
CLUSTER_OWNER	VARCHAR(128)	• reserved Owner of the cluster, if any
DEPENDENCIES	VARCHAR(32)	• reserved Indicates whether row-level dependency tracking is enabled (ENABLED) or disabled (DISABLED)
COMPRESSION	VARCHAR(32)	• reserved Indicates whether table compression is enabled (ENABLED) or not (DISABLED)
COMPRESS_FOR	VARCHAR(32)	• reserved Default compression for what kind of operations
		Indicates whether the table has been dropped and is in the recycle bin (YES)

Column name	Data type	Description
DROPPED	VARCHAR(3)	or not (NO)
READ_ONLY	VARCHAR(3)	Indicates whether the table IS READ-ONLY (YES) or not (NO)
SEGMENT_CREATED	VARCHAR(3)	Indicates whether the table segment has been created (YES) or not (NO)

DBA_ARGUMENTS

DBA_ARGUMENTS lists all arguments of functions, procedures.

Table 9-57 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of function, procedures or pacakage
SCHEMA_NAME	VARCHAR(128)	Schema Name of function, procedures or package
OBJECT_NAME	VARCHAR(128)	Name of function, procedures
PACKAGE_NAME	VARCHAR(128)	Package Name of function, procedures
OBJECT_ID	NUMBER	ID of a function, procedures
SUBPROGRAM_ID	NUMBER	ID of procedures in pacakage
ARGUMENT_NAME	VARCHAR(128)	Name of argument or attribute name of record type argument
POSITION	NUMBER	Position of argument or position of attribute in record type
SEQUENCE	NUMBER	Sequential order of argument and its attributes
DATA_LEVEL	NUMBER	Nesting depth of the argument for composite types
DATA_TYPE	VARCHAR(128)	Data type of the argument
DEFAULTED	VARCHAR(1)	Whether or not the argument is defaulted
DEFAULT_VALUE	VARCHAR(1)	Reserved for future use
DEFAULT_LENGTH	VARCHAR(1)	Reserved for future use
IN_OUT	VARCHAR(32)	Direction of the argument (IN, OUT, IN/OUT)
DATA_LENGTH	NUMBER	Length of the column(in bytes)
DATA_PRECISION	NUMBER	Length in decimal digits(NUMBER) or binary digits(FLOAT)
DATA_SCALE	NUMBER	Digits to the right of the decimal point in a number
RADIX	NUMBER	Argument radix for a number
CHARACTER_SET_NAME	VARCHAR(128)	Character set name for the argument
TYPE_OWNER	VARCHAR(128)	Owner of the type of the argument
TYPE_NAME	VARCHAR(128)	Name of the type of the argument
TYPE_SUBNAME	VARCHAR(128)	Name of the type of the argument declared in package
TYPE_LINK	VARCHAR(128)	Name of the type of the argument declared in a remote package
PLS_TYPE	VARCHAR(128)	Name of the type of the argument at PSM
CHAR_LENGTH	NUMBER	Character limit for string datatypes
CHAR_USED	VARCHAR(1)	Whether the byte limit(B) or char limit(C) is official for the string
ORIGIN_CON_ID	VARCHAR(256)	ID of the container where the data originates

DBA_CATALOG

DBA_CATALOG lists all tables, views, synonyms, and sequences in the database.

Table 9-58 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the TABLE, VIEW, SYNONYM, SEQUENCE, or UNDEFINED
TABLE_SCHEMA	VARCHAR(128)	Schema of the TABLE, VIEW, SYNONYM, SEQUENCE, or UNDEFINED
TABLE_NAME	VARCHAR(128)	Name of the TABLE, VIEW, SYNONYM, SEQUENCE, or UNDEFINED
TABLE_TYPE	VARCHAR(32)	Type of the TABLE, VIEW, SYNONYM, SEQUENCE, or UNDEFINED

DBA_CLUSTER

DBA_CLUSTER describes all cluster members in the cluster system.


It is available only on a cluster.

Table 9-59 Column information

Column name	Data type	Description
GROUP_ID	NUMBER	Group identifier of the cluster member
GROUP_NAME	VARCHAR(128)	Group name of the cluster member
MEMBER_ID	NUMBER	Member identifier of the cluster member
MEMBER_NAME	VARCHAR(128)	Member name of the cluster member
MEMBER_HOST	VARCHAR(256)	Host name or IP address of the cluster member
MEMBER_PORT	NUMBER	Port number of the cluster member
MEMBER_POSITION	NUMBER	Member position number of the cluster member

DBA_CLUSTER_COMMENTS

DBA_CLUSTER_COMMENTS displays comments on the cluster objects in the cluster system.

It is available only on a cluster.

Table 9-60 Column information

Column name	Data type	Description
OBJECT_NAME	VARCHAR(128)	Name of the cluster object
OBJECT_TYPE	VARCHAR(32)	Type of the cluster object: CLUSTER GROUP, CLUSTER MEMBER
COMMENTS	VARCHAR(1024)	Comment on the cluster object

DBA_CLUSTER_TABLES

DBA_CLUSTER_TABLES describes all cluster tables in the cluster system.

It is available only on a cluster.

Table 9-61 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the table
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
SHARD_STRATEGY	VARCHAR(32)	Sharding strategy of the table: the value in (CLONED, HASH SHARDING, RANGE SHARDING, LIST SHARDING)
SHARD_PLACEMENT	VARCHAR(32)	Shard placement of the table: the value in (AT CLUSTER WIDE or AT CLUSTER GROUP)
SHARD_COUNT	NUMBER	Shard count of the table (if cloned table, the value is null)
SHARD_KEY_COUNT	NUMBER	Shard key column count of the table (if cloned table, the value is null)
HAS_GSI	VARCHAR(3)	Indicate whether the table has global secondary index: (YES) or (NO)
DROPPED	VARCHAR(3)	Indicates whether the table has been dropped and is in the recycle bin (YES) or not (NO)

DBA_COL_COMMENTS

DBA_COL_COMMENTS displays comments on the columns of all tables and views in the database.

Table 9-62 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Name of the column
COMMENTS	VARCHAR(1024)	Comment on the column

DBA_COL_PRIVS

DBA_COL_PRIVS describes all column object grants in the database.

Table 9-63 Column information

Column name	Data type	Description
GRANTOR	CHARACTER VARYING(128)	Name of the user who performed the grant
GRANTEE	CHARACTER VARYING(128)	Name of the user or role to whom access was granted
OWNER	CHARACTER VARYING(128)	Owner of the object
TABLE_SCHEMA	CHARACTER VARYING(128)	Schema of the object
TABLE_NAME	CHARACTER VARYING(128)	Name of the object
COLUMN_NAME	CHARACTER VARYING(128)	Name of the column
PRIVILEGE	CHARACTER VARYING(32)	Privilege on the column
GRANTABLE	CHARACTER VARYING(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

DBA_CONSTRAINTS

DBA_CONSTRAINTS describes all constraint definitions on all tables in the database.

Table 9-64 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the constraint definition
CONSTRAINT_SCHEMA	VARCHAR(128)	Schema of the constraint definition
CONSTRAINT_NAME	VARCHAR(128)	Name of the constraint definition
CONSTRAINT_TYPE	VARCHAR(1)	Type of the constraint definition: the value in (C: check constraint, P: Primary key, U: Unique Key, R: Referential intgrity)
TABLE_OWNER	VARCHAR(128)	Owner of the table (or view) associated with the constraint definition
TABLE_SCHEMA	VARCHAR(128)	Schema of the table (or view) associated with the constraint definition
TABLE_NAME	VARCHAR(128)	Name of the table (or view) associated with the constraint definition
SEARCH_CONDITION	LONG VARCHAR	Text of search condition for a check constraint
R_OWNER	VARCHAR(128)	Owner of the unique constraint definition for the referenced table
R_SCHEMA	VARCHAR(128)	Schema of the unique constraint definition for the referenced table
R_CONSTRAINT_NAME	VARCHAR(128)	Name of the unique constraint definition for the referenced table
DELETE_RULE	VARCHAR(32)	Delete rule for a referential constraint: the value in (NO ACTION, RESTRICT, CASCADE, SET NULL, SET DEFAULT)
UPDATE_RULE	VARCHAR(32)	Update rule for a referential constraint: the value in (NO ACTION, RESTRICT, CASCADE, SET NULL, SET DEFAULT)
STATUS	VARCHAR(32)	Enforcement status of the constraint: the value in (ENABLED, DISABLE)
DEFERRABLE	VARCHAR(32)	Indicates whether the constraint is deferrable (DEFERRABLE) or not (NOT DEFERRABLE)
DEFERRED	VARCHAR(32)	Indicates whether the constraint was initially deferred (DEFERRED) or not (IMMEDIATE)
VALIDATED	VARCHAR(32)	Indicates whether all data may obey the constraint or not: the value in (VALIDATED, NOT VALIDATED)
GENERATED	VARCHAR(32)	Indicates whether the name of the constraint is user-generated (USER NAME) or system-generated (GENERATED NAME)
BAD	VARCHAR(32)	• reserved Indicates whether this constraint specifies a century in an ambiguous manner (BAD) or not (NULL)
RELY	VARCHAR(32)	• reserved When NOT VALIDATED, indicates whether the constraint is to be taken into account for query rewrite (RELY) or not (NULL)
	TIMESTAMP(6) WITH	

Column name	Data type	Description
LAST_CHANGE	OUT TIME ZONE	When the constraint was last enabled or disabled
INDEX_OWNER	VARCHAR(128)	Owner of the index associated with the key constraint
INDEX_SCHEMA	VARCHAR(128)	Schema of the index associated with the key constraint
INDEX_NAME	VARCHAR(128)	Name of the index associated with the key constraint
INVALID	VARCHAR(32)	Indicates whether the constraint is invalid (INVALID) or not (NULL)
VIEW_RELATED	VARCHAR(32)	• reserved Indicates whether the constraint depends on a view (DEPEND ON VIEW) or not (NULL)
DROPPED	VARCHAR(3)	Indicates whether the constraint has been dropped and is in the recycle bin (YES) or not (NO)
COMMENTS	VARCHAR(1024)	Comments of the constraint definition

DBA_CONS_COLUMNS

DBA_CONS_COLUMNS describes all columns in the database that are specified in constraints.

Table 9-65 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the constraint definition
CONSTRAINT_SCHEMA	VARCHAR(128)	Schema of the constraint definition
CONSTRAINT_NAME	VARCHAR(128)	Name of the constraint definition
TABLE_OWNER	VARCHAR(128)	Owner of the table with the constraint definition
TABLE_SCHEMA	VARCHAR(128)	Schema of the table with the constraint definition
TABLE_NAME	VARCHAR(128)	Name of the table with the constraint definition
COLUMN_NAME	VARCHAR(128)	Name of the column or attribute of the object type column specified in the constraint definition
POSITION	NUMBER	Original position of the column or attribute in the definition of the object

DBA_DB_PRIVS

DBA_DB_PRIVS describes all database grants in the database.

Table 9-66 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
PRIVILEGE	VARCHAR(32)	Privilege on the database
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTIO N (YES) or not (NO)

DBA_DEPENDENCIES

DBA_DEPENDENCIES describes all dependencies between objects in the database

Table 9-67 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of object
SCHEMA_NAME	VARCHAR(128)	Schema Name of object
NAME	VARCHAR(128)	Name of object
TYPE	VARCHAR(32)	Type of object: FUNCTION, PROCEDURE, VIEW, PACKAGE, PACKAGE BODY, TRIGGER
REFERENCED_OWNER	VARCHAR(128)	Owner of the referenced object
REFERENCED_SCHEMA_NAME	VARCHAR(128)	Schema Name of the referenced object
REFERENCED_TYPE	VARCHAR(32)	Type of the referenced object: FUNCTION, PROCEDURE, TABLE, VIEW, SEQUENCE, PACKAGE, PACKAGE BODY, TRIGGER
REFERENCED_LINK_NAME	VARCHAR(128)	Name of the link to the parent object
REFERENCED_NAME	VARCHAR(128)	Name of the referenced object
DEPENDENCY_TYPE	VARCHAR(32)	Indicates whether the dependency is a REF dependency (REF) or not (HARD)

DBA_EXTENTS

DBA_EXTENTS describes the extents comprising the segments in all tablespaces in the database.

Table 9-68 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the segment associated with the extent
SEGMENT_SCHEMA	VARCHAR(128)	Schema of the segment associated with the extent
SEGMENT_NAME	VARCHAR(128)	Name of the segment associated with the extent
PARTITION_NAME	VARCHAR(128)	• reserved Object Partition Name (Set to NULL for non-partitioned objects)
SEGMENT_TYPE	VARCHAR(32)	Type of the segment: TABLE, INDEX
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the extent
EXTENT_ID	NUMBER	• reserved Extent number in the segment
FILE_ID	NUMBER	• reserved File identifier number of the file containing the extent
BLOCK_ID	NUMBER	• reserved Starting block number of the extent
BYTES	NUMBER	Size of the extent in bytes
BLOCKS	NUMBER	Size of the extent in Oracle blocks
RELATIVE_FNO	NUMBER	• reserved Relative file number of the first extent block

DBA_GLOBAL_SECONDARY_INDEXES

DBA_GLOBAL_SECONDARY_INDEXES describes all global secondary indexes in the database.


It is available only on a cluster.

Table 9-69 Column information

Column name	Data type	Description
TABLE_OWNER	VARCHAR(128)	Owner of the global secondary indexed object
TABLE_SCHEMA	VARCHAR(128)	Schema of the global secondary indexed object
TABLE_NAME	VARCHAR(128)	Name of the global secondary indexed object
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the global secondary index
INI_TRANS	NUMBER	Initial number of transactions
MAX_TRANS	NUMBER	Maximum number of transactions
INITIAL_EXTENT	NUMBER	Size of the initial extent
NEXT_EXTENT	NUMBER	Size of secondary extents
MIN_EXTENTS	NUMBER	Minimum number of extents allowed in the segment
MAX_EXTENTS	NUMBER	Maximum number of extents allowed in the segment
PCT_FREE	NUMBER	Minimum percentage of free space in a block
LOGGING	VARCHAR(3)	Indicates whether or not changes to the global secondary index are l ogged: (YES) or (NO)
BLOCKS	NUMBER	Number of used blocks in the global secondary index
EMPTY_BLOCKS	NUMBER	Number of empty blocks in the global secondary index
DROPPED	VARCHAR(3)	Indicates whether the global secondary index has been dropped and is in the recycle bin (YES) or not (NO)

DBA_GSI_PLACE

DBA_GSI_PLACE describes node placement of all global secondary indexes in the cluster system.


It is available only on a cluster.

Table 9-70 Column information

Column name	Data type	Description
TABLE_OWNER	VARCHAR(128)	Owner of the global secondary indexed object
TABLE_SCHEMA	VARCHAR(128)	Schema of the global secondary indexed object
TABLE_NAME	VARCHAR(128)	Name of the global secondary indexed object
GROUP_ID	NUMBER	Group identifier of the node where the global secondary index placed
GROUP_NAME	VARCHAR(128)	Group name of the node where the global secondary index placed
MEMBER_ID	NUMBER	Member identifier of the node where the global secondary index placed
MEMBER_NAME	VARCHAR(128)	Member name of the node where the global secondary index placed
MEMBER_OFFLINE	BOOLEAN	data of the cluster member is offline or not
DROPPED	VARCHAR(3)	Indicates whether the global secondary index has been dropped and is in the recycle bin (YES) or not (NO)
BLOCKS	NUMBER	Number of used blocks of the node where the global secondary index placed

DBA_INDEXES

DBA_INDEXES describes all indexes in the database.

Table 9-71 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the index
INDEX_SCHEMA	VARCHAR(128)	Schema of the index
INDEX_NAME	VARCHAR(128)	Name of the index
INDEX_TYPE	VARCHAR(32)	Type of the index: the value in (NORMAL, NORMAL/REV, BITMAP, FUNCTION-BASED NORMAL, FUNCTION-BASED NORMAL/REV, FUNCTION-BASED BITMAP, IOT - TOP, DOMAIN)
TABLE_OWNER	VARCHAR(128)	Owner of the indexed object
TABLE_SCHEMA	VARCHAR(128)	Schema of the indexed object
TABLE_NAME	VARCHAR(128)	Name of the indexed object
TABLE_TYPE	VARCHAR(32)	Type of the indexed object: the value in (NEXT OBJECT, INDEX, TABLE, VIEW, SYNONYM, SEQUENCE)
UNIQUENESS	VARCHAR(32)	Indicates whether the index is unique (UNIQUE) or nonunique (NONUNIQUE)
COMPRESSION	VARCHAR(32)	• reserved Indicates whether index compression is enabled (ENABLED) or not (DISABLED)
PREFIX_LENGTH	NUMBER	• reserved Number of columns in the prefix of the compression key
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the index
INI_TRANS	NUMBER	Initial number of transactions
MAX_TRANS	NUMBER	Maximum number of transactions
INITIAL_EXTENT	NUMBER	Size of the initial extent
NEXT_EXTENT	NUMBER	Size of secondary extents
MIN_EXTENTS	NUMBER	Minimum number of extents allowed in the segment
MAX_EXTENTS	NUMBER	Maximum number of extents allowed in the segment
PCT_INCREASE	NUMBER	• reserved Percentage increase in extent size
PCT_THRESHOLD	NUMBER	• reserved Threshold percentage of block space allowed per index entry
INCLUDE_COLUMN	NUMBER	• reserved Column ID of the last column to be included in index-organized table primary key (non-overflow) index
FREELISTS	NUMBER	• reserved Number of process freelists allocated to this segment
FREELIST_GROUPS	NUMBER	• reserved Number of freelist groups allocated to this segment

Column name	Data type	Description
PCT_FREE	NUMBER	Minimum percentage of free space in a block
LOGGING	VARCHAR(3)	Indicates whether or not changes to the index are logged: (YES) or (NO)
BLOCKS	NUMBER	Number of used blocks in the index
ANAL_BLOCKS	NUMBER	Number of used blocks in the index when most recently analyzed
EMPTY_BLOCKS	NUMBER	Number of empty blocks in the index
BLEVEL	NUMBER	B-Tree level (depth of the index from its root block to its leaf blocks)
LEAF_BLOCKS	NUMBER	Number of leaf blocks in the index
DISTINCT_KEYS	NUMBER	Number of distinct indexed values.
AVG_LEAF_BLOCKS_PER_KEY	NUMBER	• reserved Average number of leaf blocks in which each distinct value in the in dex appears, rounded to the nearest integer
AVG_DATA_BLOCKS_PER_KEY	NUMBER	• reserved Average number of data blocks in the table that are pointed to by a distinct value in the index rounded to the nearest integer
CLUSTERING_FACTOR	NUMBER	Indicates the amount of order of the rows in the table based on the values of the index
STATUS	VARCHAR(32)	• reserved Indicates whether a nonpartitioned index is VALID or UNUSABLE
NUM_ROWS	NUMBER	• reserved Number of rows in the index
SAMPLE_SIZE	NUMBER	Size of the sample used to analyze the index
LAST_ANALYZED	TIMESTAMP(6) WITH OUT TIME ZONE	Date on which this index was most recently analyzed
DEGREE	VARCHAR(32)	• reserved Number of threads per instance for scanning the index, or DEFAUL T
INSTANCES	VARCHAR(32)	• reserved Number of instances across which the indexes to be scanned, or DE FAULT
PARTITIONED	VARCHAR(3)	• reserved Indicates whether the index is partitioned (YES) or not (NO)
TEMPORARY	VARCHAR(1)	Indicates whether the index is on a temporary table (Y) or not (N)
GENERATED	VARCHAR(1)	Indicates whether the name of the index is system-generated (Y) or not (N)
SECONDARY	VARCHAR(1)	• reserved Indicates whether the index is a secondary object created by the me thod of the Data Cartridge (Y) or not (N)
BUFFER_POOL	VARCHAR(32)	• reserved Buffer pool to be used for index blocks
		• reserved

Column name	Data type	Description
FLASH_CACHE	VARCHAR(32)	Database Smart Flash Cache hint to be used for index blocks
CELL_FLASH_CACHE	VARCHAR(32)	• reserved Cell flash cache hint to be used for index blocks
USER_STATS	VARCHAR(3)	• reserved Indicates whether statistics were entered directly by the user (YES) or not (NO)
DURATION	VARCHAR(32)	Indicates the duration of a temporary table, the value is in (TRANSACTION, SESSION)
PCT_DIRECT_ACCESS	NUMBER	• reserved For a secondary index on an index-organized table, the percentage of rows with VALID guess
ITYP_OWNER	VARCHAR(128)	• reserved For a domain index, the owner of the indextype
ITYP_NAME	VARCHAR(128)	• reserved For a domain index, the name of the indextype
PARAMETERS	VARCHAR(1024)	• reserved For a domain index, the parameter string
GLOBAL_STATS	VARCHAR(3)	• reserved For partitioned indexes, indicates whether statistics were collected by analyzing the index as a whole (YES) or were estimated from statistics on underlying index partitions and subpartitions (NO)
DOMIDX_STATUS	VARCHAR(32)	• reserved Status of a domain index
DOMIDX_OPSTATUS	VARCHAR(32)	• reserved Status of the operation on a domain index
FUNCIDX_STATUS	VARCHAR(32)	• reserved Status of a function-based index
JOIN_INDEX	VARCHAR(3)	• reserved Indicates whether the index is a join index (YES) or not (NO)
IOT_REDUNDANT_PKEY_ELIM	VARCHAR(3)	• reserved Indicates whether redundant primary key columns are eliminated from secondary indexes on index-organized tables (YES) or not (NO)
DROPPED	VARCHAR(3)	Indicates whether the index has been dropped and is in the recycle bin (YES) or not (NO)
VISIBILITY	VARCHAR(3)	• reserved Indicates whether the index is VISIBLE or INVISIBLE to the optimizer
DOMIDX_MANAGEMENT	VARCHAR(32)	• reserved If this is a domain index, indicates whether the domain index is system-managed (SYSTEM_MANAGED) or user-managed (USER_MANAGED)
		Indicates whether the index segment has been created (YES) or not

Column name	Data type	Description
SEGMENT_CREATED	VARCHAR(3)	(NO)
COMMENTS	VARCHAR(1024)	Comments of the index
EMPTY_BLOCKS	NUMBER	Number of empty blocks in the index

DBA_IND_COLUMNS

DBA_IND_COLUMNS describes the columns of all the indexes on all tables and clusters in the database.

Table 9-72 Column information

Column name	Data type	Description
INDEX_OWNER	VARCHAR(128)	Owner of the index
INDEX_SCHEMA	VARCHAR(128)	Schema of the index
INDEX_NAME	VARCHAR(128)	Name of the index
TABLE_OWNER	VARCHAR(128)	Owner of the table or cluster
TABLE_SCHEMA	VARCHAR(128)	Schema of the table or cluster
TABLE_NAME	VARCHAR(128)	Name of the table or cluster
COLUMN_NAME	VARCHAR(128)	Column name or attribute of the object type column
COLUMN_POSITION	NUMBER	Position of the column or attribute within the index
COLUMN_LENGTH	NUMBER	Indexed length of the column
CHAR_LENGTH	NUMBER	• reserved Maximum codepoint length of the column
DESCEND	VARCHAR(32)	Indicates whether the column is sorted in descending order (DESC) or ascending order (ASC)
NULL_ORDER	VARCHAR(32)	Indicates whether the null value of the column is sorted in nulls first order (NULLS FIRST) or nulls last order (NULLS LAST)

DBA_IND_PLACE

DBA_IND_PLACE describes node placement of all indexes in the cluster system.


It is available only on a cluster.

Table 9-73 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the index
INDEX_SCHEMA	VARCHAR(128)	Schema of the index
INDEX_NAME	VARCHAR(128)	Name of the index
TABLE_OWNER	VARCHAR(128)	Owner of the indexed object
TABLE_SCHEMA	VARCHAR(128)	Schema of the indexed object
TABLE_NAME	VARCHAR(128)	Name of the indexed object
GROUP_ID	NUMBER	Group identifier of the node where the index placed
GROUP_NAME	VARCHAR(128)	Group name of the node where the index placed
MEMBER_ID	NUMBER	Member identifier of the node where the index placed
MEMBER_NAME	VARCHAR(128)	Member name of the node where the index placed
MEMBER_OFFLINE	BOOLEAN	data of the cluster member is offline or not
DROPPED	VARCHAR(3)	Indicates whether the index has been dropped and is in the recycle bin (YES) or not (NO)
DISTINCT_KEYS	NUMBER	(deprecated)
SAMPLE_SIZE	NUMBER	(deprecated)
BLOCKS	NUMBER	Number of used blocks of the node where the index placed
LAST_ANALYZED	TIMESTAMP(6) WITH OUT TIME ZONE	(deprecated)

DBA_NONSCHEMA_COMMENTS

DBA_NONSCHEMA_COMMENTS displays comments on all non-schema objects (database, authorizations, schemas, tablespaces).

Table 9-74 Column information

Column name	Data type	Description
OBJECT_NAME	VARCHAR(128)	Name of the non-schema object
OBJECT_TYPE	VARCHAR(32)	Type of the non-schema object: DATABASE, PROFILE, AUTHORIZATION, SCHEMA, TABLESPACE
COMMENTS	VARCHAR(1024)	Comments of the non-schema object

DBA_OBJECTS

DBA_OBJECTS describes all objects in the database.

Table 9-75 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the object
SCHEMA_NAME	VARCHAR(128)	Schema of the object
OBJECT_NAME	VARCHAR(128)	Name of the object
SUBOBJECT_NAME	VARCHAR(128)	Name of the subobject (for example, partition)
OBJECT_ID	NUMBER	Dictionary object number of the object
DATA_OBJECT_ID	NUMBER	Dictionary object number of the segment that contains the object
OBJECT_TYPE	VARCHAR(32)	Type of the object (such as TABLE, INDEX)
CREATED	TIMESTAMP(6) WITH OUT TIME ZONE	Timestamp for the creation of the object
LAST_DDL_TIME	TIMESTAMP(6) WITH OUT TIME ZONE	Timestamp for the last modification of the object resulting from a DDL statement
TIMESTAMP	VARCHAR(32)	Timestamp for the specification of the object (character data)
STATUS	VARCHAR(32)	Status of the object: the value in (VALID, INVALID, N/A)
TEMPORARY	VARCHAR(1)	Indicates whether the object is temporary (the current session can see only data that it placed in this object itself) (Y) or not (N)
GENERATED	VARCHAR(1)	Indicates whether the name of this object was system-generated (Y) or not (N)
SECONDARY	VARCHAR(1)	Indicates whether this is a secondary object created by the ODCIIndexCreate method of the Oracle Data Cartridge (Y) or not (N)
NAMESPACE	NUMBER	Namespace for the object
EDITION_NAME	VARCHAR(128)	• reserved Name of the edition in which the object is actual
DROPPED	VARCHAR(3)	Indicates whether the object has been dropped and is in the recycle bin (YES) or not (NO)

DBA_PACKAGE_PRIVS

DBA_PACKAGE_PRIVS describes all packages grants in the database.

Table 9-76 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
PROCEDURE_OWNER	VARCHAR(128)	Owner of the procedure, function or package
PROCEDURE_SCHEMA	VARCHAR(128)	Schema of the procedure, function or package
PROCEDURE_NAME	VARCHAR(128)	Name of the procedure, function or package
PRIVILEGE	VARCHAR(32)	Privilege on the procedure, function or package
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTI ON (YES) or not (NO)

DBA_PROCEDURES

DBA_PROCEDURES lists all function, procedures or pacakage

Table 9-77 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of function, procedures or pacakage
SCHEMA_NAME	VARCHAR(128)	Schema Name of function, procedures or package
OBJECT_NAME	VARCHAR(128)	Name of function, procedures or pacakage
PROCEDURE_NAME	VARCHAR(128)	Name when a procedures in pacakage
OBJECT_ID	NUMBER	ID of a function, procedures or pacakage
SUBPROGRAM_ID	NUMBER	ID of procedures in pacakage
OVERLOAD	VARCHAR(32)	ID of overloading procedure in pacakage
OBJECT_TYPE	VARCHAR(32)	Type of function, procedures or package
AGGREGATE	VARCHAR(3)	Indicate whether the procedure is an aggreage function(YES) or not (NO)
PIPELINED	VARCHAR(3)	Indicate whether the procedure is a pipelined table function(YES) or not(NO)
IMPLTYPEOWNER	VARCHAR(128)	Name of the owner of the implementation type, if any
IMPLTYPENAME	VARCHAR(128)	Name of the implementation type, if any
PARALLEL	VARCHAR(3)	Indicates whether the procedure or function is parallel-enabled (YES) or not (NO)
INTERFACE	VARCHAR(3)	YES, if the procedure/function is a table function implemented usin g the SQLCLI interface; otherwise NO
DETERMINISTIC	VARCHAR(3)	YES, if the procedure/function is declared to be deterministic; other wise NO
AUTHID	VARCHAR(32)	Indicates whether the procedure/function is declared to execute as DEFINER or CURRENT_USER (invoker)

DBA_PROC_PRIVS

DBA_PROC_PRIVS describes the procedure grants, for which the current user is the procedure owner, gra ntor, or grantee, or for which an enabled role or PUBLIC is the grantee.

Table 9-78 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
PROCEDURE_OWNER	VARCHAR(128)	Owner of the procedure, function or package
PROCEDURE_SCHEMA	VARCHAR(128)	Schema of the procedure, function or package
PROCEDURE_NAME	VARCHAR(128)	Name of the procedure, function or package
PRIVILEGE	VARCHAR(32)	Privilege on the procedure, function or package
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTI ON (YES) or not (NO)

DBA_PROFILES

DBA_PROFILES displays all profiles and their limits.

Table 9-79 Column information

Column name	Data type	Description
PROFILE_NAME	VARCHAR(128)	Profile name
RESOURCE_NAME	VARCHAR(128)	Resource name
RESOURCE_TYPE	VARCHAR(32)	Indicates whether the resource profile is a KERNEL or a PASSWORD parameter
LIMIT_VALUE	LONG VARCHAR	Limit placed on this resource for this profile
COMMON	VARCHAR(3)	Indicates whether a given profile is common. (YES or NO)

DBA_RECYCLEBIN

DBA_RECYCLEBIN describes all recycle bins in the database.

Table 9-80 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the object
SCHEMA_NAME	VARCHAR(128)	Schema name of the object
OBJECT_NAME	VARCHAR(128)	Name of the object
ORIGINAL_NAME	VARCHAR(128)	Original name of the object
OPERATION	VARCHAR(4)	Operation carried out on the object
OBJECT_TYPE	VARCHAR(32)	Type of the object
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the object
CREATED_TIME	TIMESTAMP(6) WITH OUT TIME ZONE	Created time of the object
DROPPED_TIME	TIMESTAMP(6) WITH OUT TIME ZONE	Dropped time of the object
DROP_SCN	VARCHAR(128)	System change number (SCN) of the transaction which moved the object to the recycle bin
DROP_GCN	NUMBER	Global change number (GCN) of the transaction which moved the object to the recycle bin
DROP_DCN	NUMBER	Domain change number (DCN) of the transaction which moved the object to the recycle bin
DROP_LCN	NUMBER	Local change number (LCN) of the transaction which moved the object to the recycle bin
CAN_UNDROP	VARCHAR(3)	Indicates whether the object can be undropped (YES) or not (NO)
CAN_PURGE	VARCHAR(3)	Indicates whether the object can be purged (YES) or not (NO)
BASE_OBJECT	NUMBER	Object number of the base object
PURGE_OBJECT	NUMBER	Object number for the object which gets purged

DBA_SCHEMAS

Identify the schemata in the database.

Table 9-81 Column information

Column name	Data type	Description
SCHEMA_OWNER	VARCHAR(128)	Owner of the schema
SCHEMA_NAME	VARCHAR(128)	Name of the schema
CREATED_TIME	TIMESTAMP(6) WITHOUT TIME ZONE	Created time of the schema
MODIFIED_TIME	TIMESTAMP(6) WITHOUT TIME ZONE	Last modified time of the schema
COMMENTS	VARCHAR(1024)	Comments of the schema

DBA_SCHEMA_PATH

DBA_SCHEMA_PATH describes the schema search order of all authorizations in the database.

Table 9-82 Column information

Column name	Data type	Description
AUTH_NAME	VARCHAR(128)	Name of the authorization
SCHEMA_NAME	VARCHAR(128)	Name of the schema
SEARCH_ORDER	NUMBER	Schema search order of the authorization

DBA_SCHEMA_PRIVS

DBA_SCHEMA_PRIVS describes all schema grants in the database.

Table 9-83 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the schema
SCHEMA_NAME	VARCHAR(128)	Name of the schema
PRIVILEGE	VARCHAR(32)	Privilege on the schema
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

DBA_SEQUENCES

DBA_SEQUENCES describes all sequences in the database.

Table 9-84 Column information

Column name	Data type	Description
SEQUENCE_OWNER	VARCHAR(128)	Owner of the sequence
SEQUENCE_SCHEMA	VARCHAR(128)	Schema of the sequence
SEQUENCE_NAME	VARCHAR(128)	Sequence name
MIN_VALUE	NUMBER	Minimum value of the sequence
MAX_VALUE	NUMBER	Maximum value of the sequence
INCREMENT_BY	NUMBER	Value by which sequence is incremented
CYCLE_FLAG	VARCHAR(1)	Indicates whether the sequence wraps around on reaching the limit (Y) or not (N)
ORDER_FLAG	VARCHAR(1)	• reserved Indicates whether sequence numbers are generated in order (Y) or not (N)
CACHE_SIZE	NUMBER	Number of sequence numbers to cache
LAST_NUMBER	NUMBER	Last sequence number written to database. If a sequence uses caching, the number written to database is the last number placed in the sequence cache.
COMMENTS	VARCHAR(1024)	Comments of the sequence

DBA_SEQ_PRIVS

DBA_SEQ_PRIVS describes all sequence grants in the database.

Table 9-85 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
SEQUENCE_OWNER	VARCHAR(128)	Owner of the sequence
SEQUENCE_SCHEMA	VARCHAR(128)	Schema of the sequence
SEQUENCE_NAME	VARCHAR(128)	Name of the sequence
PRIVILEGE	VARCHAR(32)	Privilege on the sequence
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

DBA_SHARD_KEY_COLUMNS

DBA_SHARD_KEY_COLUMNS describes shard key columns of all shareded tables in the cluster system.

It is available only on a cluster.

Table 9-86 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the table
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
COLUMN_NAME	VARCHAR(128)	Column name of the shard key
COLUMN_POSITION	NUMBER	Position of the column within the shard key

DBA_SOURCE

DBA_SOURCE describes the text source of the stored objects accessible to the current user.

Table 9-87 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of object
SCHEMA_NAME	VARCHAR(128)	Schema Name of object
NAME	VARCHAR(128)	Name of object
TYPE	VARCHAR(32)	Type of object: FUNCTION, PROCEDURE, PACKAGE, PACKAGE BODY, TRIGGER
LINE	NUMBER	Line number of this line of source
TEXT	LONG VARCHAR	Text source of the strored object
ORIGIN_CON_ID	VARCHAR(256)	ID of the container where the data originates

DBA_STAT_SYSTEM

DBA_STAT_SYSTEM describes analyzed system statistics.

Table 9-88 Column information

Column name	Data type	Description
CPU_OPS	NATIVE_BIGINT	OPS(operations per second) of CPU
NETWORK_IOPS	NATIVE_BIGINT	IOPS(I/O operations per second) of Cluster NETWORK
NETWORK_BUFSIZE	NATIVE_BIGINT	buffer size of Cluster NETWORK when analyzed
BUFFER_MISS_PERCENT	NATIVE_BIGINT	disk buffer miss percent
LAST_ANALYZED	TIMESTAMP(6) WITHOUT TIME ZONE	Date on which the table was most recently analyzed

DBA_SYS_PRIVS

DBA_SYS_PRIVS describes all system (database, tablespace, schema) privileges in the database.

Table 9-89 Column information

Column name	Data type	Description
GRANTEE	VARCHAR(128)	Name of the grantee
PRIVILEGE	VARCHAR(256)	System(database, tablespace, schema) privilege
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTIO N (YES) or not (NO)
ADMIN_OPTION	VARCHAR(3)	equal to GRANTABLE column

DBA_SYNONYMS

DBA_SYNONYMS describes all synonyms in the database.

Table 9-90 Column information

Column name	Data type	Description
SYNONYM_OWNER	VARCHAR(128)	Owner of the synonym
SYNONYM_SCHEMA	VARCHAR(128)	Schema of the synonym
SYNONYM_NAME	VARCHAR(128)	Synonym name
OBJECT_SCHEMA_NAME	VARCHAR(128)	Object schema name
OBJECT_NAME	VARCHAR(128)	Object name
DB_LINK	VARCHAR(128)	Reserved for future use

DBA_TABLES

DBA_TABLES describes all relational tables in the database.

Table 9-91 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the table
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the table
CLUSTER_NAME	VARCHAR(128)	• reserved Name of the cluster
IOT_NAME	VARCHAR(128)	• reserved Name of the index-organized table
STATUS	VARCHAR(32)	• reserved If a previous DROP TABLE operation failed, indicates whether the table is unusable (UNUSABLE) or valid (VALID)
PCT_FREE	NUMBER	Minimum percentage of free space in a block
PCT_USED	NUMBER	Minimum percentage of used space in a block
INI_TRANS	NUMBER	Initial number of transactions
MAX_TRANS	NUMBER	Maximum number of transactions
INITIAL_EXTENT	NUMBER	Size of the initial extent (in bytes)
NEXT_EXTENT	NUMBER	Size of secondary extents (in bytes)
MIN_EXTENTS	NUMBER	Minimum number of extents allowed in the segment
MAX_EXTENTS	NUMBER	Maximum number of extents allowed in the segment
PCT_INCREASE	NUMBER	• reserved Percentage increase in extent size
FREELISTS	NUMBER	• reserved Number of process freelists allocated to the segment
FREELIST_GROUPS	NUMBER	• reserved Number of freelist groups allocated to the segment
LOGGING	VARCHAR(3)	Indicates whether or not changes to the table are logged
BACKED_UP	VARCHAR(1)	• reserved Indicates whether the table has been backed up since the last modification (Y) or not (N)
NUM_ROWS	NUMBER	Number of rows in the table
BLOCKS	NUMBER	Number of used blocks in the table
ANAL_BLOCKS	NUMBER	Number of used blocks in the table when most recently analyzed
EMPTY_BLOCKS	NUMBER	• reserved Number of empty (never used) blocks in the table
		• reserved

Column name	Data type	Description
AVG_SPACE	NUMBER	Average available free space in the table
CHAIN_CNT	NUMBER	• reserved Number of rows in the table that are chained from one data block to another or that have migrated to a new block, requiring a link to preserve the old rowid
AVG_ROW_LEN	NUMBER	• reserved Average row length, including row overhead
AVG_SPACE_FREELIST_BLOCKS	NUMBER	• reserved Average freespace of all blocks on a freelist
NUM_FREELIST_BLOCKS	NUMBER	• reserved Number of blocks on the freelist
DEGREE	VARCHAR(32)	• reserved Number of threads per instance for scanning the table, or DEFAULT
INSTANCES	VARCHAR(32)	• reserved Number of instances across which the table is to be scanned, or DEFAULT
CACHE	VARCHAR(1)	• reserved Indicates whether the table is to be cached in the buffer cache (Y) or not (N)
TABLE_LOCK	VARCHAR(32)	Indicates whether table locking is enabled (ENABLED) or disabled (DISABLED)
SAMPLE_SIZE	NUMBER	Sample size used in analyzing the table
LAST_ANALYZED	TIMESTAMP(6) WITHOUT TIME ZONE	Date on which the table was most recently analyzed
PARTITIONED	VARCHAR(3)	• reserved Indicates whether the table is partitioned (YES) or not (NO)
IOT_TYPE	VARCHAR(32)	• reserved If the table is an index-organized table, then IOT_TYPE is IOT, IOT_OVERFLOW, or IOT_MAPPING.
TEMPORARY	VARCHAR(1)	Indicates whether the table is temporary (Y) or not (N)
SECONDARY	VARCHAR(1)	• reserved Indicates whether the table is a secondary object created by cartridge
NESTED	VARCHAR(3)	• reserved Indicates whether the table is a nested table (YES) or not (NO)
BUFFER_POOL	VARCHAR(32)	• reserved Buffer pool to be used for table blocks
FLASH_CACHE	VARCHAR(32)	• reserved Database Smart Flash Cache hint to be used for table blocks
CELL_FLASH_CACHE	VARCHAR(32)	• reserved Cell flash cache hint to be used for table blocks
		• reserved

Column name	Data type	Description
ROW_MOVEMENT	VARCHAR(32)	If a partitioned table, indicates whether row movement is enabled (ENABLED) or disabled (DISABLED)
GLOBAL_STATS	VARCHAR(3)	• reserved For partitioned tables, indicates whether statistics for the table as a whole (global statistics) are accurate (YES)
USER_STATS	VARCHAR(3)	• reserved Indicates whether statistics were entered directly by the user (YES) or not (NO)
DURATION	VARCHAR(32)	Indicates the duration of a temporary table, the value is in (TRANSACTION, SESSION)
SKIP_CORRUPT	VARCHAR(32)	• reserved Indicates whether Database ignores blocks marked corrupt during table and index scans (ENABLED) or raises an error (DISABLED)
MONITORING	VARCHAR(3)	• reserved Indicates whether the table has the MONITORING attribute set (YES) or not (NO)
CLUSTER_OWNER	VARCHAR(128)	• reserved Owner of the cluster, if any
DEPENDENCIES	VARCHAR(32)	• reserved Indicates whether row-level dependency tracking is enabled (ENABLED) or disabled (DISABLED)
COMPRESSION	VARCHAR(32)	• reserved Indicates whether table compression is enabled (ENABLED) or not (DISABLED)
COMPRESS_FOR	VARCHAR(32)	• reserved Default compression for what kind of operations
DROPPED	VARCHAR(3)	Indicates whether the table has been dropped and is in the recycle bin (YES) or not (NO)
READ_ONLY	VARCHAR(3)	Indicates whether the table IS READ-ONLY (YES) or not (NO)
SEGMENT_CREATED	VARCHAR(3)	Indicates whether the table segment has been created (YES) or not (NO)
RESULT_CACHE	VARCHAR(32)	• reserved Result cache mode annotation for the table: the value in (NULL, DEFAULT, FORCE, MANUAL)

DBA_TABLESPACES

DBA_TABLESPACES describes all tablespaces in the database.

Table 9-92 Column information

Column name	Data type	Description
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace
BLOCK_SIZE	NUMBER	Tablespace block size
INITIAL_EXTENT	NUMBER	• reserved Default initial extent size (in bytes)
NEXT_EXTENT	NUMBER	• reserved Default incremental extent size (in bytes)
MIN_EXTENTS	NUMBER	• reserved Default minimum number of extents
MAX_EXTENTS	NUMBER	• reserved Default maximum number of extents
MAX_SIZE	NUMBER	• reserved Default maximum size of segments
PCT_INCREASE	NUMBER	• reserved Default percent increase for extent size
MIN_EXTLEN	NUMBER	• reserved Minimum extent size for this tablespace (in bytes)
STATUS	VARCHAR(32)	Tablespace status: the value in (ONLINE, OFFLINE, READ ONLY)
CONTENTS	VARCHAR(32)	Tablespace contents: the value in (SYSTEM, DATA, TEMPORARY, UNDO)
LOGGING	VARCHAR(32)	Default logging attribute: LOGGING, NOLOGGING
FORCE_LOGGING	VARCHAR(3)	• reserved Indicates whether the tablespace is under force logging mode (YES) or not (NO)
EXTENT_MANAGEMENT	VARCHAR(32)	• reserved Indicates whether the extents in the tablespace are dictionary managed (DICTIONARY) or locally managed (LOCAL)
ALLOCATION_TYPE	VARCHAR(32)	• reserved Type of extent allocation in effect for the tablespace: the value in (SYSTEM, UNIFORM, USER)
PLUGGED_IN	VARCHAR(3)	• reserved Indicates whether the tablespace is plugged in (YES) or not (NO)
SEGMENT_SPACE_MANAGEMENT	VARCHAR(32)	• reserved Indicates whether the free and used segment space in the tablespace is managed using free lists (MANUAL) or bitmaps (AUTO)
DEF_TAB_COMPRESSION	VARCHAR(32)	• reserved Indicates whether default table compression is enabled (ENABLED)

Column name	Data type	Description
		or not (DISABLED)
RETENTION	VARCHAR(32)	• reserved Undo tablespace retention: the value in (GUARANTEE, NOGUARANTEE, NOT APPLY)
BIGFILE	VARCHAR(3)	• reserved Indicates whether the tablespace is a bigfile tablespace (YES) or a smallfile tablespace (NO)
PREDICATE_EVALUATION	VARCHAR(32)	• reserved Indicates whether predicates are evaluated by host (HOST) or by storage (STORAGE)
ENCRYPTED	VARCHAR(3)	• reserved Indicates whether the tablespace is encrypted (YES) or not (NO)
COMPRESS_FOR	VARCHAR(32)	• reserved Indicates whether the tablespace is encrypted (YES) or not (NO)

DBA_TAB_COLS

DBA_TAB_COLS describes the columns (including hidden columns) of all tables, views, and clusters in the database.

Table 9-93 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Column name
DATA_TYPE	VARCHAR(128)	Datatype of the column
DATA_TYPE_MOD	VARCHAR(3)	• reserved Datatype modifier of the column
DATA_TYPE_OWNER	VARCHAR(128)	• reserved Owner of the datatype of the column
DATA_LENGTH	NUMBER	Length of the column (in bytes)
DATA_PRECISION	NUMBER	Decimal precision for NUMBER datatype; binary precision for FLOAT datatype; NULL for all other datatypes
DATA_SCALE	NUMBER	Digits to the right of the decimal point in a number
NULLABLE	VARCHAR(1)	Indicates whether a column allows NULLs.
COLUMN_ID	NUMBER	Sequence number of the column as created
DEFAULT_LENGTH	NUMBER	Length of the default value for the column
DATA_DEFAULT	LONG VARCHAR	Default value for the column
NUM_DISTINCT	NUMBER	Number of distinct values in the column
LOW_VALUE	VARBINARY(32)	Low value in the column
HIGH_VALUE	VARBINARY(32)	High value in the column
DENSITY	NUMBER	• reserved If a histogram is available on COLUMN_NAME, then this column dis plays the selectivity of a value that spans fewer than 2 endpoints in t he histogram.
NUM_NULLS	NUMBER	Number of NULLs in the column
NUM_BUCKETS	NUMBER	• reserved Number of buckets in the histogram for the column
LAST_ANALYZED	TIMESTAMP(6) WITH OUT TIME ZONE	Date on which this column was most recently analyzed
SAMPLE_SIZE	NUMBER	Sample size used in analyzing this column
CHARACTER_SET_N AME	VARCHAR(128)	• reserved Name of the character set
CHAR_COL_DECL_LE NGTH	NUMBER	Declaration length of the character type column
		• reserved

Column name	Data type	Description
GLOBAL_STATS	VARCHAR(3)	For partitioned tables, indicates whether column statistics were collected for the table
USER_STATS	VARCHAR(3)	• reserved Indicates whether statistics were entered directly by the user (YES) or not (NO)
AVG_COL_LEN	NUMBER	Average length of the column (in bytes)
CHAR_LENGTH	NUMBER	Displays the length of the column in characters.
CHAR_USED	VARCHAR(1)	Indicates that the column uses BYTE length semantics (B) or CHAR length semantics (C)
V80_FMT_IMAGE	VARCHAR(3)	• reserved Indicates whether the column data is in release older image format (YES) or not (NO)
DATA_UPGRADED	VARCHAR(3)	• reserved Indicates whether the column data has been upgraded to the latest type version format (YES) or not (NO)
HIDDEN_COLUMN	VARCHAR(3)	Indicates whether the column is a hidden column (YES) or not (NO)
VIRTUAL_COLUMN	VARCHAR(3)	• reserved Indicates whether the column is a virtual column (YES) or not (NO)
SEGMENT_COLUMN_ID	NUMBER	Sequence number of the column in the segment
INTERNAL_COLUMN_ID	NUMBER	Internal sequence number of the column
HISTOGRAM	VARCHAR(32)	• reserved Indicates existence/type of histogram
QUALIFIED_COL_NAME	VARCHAR(4000)	Qualified column name
IDENTITY_COLUMN	VARCHAR(3)	Indicates whether this is an identity column (YES) or not (NO)

DBA_TAB_COLUMNS

DBA_TAB_COLUMNS describes the columns of the tables, views, and clusters accessible to the current us er.

Table 9-94 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Column name
DATA_TYPE	VARCHAR(128)	Datatype of the column
DATA_TYPE_MOD	VARCHAR(3)	• reserved Datatype modifier of the column
DATA_TYPE_OWNER	VARCHAR(128)	• reserved Owner of the datatype of the column
DATA_LENGTH	NUMBER	Length of the column (in bytes)
DATA_PRECISION	NUMBER	Decimal precision for NUMBER datatype; binary precision for FLOAT datatype; NULL for all other datatypes
DATA_SCALE	NUMBER	Digits to the right of the decimal point in a number
NULLABLE	VARCHAR(1)	Indicates whether a column allows NULLs.
COLUMN_ID	NUMBER	Sequence number of the column as created
DEFAULT_LENGTH	NUMBER	Length of the default value for the column
DATA_DEFAULT	LONG VARCHAR	Default value for the column
NUM_DISTINCT	NUMBER	Number of distinct values in the column
LOW_VALUE	VARBINARY(32)	Low value in the column
HIGH_VALUE	VARBINARY(32)	High value in the column
DENSITY	NUMBER	• reserved If a histogram is available on COLUMN_NAME, then this column dis plays the selectivity of a value that spans fewer than 2 endpoints in t he histogram.
NUM_NULLS	NUMBER	Number of NULLs in the column
NUM_BUCKETS	NUMBER	• reserved Number of buckets in the histogram for the column
LAST_ANALYZED	TIMESTAMP(6) WITH OUT TIME ZONE	Date on which this column was most recently analyzed
SAMPLE_SIZE	NUMBER	Sample size used in analyzing this column
CHARACTER_SET_N AME	VARCHAR(128)	• reserved Name of the character set
CHAR_COL_DECL_LE NGTH	NUMBER	Declaration length of the character type column
		• reserved

Column name	Data type	Description
GLOBAL_STATS	VARCHAR(3)	For partitioned tables, indicates whether column statistics were collected for the table
USER_STATS	VARCHAR(3)	• reserved Indicates whether statistics were entered directly by the user (YES) or not (NO)
AVG_COL_LEN	NUMBER	Average length of the column (in bytes)
CHAR_LENGTH	NUMBER	Displays the length of the column in characters.
CHAR_USED	VARCHAR(1)	Indicates that the column uses BYTE length semantics (B) or CHAR length semantics (C)
V80_FMT_IMAGE	VARCHAR(3)	• reserved Indicates whether the column data is in release older image format (YES) or not (NO)
DATA_UPGRADED	VARCHAR(3)	• reserved Indicates whether the column data has been upgraded to the latest type version format (YES) or not (NO)
HISTOGRAM	VARCHAR(32)	• reserved Indicates existence/type of histogram
IDENTITY_COLUMN	VARCHAR(3)	Indicates whether this is an identity column (YES) or not (NO)

DBA_TAB_COMMENTS

DBA_TAB_COMMENTS displays comments on all tables and views in the database.

Table 9-95 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
TABLE_TYPE	VARCHAR(32)	Type of the object
COMMENTS	VARCHAR(1024)	Comment on the object

DBA_TAB_IDENTITY_COLS

DBA_TAB_IDENTITY_COLS describes all table identity columns.

Table 9-96 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the table
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
COLUMN_NAME	VARCHAR(128)	Name of the identity column
GENERATION_TYPE	VARCHAR(32)	Generation type of the identity column. Possible values are ALWAYS or BY DEFAULT
IDENTITY_OPTIONS	VARCHAR(1024)	Options for the identity column sequence generator

DBA_TAB_PLACE

DBA_TAB_PLACE describes node placement of all cluster tables in the cluster system.

It is available only on a cluster.

Table 9-97 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the table
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
GROUP_ID	NUMBER	Group identifier of the node where the table placed
GROUP_NAME	VARCHAR(128)	Group name of the node where the table placed
MEMBER_ID	NUMBER	Member identifier of the node where the table placed
MEMBER_NAME	VARCHAR(128)	Member name of the node where the table placed
MEMBER_POSITION	NUMBER	Member position of the node where the table placed
MEMBER_OFFLINE	BOOLEAN	data of the cluster member is offline or not
IS_UPDATE_MASTER	BOOLEAN	whether the cluster member is update master or not
DROPPED	VARCHAR(3)	Indicates whether the table has been dropped and is in the recycle b in (YES) or not (NO)
SCN	VARCHAR(64)	table scn of the node where the table placed
NUM_ROWS	NUMBER	Number of rows in the table
BLOCKS	NUMBER	Number of used blocks of the node where the table placed
LAST_ANALYZED	TIMESTAMP(6) WITH OUT TIME ZONE	Date on which the table was most recently analyzed

DBA_TAB_PRIVS

DBA_TAB_PRIVS describes all object grants in the database.

Table 9-98 Column information

Column name	Data type	Description
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
PRIVILEGE	VARCHAR(32)	Privilege on the object
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTIO N (YES) or not (NO)
HIERARCHY	VARCHAR(3)	Indicates whether the privilege was granted with the HIERARCHY O PTION (YES) or not (NO)

DBA_TAB_SHARDS

DBA_TAB_SHARDS describes shard information of all sharded tables in the cluster system.

It is available only on a cluster.

Table 9-99 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the table
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
SHARD_STRATEGY	VARCHAR(32)	Sharding strategy of the table: the value in (HASH SHARDING, RANGE SHARDING, LIST SHARDING)
SHARD_NAME	VARCHAR(128)	Shard name
SHARD_NUMBER	NUMBER	Shard number
SHARD_DEFINITION	LONG VARCHAR	Shard definition (if hash sharded, the value is null)
GROUP_ID	NUMBER	Group identifier where the shard placed
GROUP_NAME	VARCHAR(128)	Group Name where the shard placed
DROPPED	VARCHAR(3)	Indicates whether the table has been dropped and is in the recycle bin (YES) or not (NO)

DBA_TBS_PRIVS

DBA_TBS_PRIVS describes all tablespace grants in the database.

Table 9-100 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace
PRIVILEGE	VARCHAR(32)	Privilege on the tablespace
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

DBA_USERS

DBA_USERS describes all users of the database.

Table 9-101 Column information

Column name	Data type	Description
USERNAME	VARCHAR(128)	Name of the user
USER_ID	NUMBER	ID number of the user
PASSWORD	VARCHAR(128)	encrypted password
ACCOUNT_STATUS	VARCHAR(32)	Account status: the value in (OPEN, EXPIRED, EXPIRED(GRACE), LOCKED(TIMED), LOCKED, EXPIRED & LOCKED(TIMED), EXPIRED(GRACE) & LOCKED (TIMED), EXPIRED & LOCKED, EXPIRED(GRACE) & LOCKED)
LOCK_DATE	TIMESTAMP(6) WITHOUT TIME ZONE	Timestamp the account was locked if account status was LOCKED
EXPIRY_DATE	TIMESTAMP(6) WITHOUT TIME ZONE	Timestamp of expiration of the account
FAILED_LOGIN_ATTEMPTS	NUMBER	Consecutive failed login attempts count
DEFAULT_TABLESPACE	VARCHAR(128)	Default tablespace for data
TEMPORARY_TABLESPACE	VARCHAR(128)	Name of the default tablespace for temporary tables or the name of a tablespace group
INDEX_TABLESPACE	VARCHAR(128)	Default tablespace for index
CREATED	TIMESTAMP(6) WITHOUT TIME ZONE	User creation timestamp
PROFIL_NAME	VARCHAR(128)	User resource profile name
INITIAL_RSRC_CONSUMER_GROUP	VARCHAR(128)	• reserved Initial resource consumer group for the user
EXTERNAL_NAME	VARCHAR(128)	• reserved User external name
PASSWORD_VERSIONS	VARCHAR(32)	• reserved Shows the list of versions of the password hashes (verifiers).
EDITIONS_ENABLED	VARCHAR(1)	Indicates whether editions have been enabled for the corresponding user (Y) or not (N).
AUTHENTICATION_TYPE	VARCHAR(32)	• reserved Indicates the authentication mechanism for the user.

DBA_VIEWS

DBA_VIEWS describes all views in the database.

Table 9-102 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the view
VIEW_SCHEMA	VARCHAR(128)	Schema of the view
VIEW_NAME	VARCHAR(128)	Name of the view
TEXT_LENGTH	NUMBER	Length of the view text
TEXT	LONG VARCHAR	View text
TYPE_TEXT_LENGTH	NUMBER	• reserved Length of the type clause of the typed view
TYPE_TEXT	VARCHAR(4000)	• reserved Type clause of the typed view
OID_TEXT_LENGTH	NUMBER	• reserved Length of the WITH OID clause of the typed view
OID_TEXT	VARCHAR(4000)	• reserved WITH OID clause of the typed view
VIEW_TYPE_OWNER	VARCHAR(128)	• reserved Owner of the type of the view if the view is a typed view
VIEW_TYPE	VARCHAR(32)	• reserved Type of the view if the view is a typed view
SUPERVIEW_NAME	VARCHAR(128)	• reserved Name of the superview
EDITIONING_VIEW	VARCHAR(1)	Reserved for future use
READ_ONLY	VARCHAR(1)	Indicates whether the view is read-only (Y) or not (N)

Views of USER_family

It retrieves information about objects which are owned by current user.

USER_ALL_TABLES

USER_ALL_TABLES describes the object tables and relational tables owned by the current user.

Table 9-103 Column information

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the table
CLUSTER_NAME	VARCHAR(128)	• reserved Name of the cluster
IOT_NAME	VARCHAR(128)	• reserved Name of the index-organized table
STATUS	VARCHAR(32)	• reserved If a previous DROP TABLE operation failed, indicates whether the table is unusable (UNUSABLE) or valid (VALID)
PCT_FREE	NUMBER	Minimum percentage of free space in a block
PCT_USED	NUMBER	Minimum percentage of used space in a block
INI_TRANS	NUMBER	Initial number of transactions
MAX_TRANS	NUMBER	Maximum number of transactions
INITIAL_EXTENT	NUMBER	Size of the initial extent (in bytes)
NEXT_EXTENT	NUMBER	Size of secondary extents (in bytes)
MIN_EXTENTS	NUMBER	Minimum number of extents allowed in the segment
MAX_EXTENTS	NUMBER	Maximum number of extents allowed in the segment
PCT_INCREASE	NUMBER	• reserved Percentage increase in extent size
FREELISTS	NUMBER	• reserved Number of process freelists allocated to the segment
FREELIST_GROUPS	NUMBER	• reserved Number of freelist groups allocated to the segment
LOGGING	VARCHAR(3)	Indicates whether or not changes to the table are logged
BACKED_UP	VARCHAR(1)	• reserved Indicates whether the table has been backed up since the last modification (Y) or not (N)
NUM_ROWS	NUMBER	Number of rows in the table
BLOCKS	NUMBER	Number of used blocks in the table

Column name	Data type	Description
ANAL_BLOCKS	NUMBER	Number of used blocks in the table when most recently analyzed
EMPTY_BLOCKS	NUMBER	• reserved Number of empty (never used) blocks in the table
AVG_SPACE	NUMBER	• reserved Average available free space in the table
CHAIN_CNT	NUMBER	• reserved Number of rows in the table that are chained from one data block to another or that have migrated to a new block, requiring a link to preserve the old rowid
AVG_ROW_LEN	NUMBER	• reserved Average row length, including row overhead
AVG_SPACE_FREELIST_BLOCKS	NUMBER	• reserved Average freespace of all blocks on a freelist
NUM_FREELIST_BLOCKS	NUMBER	• reserved Number of blocks on the freelist
DEGREE	VARCHAR(32)	• reserved Number of threads per instance for scanning the table, or DEFAULT
INSTANCES	VARCHAR(32)	• reserved Number of instances across which the table is to be scanned, or DEFAULT
CACHE	VARCHAR(1)	• reserved Indicates whether the table is to be cached in the buffer cache (Y) or not (N)
TABLE_LOCK	VARCHAR(32)	Indicates whether table locking is enabled (ENABLED) or disabled (DISABLED)
SAMPLE_SIZE	NUMBER	Sample size used in analyzing the table
LAST_ANALYZED	TIMESTAMP(6) WITHOUT TIME ZONE	Date on which the table was most recently analyzed
PARTITIONED	VARCHAR(3)	• reserved Indicates whether the table is partitioned (YES) or not (NO)
IOT_TYPE	VARCHAR(32)	• reserved If the table is an index-organized table, then IOT_TYPE is IOT, IOT_OVERFLOW, or IOT_MAPPING.
OBJECT_ID_TYPE	VARCHAR(32)	• reserved Indicates whether the object ID (OID) is USER-DEFINED or SYSTEM GENERATED
TABLE_TYPE_OWNER	VARCHAR(128)	• reserved If an object table, owner of the type from which the table is created
TABLE_TYPE	VARCHAR(128)	• reserved If an object table, type of the table
TEMPORARY	VARCHAR(1)	Indicates whether the table is temporary (Y) or not (N)

Column name	Data type	Description
SECONDARY	VARCHAR(1)	• reserved Indicates whether the table is a secondary object created by cartridge
NESTED	VARCHAR(3)	• reserved Indicates whether the table is a nested table (YES) or not (NO)
BUFFER_POOL	VARCHAR(32)	• reserved Buffer pool to be used for table blocks
FLASH_CACHE	VARCHAR(32)	• reserved Database Smart Flash Cache hint to be used for table blocks
CELL_FLASH_CACHE	VARCHAR(32)	• reserved Cell flash cache hint to be used for table blocks
ROW_MOVEMENT	VARCHAR(32)	• reserved If a partitioned table, indicates whether row movement is enabled (ENABLED) or disabled (DISABLED)
GLOBAL_STATS	VARCHAR(3)	• reserved For partitioned tables, indicates whether statistics for the table as a whole (global statistics) are accurate (YES)
USER_STATS	VARCHAR(3)	• reserved Indicates whether statistics were entered directly by the user (YES) or not (NO)
DURATION	VARCHAR(32)	Indicates the duration of a temporary table, the value is in (TRANSACTION, SESSION)
SKIP_CORRUPT	VARCHAR(32)	• reserved Indicates whether Database ignores blocks marked corrupt during table and index scans (ENABLED) or raises an error (DISABLED)
MONITORING	VARCHAR(3)	• reserved Indicates whether the table has the MONITORING attribute set (YES) or not (NO)
CLUSTER_OWNER	VARCHAR(128)	• reserved Owner of the cluster, if any
DEPENDENCIES	VARCHAR(32)	• reserved Indicates whether row-level dependency tracking is enabled (ENABLED) or disabled (DISABLED)
COMPRESSION	VARCHAR(32)	• reserved Indicates whether table compression is enabled (ENABLED) or not (DISABLED)
COMPRESS_FOR	VARCHAR(32)	• reserved Default compression for what kind of operations
DROPPED	VARCHAR(3)	Indicates whether the table has been dropped and is in the recycle bin (YES) or not (NO)
READ_ONLY	VARCHAR(3)	Indicates whether the table IS READ-ONLY (YES) or not (NO)

Column name	Data type	Description
SEGMENT_CREATED	VARCHAR(3)	Indicates whether the table segment has been created (YES) or not (NO)

USER_ARGUMENTS

USER_ARGUMENTS lists all arguments of functions, procedures.

Table 9-104 Column information

Column name	Data type	Description
SCHEMA_NAME	VARCHAR(128)	Schema Name of function, procedures or package
OBJECT_NAME	VARCHAR(128)	Name of function, procedures
PACKAGE_NAME	VARCHAR(128)	Package Name of function, procedures
OBJECT_ID	NUMBER	ID of a function, procedures
SUBPROGRAM_ID	NUMBER	ID of procedures in pacakage
ARGUMENT_NAME	VARCHAR(128)	Name of argument or attribute name of record type argument
POSITION	NUMBER	Position of argument or position of attribute in record type
SEQUENCE	NUMBER	Sequential order of argument and its attributes
DATA_LEVEL	NUMBER	Nesting depth of the argument for composite types
DATA_TYPE	VARCHAR(128)	Data type of the argument
DEFAULTED	VARCHAR(1)	Whether or not the argument is defaulted
DEFAULT_VALUE	VARCHAR(1)	Reserved for future use
DEFAULT_LENGTH	VARCHAR(1)	Reserved for future use
IN_OUT	VARCHAR(32)	Direction of the argument (IN, OUT, IN/OUT)
DATA_LENGTH	NUMBER	Length of the column(in bytes)
DATA_PRECISION	NUMBER	Length in decimal digits(NUMBER) or binary digits(FLOAT)
DATA_SCALE	NUMBER	Digits to the right of the decimal point in a number
RADIX	NUMBER	Argument radix for a number
CHARACTER_SET_NAME	VARCHAR(128)	Character set name for the argument
TYPE_OWNER	VARCHAR(128)	Owner of the type of the argument
TYPE_NAME	VARCHAR(128)	Name of the type of the argument
TYPE_SUBNAME	VARCHAR(128)	Name of the type of the argument declared in package
TYPE_LINK	VARCHAR(128)	Name of the type of the argument declared in a remote package
PLS_TYPE	VARCHAR(128)	Name of the type of the argument at PSM
CHAR_LENGTH	NUMBER	Character limit for string datatypes
CHAR_USED	VARCHAR(1)	Whether the byte limit(B) or char limit(C) is official for the string
ORIGIN_CON_ID	VARCHAR(256)	ID of the container where the data originates

USER_CATALOG

USER_CATALOG lists tables, views, synonyms, and sequences owned by the current user.

Table 9-105 Column information

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the TABLE, VIEW, SYNONYM, SEQUENCE, or UNDEFINED
TABLE_NAME	VARCHAR(128)	Name of the TABLE, VIEW, SYNONYM, SEQUENCE, or UNDEFINED
TABLE_TYPE	VARCHAR(32)	Type of the TABLE, VIEW, SYNONYM, SEQUENCE, or UNDEFINED

USER_COL_COMMENTS

USER_COL_COMMENTS displays comments on the columns of the tables and views owned by the curren t user.

Table 9-106 Column information

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Name of the column
COMMENTS	VARCHAR(1024)	Comment on the column

USER_CLUSTER_TABLES

USER_CLUSTER_TABLES describes all cluster tables owned by the current user in the cluster system.


It is available only on a cluster.

Table 9-107 Column information

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
SHARD_STRATEGY	VARCHAR(32)	Sharding strategy of the table: the value in (CLONED, HASH SHARDING, RANGE SHARDING, LIST SHARDING)
SHARD_PLACEMENT	VARCHAR(32)	Shard placement of the table: the value in (AT CLUSTER WIDE or AT CLUSTER GROUP)
SHARD_COUNT	NUMBER	Shard count of the table (if cloned table, the value is null)
SHARD_KEY_COUNT	NUMBER	Shard key column count of the table (if cloned table, the value is null)
HAS_GSI	VARCHAR(3)	Indicate whether the table has global secondary index: (YES) or (NO)
DROPPED	VARCHAR(3)	Indicates whether the table has been dropped and is in the recycle bin (YES) or not (NO)

USER_COL_PRIVS

USER_COL_PRIVS describes the column object grants for which the current user is the object owner, gran tor, or grantee.

Table 9-108 Column information

Column name	Data type	Description
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Name of the column
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
PRIVILEGE	VARCHAR(32)	Privilege on the column
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTIO N (YES) or not (NO)

USER_COL_PRIVS_MADE

USER_COL_PRIVS_MADE describes the column object grants for which the current user is the object owner.

Table 9-109 Column information

Column name	Data type	Description
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Name of the column
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
PRIVILEGE	VARCHAR(32)	Privilege on the column
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

USER_COL_PRIVS_RECD

USER_COL_PRIVS_RECD describes the column object grants for which the current user is the grantee.

Table 9-110 Column information

Column name	Data type	Description
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Name of the column
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
PRIVILEGE	VARCHAR(32)	Privilege on the column
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTIO N (YES) or not (NO)

USER_CONSTRAINTS

USER_CONSTRAINTS describes all constraint definitions on tables owned by the current user.

Table 9-111 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the constraint definition
CONSTRAINT_SCHEMA	VARCHAR(128)	Schema of the constraint definition
CONSTRAINT_NAME	VARCHAR(128)	Name of the constraint definition
CONSTRAINT_TYPE	VARCHAR(1)	Type of the constraint definition: the value in (C: check constraint, P: Primary key, U: Unique Key, R: Referential intgrity)
TABLE_OWNER	VARCHAR(128)	Owner of the table (or view) associated with the constraint definition
TABLE_SCHEMA	VARCHAR(128)	Schema of the table (or view) associated with the constraint definition
TABLE_NAME	VARCHAR(128)	Name of the table (or view) associated with the constraint definition
SEARCH_CONDITION	LONG VARCHAR	Text of search condition for a check constraint
R_OWNER	VARCHAR(128)	Owner of the unique constraint definition for the referenced table
R_SCHEMA	VARCHAR(128)	Schema of the unique constraint definition for the referenced table
R_CONSTRAINT_NAME	VARCHAR(128)	Name of the unique constraint definition for the referenced table
DELETE_RULE	VARCHAR(32)	Delete rule for a referential constraint: the value in (NO ACTION, RESTRICT, CASCADE, SET NULL, SET DEFAULT)
UPDATE_RULE	VARCHAR(32)	Update rule for a referential constraint: the value in (NO ACTION, RESTRICT, CASCADE, SET NULL, SET DEFAULT)
STATUS	VARCHAR(32)	Enforcement status of the constraint: the value in (ENABLED, DISABLE)
DEFERRABLE	VARCHAR(32)	Indicates whether the constraint is deferrable (DEFERRABLE) or not (NOT DEFERRABLE)
DEFERRED	VARCHAR(32)	Indicates whether the constraint was initially deferred (DEFERRED) or not (IMMEDIATE)
VALIDATED	VARCHAR(32)	Indicates whether all data may obey the constraint or not: the value in (VALIDATED, NOT VALIDATED)
GENERATED	VARCHAR(32)	Indicates whether the name of the constraint is user-generated (USER NAME) or system-generated (GENERATED NAME)
BAD	VARCHAR(32)	• reserved Indicates whether this constraint specifies a century in an ambiguous manner (BAD) or not (NULL)
RELY	VARCHAR(32)	• reserved When NOT VALIDATED, indicates whether the constraint is to be taken in to account for query rewrite (RELY) or not (NULL)
LAST_CHANGE	TIMESTAMP(6) WITHOUT TIME ZONE	When the constraint was last enabled or disabled

Column name	Data type	Description
INDEX_OWNER	VARCHAR(128)	Owner of the index associated with the key constraint
INDEX_SCHEMA	VARCHAR(128)	Schema of the index associated with the key constraint
INDEX_NAME	VARCHAR(128)	Name of the index associated with the key constraint
INVALID	VARCHAR(32)	Indicates whether the constraint is invalid (INVALID) or not (NULL)
VIEW_RELATED	VARCHAR(32)	• reserved Indicates whether the constraint depends on a view (DEPEND ON VIEW) o r not (NULL)
DROPPED	VARCHAR(3)	Indicates whether the constraint has been dropped and is in the recycle bi n (YES) or not (NO)
COMMENTS	VARCHAR(1024)	Comments of the constraint definition

USER_CONS_COLUMNS

USER_CONS_COLUMNS describes columns that are owned by the current user and that are specified in c onstraint definitions.

Table 9-112 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the constraint definition
CONSTRAINT_SCHEMA	VARCHAR(128)	Schema of the constraint definition
CONSTRAINT_NAME	VARCHAR(128)	Name of the constraint definition
TABLE_OWNER	VARCHAR(128)	Owner of the table with the constraint definition
TABLE_SCHEMA	VARCHAR(128)	Schema of the table with the constraint definition
TABLE_NAME	VARCHAR(128)	Name of the table with the constraint definition
COLUMN_NAME	VARCHAR(128)	Name of the column or attribute of the object type column specified in the constraint definition
POSITION	NUMBER	Original position of the column or attribute in the definition of the object

USER_DEPENDENCIES

USER_DEPENDENCIES describes dependencies between objects accessible to the current user

Table 9-113 Column information

Column name	Data type	Description
SCHEMA_NAME	VARCHAR(128)	Schema Name of object
NAME	VARCHAR(128)	Name of object
TYPE	VARCHAR(32)	Type of object: FUNCTION, PROCEDURE, VIEW, PACKAGE, PACKAGE BODY, TRIGGER
REFERENCED_OWNER	VARCHAR(128)	Owner of the referenced object
REFERENCED_SCHEMA_NAME	VARCHAR(128)	Schema Name of the referenced object
REFERENCED_TYPE	VARCHAR(32)	Type of the referenced object: FUNCTION, PROCEDURE, TABLE, VIEW, SEQUENCE, PACKAGE, PACKAGE BODY, TRIGGER
REFERENCED_LINK_NAME	VARCHAR(128)	Name of the link to the parent object
REFERENCED_NAME	VARCHAR(128)	Name of the referenced object
DEPENDENCY_TYPE	VARCHAR(32)	Indicates whether the dependency is a REF dependency (REF) or not (HARD)

USER_EXTENTS

USER_EXTENTS describes the extents comprising the segments owned by the current user's objects.

Table 9-114 Column information

Column name	Data type	Description
SEGMENT_SCHEMA	VARCHAR(128)	Schema of the segment associated with the extent
SEGMENT_NAME	VARCHAR(128)	Name of the segment associated with the extent
PARTITION_NAME	VARCHAR(128)	• reserved Object Partition Name (Set to NULL for non-partitioned objects)
SEGMENT_TYPE	VARCHAR(32)	Type of the segment: TABLE, INDEX
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the extent
EXTENT_ID	NUMBER	• reserved Extent number in the segment
BYTES	NUMBER	Size of the extent in bytes
BLOCKS	NUMBER	Size of the extent in Oracle blocks

USER_GLOBAL_SECONDARY_INDEXES

USER_GLOBAL_SECONDARY_INDEXES describes the global secondary indexes on the tables owned by th e current user.


It is available only on a cluster.

Table 9-115 Column information

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the global secondary indexed object
TABLE_NAME	VARCHAR(128)	Name of the global secondary indexed object
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the global secondary index
INI_TRANS	NUMBER	Initial number of transactions
MAX_TRANS	NUMBER	Maximum number of transactions
INITIAL_EXTENT	NUMBER	Size of the initial extent
NEXT_EXTENT	NUMBER	Size of secondary extents
MIN_EXTENTS	NUMBER	Minimum number of extents allowed in the segment
MAX_EXTENTS	NUMBER	Maximum number of extents allowed in the segment
PCT_FREE	NUMBER	Minimum percentage of free space in a block
LOGGING	VARCHAR(3)	Indicates whether or not changes to the global secondary index are l ogged: (YES) or (NO)
BLOCKS	NUMBER	Number of used blocks in the global secondary index
EMPTY_BLOCKS	NUMBER	Number of empty blocks in the global secondary index
DROPPED	VARCHAR(3)	Indicates whether the global secondary index has been dropped and is in the recycle bin (YES) or not (NO)

USER_GSI_PLACE

USER_GSI_PLACE describes node placement of all global secondary indexes on the tables owned by the current user in the cluster system.

It is available only on a cluster.

Table 9-116 Column information

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the global secondary indexed object
TABLE_NAME	VARCHAR(128)	Name of the global secondary indexed object
GROUP_ID	NUMBER	Group identifier of the node where the global secondary index placed
GROUP_NAME	VARCHAR(128)	Group name of the node where the global secondary index placed
MEMBER_ID	NUMBER	Member identifier of the node where the global secondary index placed
MEMBER_NAME	VARCHAR(128)	Member name of the node where the global secondary index placed
MEMBER_OFFLINE	BOOLEAN	data of the cluster member is offline or not
DROPPED	VARCHAR(3)	Indicates whether the global secondary index has been dropped and is in the recycle bin (YES) or not (NO)
BLOCKS	NUMBER	Number of used blocks of the node where the global secondary index placed

USER_INDEXES

USER_INDEXES describes indexes owned by the current user.

Table 9-117 Column information

Column name	Data type	Description
INDEX_SCHEMA	VARCHAR(128)	Schema of the index
INDEX_NAME	VARCHAR(128)	Name of the index
INDEX_TYPE	VARCHAR(32)	Type of the index: the value in (NORMAL, NORMAL/REV, BITMAP, FUNCTION-BASED NORMAL, FUNCTION-BASED NORMAL/REV, FUNCTION-BASED BITMAP, IOT - TOP, DOMAIN)
TABLE_OWNER	VARCHAR(128)	Owner of the indexed object
TABLE_SCHEMA	VARCHAR(128)	Schema of the indexed object
TABLE_NAME	VARCHAR(128)	Name of the indexed object
TABLE_TYPE	VARCHAR(32)	Type of the indexed object: the value in (NEXT OBJECT, INDEX, TABLE, VIEW, SYNONYM, SEQUENCE)
UNIQUENESS	VARCHAR(32)	Indicates whether the index is unique (UNIQUE) or nonunique (NONUNIQUE)
COMPRESSION	VARCHAR(32)	• reserved Indicates whether index compression is enabled (ENABLED) or not (DISABLED)
PREFIX_LENGTH	NUMBER	• reserved Number of columns in the prefix of the compression key
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the index
INI_TRANS	NUMBER	Initial number of transactions
MAX_TRANS	NUMBER	Maximum number of transactions
INITIAL_EXTENT	NUMBER	Size of the initial extent
NEXT_EXTENT	NUMBER	Size of secondary extents
MIN_EXTENTS	NUMBER	Minimum number of extents allowed in the segment
MAX_EXTENTS	NUMBER	Maximum number of extents allowed in the segment
PCT_INCREASE	NUMBER	• reserved Percentage increase in extent size
PCT_THRESHOLD	NUMBER	• reserved Threshold percentage of block space allowed per index entry
INCLUDE_COLUMN	NUMBER	• reserved Column ID of the last column to be included in index-organized table primary key (non-overflow) index
FREELISTS	NUMBER	• reserved Number of process freelists allocated to this segment
FREELIST_GROUPS	NUMBER	• reserved Number of freelist groups allocated to this segment

Column name	Data type	Description
PCT_FREE	NUMBER	Minimum percentage of free space in a block
LOGGING	VARCHAR(3)	ndicates whether or not changes to the index are logged: (YES) or (NO)
BLOCKS	NUMBER	Number of used blocks in the index
ANAL_BLOCKS	NUMBER	Number of used blocks in the index when most recently analyzed
EMPTY_BLOCKS	NUMBER	Number of empty blocks in the index
BLEVEL	NUMBER	B-Tree level (depth of the index from its root block to its leaf blocks)
LEAF_BLOCKS	NUMBER	Number of leaf blocks in the index
DISTINCT_KEYS	NUMBER	Number of distinct indexed values.
AVG_LEAF_BLOCKS_PER_KEY	NUMBER	• reserved Average number of leaf blocks in which each distinct value in the index app ears, rounded to the nearest integer
AVG_DATA_BLOCKS_PER_KEY	NUMBER	• reserved Average number of data blocks in the table that are pointed to by a distinct value in the index rounded to the nearest integer
CLUSTERING_FACTOR	NUMBER	Indicates the amount of order of the rows in the table based on the values o f the index
STATUS	VARCHAR(32)	• reserved Indicates whether a nonpartitioned index is VALID or UNUSABLE
NUM_ROWS	NUMBER	• reserved Number of rows in the index
SAMPLE_SIZE	NUMBER	Size of the sample used to analyze the index
LAST_ANALYZED	TIMESTAMP(6) WITHOUT TIME ZONE	Date on which this index was most recently analyzed
DEGREE	VARCHAR(32)	• reserved Number of threads per instance for scanning the index, or DEFAULT
INSTANCES	VARCHAR(32)	• reserved Number of instances across which the indexes to be scanned, or DEFAULT
PARTITIONED	VARCHAR(3)	• reserved Indicates whether the index is partitioned (YES) or not (NO)
TEMPORARY	VARCHAR(1)	Indicates whether the index is on a temporary table (Y) or not (N)
GENERATED	VARCHAR(1)	Indicates whether the name of the index is system-generated (Y) or not (N)
SECONDARY	VARCHAR(1)	• reserved Indicates whether the index is a secondary object created by the method of the Data Cartridge (Y) or not (N)
BUFFER_POOL	VARCHAR(32)	• reserved Buffer pool to be used for index blocks
FLASH_CACHE	VARCHAR(32)	• reserved Database Smart Flash Cache hint to be used for index blocks
CELL_FLASH_CACHE	VARCHAR(32)	• reserved Cell flash cache hint to be used for index blocks

Column name	Data type	Description
USER_STATS	VARCHAR(3)	• reserved Indicates whether statistics were entered directly by the user (YES) or not (NO)
DURATION	VARCHAR(32)	Indicates the duration of a temporary table, the value is in (TRANSACTION, SESSION)
PCT_DIRECT_ACCESS	NUMBER	• reserved For a secondary index on an index-organized table, the percentage of rows with VALID guess
ITYP_OWNER	VARCHAR(128)	• reserved For a domain index, the owner of the indextype
ITYP_NAME	VARCHAR(128)	• reserved For a domain index, the name of the indextype
PARAMETERS	VARCHAR(1024)	• reserved For a domain index, the parameter string
GLOBAL_STATS	VARCHAR(3)	• reserved For partitioned indexes, indicates whether statistics were collected by analyzing the index as a whole (YES) or were estimated from statistics on underlying index partitions and subpartitions (NO)
DOMIDX_STATUS	VARCHAR(32)	• reserved Status of a domain index
DOMIDX_OPSTATUS	VARCHAR(32)	• reserved Status of the operation on a domain index
FUNCIDX_STATUS	VARCHAR(32)	• reserved Status of a function-based index
JOIN_INDEX	VARCHAR(3)	• reserved Indicates whether the index is a join index (YES) or not (NO)
IOT_REDUNDANT_PKEY_ELIM	VARCHAR(3)	• reserved Indicates whether redundant primary key columns are eliminated from secondary indexes on index-organized tables (YES) or not (NO)
DROPPED	VARCHAR(3)	Indicates whether the index has been dropped and is in the recycle bin (YES) or not (NO)
VISIBILITY	VARCHAR(3)	• reserved Indicates whether the index is VISIBLE or INVISIBLE to the optimizer
DOMIDX_MANAGEMENT	VARCHAR(32)	• reserved If this is a domain index, indicates whether the domain index is system-managed (SYSTEM_MANAGED) or user-managed (USER_MANAGED)
SEGMENT_CREATED	VARCHAR(3)	Indicates whether the index segment has been created (YES) or not (NO)
COMMENTS	VARCHAR(1024)	Comments of the index
EMPTY_BLOCKS	NUMBER	Number of empty blocks in the index

USER_IND_COLUMNS

USER_IND_COLUMNS describes the columns of the indexes owned by the current user and columns of in dexes on tables owned by the current user.

Table 9-118 Column information

Column name	Data type	Description
INDEX_SCHEMA	VARCHAR(128)	Schema of the index
INDEX_NAME	VARCHAR(128)	Name of the index
TABLE_SCHEMA	VARCHAR(128)	Schema of the table or cluster
TABLE_NAME	VARCHAR(128)	Name of the table or cluster
COLUMN_NAME	VARCHAR(128)	Column name or attribute of the object type column
COLUMN_POSITION	NUMBER	Position of the column or attribute within the index
COLUMN_LENGTH	NUMBER	Indexed length of the column
CHAR_LENGTH	NUMBER	• reserved Maximum codepoint length of the column
DESCEND	VARCHAR(32)	Indicates whether the column is sorted in descending order (DESC) or ascending order (ASC)
NULL_ORDER	VARCHAR(32)	Indicates whether the null value of the column is sorted in nulls first order (NULLS FIRST) or nulls last order (NULLS LAST)

USER_IND_PLACE

USER_IND_PLACE describes node placement of the indexes owned by the current user in the cluster system.


It is available only on a cluster.

Table 9-119 Column information

Column name	Data type	Description
INDEX_SCHEMA	VARCHAR(128)	Schema of the index
INDEX_NAME	VARCHAR(128)	Name of the index
TABLE_OWNER	VARCHAR(128)	Owner of the indexed object
TABLE_SCHEMA	VARCHAR(128)	Schema of the indexed object
TABLE_NAME	VARCHAR(128)	Name of the indexed object
GROUP_ID	NUMBER	Group identifier of the node where the index placed
GROUP_NAME	VARCHAR(128)	Group name of the node where the index placed
MEMBER_ID	NUMBER	Member identifier of the node where the index placed
MEMBER_NAME	VARCHAR(128)	Member name of the node where the index placed
MEMBER_OFFLINE	BOOLEAN	data of the cluster member is offline or not
DROPPED	VARCHAR(3)	Indicates whether the index has been dropped and is in the recycle bin (YES) or not (NO)
DISTINCT_KEYS	NUMBER	(deprecated)
SAMPLE_SIZE	NUMBER	(deprecated)
BLOCKS	NUMBER	Number of used blocks of the node where the index placed
LAST_ANALYZED	TIMESTAMP(6) WITH OUT TIME ZONE	(deprecated)

USER_OBJECTS

USER_OBJECTS describes all objects owned by the current user.

Table 9-120 Column information

Column name	Data type	Description
SCHEMA_NAME	VARCHAR(128)	Schema of the object
OBJECT_NAME	VARCHAR(128)	Name of the object
SUBOBJECT_NAME	VARCHAR(128)	Name of the subobject (for example, partition)
OBJECT_ID	NUMBER	Dictionary object number of the object
DATA_OBJECT_ID	NUMBER	Dictionary object number of the segment that contains the object
OBJECT_TYPE	VARCHAR(32)	Type of the object (such as TABLE, INDEX)
CREATED	TIMESTAMP(6) WITHOUT TIME ZONE	Timestamp for the creation of the object
LAST_DDL_TIME	TIMESTAMP(6) WITHOUT TIME ZONE	Timestamp for the last modification of the object resulting from a DDL statement
TIMESTAMP	VARCHAR(32)	Timestamp for the specification of the object (character data)
STATUS	VARCHAR(32)	• reserved Status of the object: the value in (VALID, INVALID, N/A)
TEMPORARY	VARCHAR(1)	Indicates whether the object is temporary (the current session can see only data that it placed in this object itself) (Y) or not (N)
GENERATED	VARCHAR(1)	Indicates whether the name of this object was system-generated (Y) or not (N)
SECONDARY	VARCHAR(1)	• reserved Indicates whether this is a secondary object created by the ODCIIndex Create method of the Oracle Data Cartridge (Y) or not (N)
NAMESPACE	NUMBER	Namespace for the object
EDITION_NAME	VARCHAR(128)	• reserved Name of the edition in which the object is actual
DROPPED	VARCHAR(3)	Indicates whether the object has been dropped and is in the recycle bin (YES) or not (NO)

USER_PACKAGE_PRIVS

USER_PACKAGE_PRIVS describes the package grants for which the current user is the package owner, gr antor, or grantee.

Table 9-121 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
PROCEDURE_OWNER	VARCHAR(128)	Owner of the package
PROCEDURE_SCHEMA	VARCHAR(128)	Schema of the package
PROCEDURE_NAME	VARCHAR(128)	Name of the package
PRIVILEGE	VARCHAR(32)	Privilege on the package
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTI ON (YES) or not (NO)

USER_PACKAGE_PRIVS_MADE

USER_PACKAGE_PRIVS_MADE describes the package grants for which the current user is the package owner.

Table 9-122 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
PROCEDURE_OWNER	VARCHAR(128)	Owner of the package
PROCEDURE_SCHEMA	VARCHAR(128)	Schema of the package
PROCEDURE_NAME	VARCHAR(128)	Name of the package
PRIVILEGE	VARCHAR(32)	Privilege on the package
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

USER_PACKAGE_PRIVS_RECD

USER_PACKAGE_PRIVS_RECD describes the package grants for which the current user is the grantee.

Table 9-123 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
PROCEDURE_OWNER	VARCHAR(128)	Owner of the package
PROCEDURE_SCHEMA	VARCHAR(128)	Schema of the package
PROCEDURE_NAME	VARCHAR(128)	Name of the package
PRIVILEGE	VARCHAR(32)	Privilege on the package
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

USER_PROCEDURES

USER_PROCEDURES lists of procedures owned by the current user.

Table 9-124 Column information

Column name	Data type	Description
SCHEMA_NAME	VARCHAR(128)	Schema Name of function, procedures or package
OBJECT_NAME	VARCHAR(128)	Name of function, procedures or pacakage
PROCEDURE_NAME	VARCHAR(128)	Name when a procedures in pacakage
OBJECT_ID	NUMBER	ID of a function, procedures or pacakage
SUBPROGRAM_ID	NUMBER	ID of procedures in pacakage
OVERLOAD	VARCHAR(32)	ID of overloading procedure in pacakage
OBJECT_TYPE	VARCHAR(32)	Type of function, procedures or package
AGGREGATE	VARCHAR(3)	Indicate whether the procedure is an aggreage function(YES) or not (NO)
PIPELINED	VARCHAR(3)	Indicate whether the procedure is a pipelined table function(YES) or not(NO)
IMPLTYPEOWNER	VARCHAR(128)	Name of the owner of the implementation type, if any
IMPLTYPENAME	VARCHAR(128)	Name of the implementation type, if any
PARALLEL	VARCHAR(3)	Indicates whether the procedure or function is parallel-enabled (YES) or not (NO)
INTERFACE	VARCHAR(3)	YES, if the procedure/function is a table function implemented usin g the SQLCLI interface; otherwise NO
DETERMINISTIC	VARCHAR(3)	YES, if the procedure/function is declared to be deterministic; other wise NO
AUTHID	VARCHAR(32)	Indicates whether the procedure/function is declared to execute as DEFINER or CURRENT_USER (invoker)

USER_PROC_PRIVS

USER_PROC_PRIVS describes the procedure grants for which the current user is the procedure owner, gr antor, or grantee.

Table 9-125 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
PROCEDURE_OWNER	VARCHAR(128)	Owner of the procedure and function
PROCEDURE_SCHEMA	VARCHAR(128)	Schema of the procedure and function
PROCEDURE_NAME	VARCHAR(128)	Name of the procedure and function
PRIVILEGE	VARCHAR(32)	Privilege on the procedure and function
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTI ON (YES) or not (NO)

USER_PROC_PRIVS_MADE

USER_PROC_PRIVS_MADE describes the procedure grants for which the current user is the procedure owner or grantor.

Table 9-126 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
PROCEDURE_OWNER	VARCHAR(128)	Owner of the procedure and function
PROCEDURE_SCHEMA	VARCHAR(128)	Schema of the procedure and function
PROCEDURE_NAME	VARCHAR(128)	Name of the procedure and function
PRIVILEGE	VARCHAR(32)	Privilege on the procedure and function
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

USER_PROC_PRIVS_RECD

USER_PROC_PRIVS_RECD describes the procedure grants, for which the current user is the grantee, or fo r which an enabled role or PUBLIC is the grantee.

Table 9-127 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
PROCEDURE_OWNER	VARCHAR(128)	Owner of the procedure and function
PROCEDURE_SCHEMA	VARCHAR(128)	Schema of the procedure and function
PROCEDURE_NAME	VARCHAR(128)	Name of the procedure and function
PRIVILEGE	VARCHAR(32)	Privilege on the procedure and function
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTI ON (YES) or not (NO)

USER_RECYCLEBIN

USER_RECYCLEBIN describes recycle bins owned by the current user.

Table 9-128 Column information

Column name	Data type	Description
SCHEMA_NAME	VARCHAR(128)	Schema name of the object
OBJECT_NAME	VARCHAR(128)	Name of the object
ORIGINAL_NAME	VARCHAR(128)	Original name of the object
OPERATION	VARCHAR(4)	Operation carried out on the object
OBJECT_TYPE	VARCHAR(32)	Type of the object
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the object
CREATED_TIME	TIMESTAMP(6) WITH OUT TIME ZONE	Created time of the object
DROPPED_TIME	TIMESTAMP(6) WITH OUT TIME ZONE	Dropped time of the object
DROP_SCN	VARCHAR(128)	System change number (SCN) of the transaction which moved the object to the recycle bin
DROP_GCN	NUMBER	Global change number (GCN) of the transaction which moved the object to the recycle bin
DROP_DCN	NUMBER	Domain change number (DCN) of the transaction which moved the object to the recycle bin
DROP_LCN	NUMBER	Local change number (LCN) of the transaction which moved the object to the recycle bin
CAN_UNDROP	VARCHAR(3)	Indicates whether the object can be undropped (YES) or not (NO)
CAN_PURGE	VARCHAR(3)	Indicates whether the object can be purged (YES) or not (NO)
BASE_OBJECT	NUMBER	Object number of the base object
PURGE_OBJECT	NUMBER	Object number for the object which gets purged

USER_SCHEMAS

Identify the schemata in a catalog that are owned by current user.

Table 9-129 Column information

Column name	Data type	Description
SCHEMA_OWNER	VARCHAR(128)	Owner of the schema
SCHEMA_NAME	VARCHAR(128)	Name of the schema
CREATED_TIME	TIMESTAMP(6) WITHOUT TIME ZONE	Created time of the schema
MODIFIED_TIME	TIMESTAMP(6) WITHOUT TIME ZONE	Last modified time of the schema
COMMENTS	VARCHAR(1024)	Comments of the schema

USER_SCHEMA_PATH

USER_SCHEMA_PATH describes the schema search order of the current user, for naming resolution of un qualified SQL schema objects.

Table 9-130 Column information

Column name	Data type	Description
AUTH_NAME	VARCHAR(128)	Name of the user
SCHEMA_NAME	VARCHAR(128)	Name of the schema
SEARCH_ORDER	NUMBER	Schema search order of the user

USER_SCHEMA_PRIVS

USER_SCHEMA_PRIVS describes the schema grants, for which the current user is the schema owner, gra ntor, or grantee.

Table 9-131 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the schema
SCHEMA_NAME	VARCHAR(128)	Name of the schema
PRIVILEGE	VARCHAR(32)	Privilege on the schema
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTIO N (YES) or not (NO)

USER_SCHEMA_PRIVS_MADE

USER_SCHEMA_PRIVS_MADE describes the schema grants for which the current user is the schema owner.

Table 9-132 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the schema
SCHEMA_NAME	VARCHAR(128)	Name of the schema
PRIVILEGE	VARCHAR(32)	Privilege on the schema
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

USER_SCHEMA_PRIVS_RECD

USER_SCHEMA_PRIVS_RECD describes the schema grants for which the current user is the grantee.

Table 9-133 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the schema
SCHEMA_NAME	VARCHAR(128)	Name of the schema
PRIVILEGE	VARCHAR(32)	Privilege on the schema
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

USER_SEQUENCES

USER_SEQUENCES describes all sequences owned by the current user.

Table 9-134 Column information

Column name	Data type	Description
SEQUENCE_OWNER	VARCHAR(128)	Owner of the sequence
SEQUENCE_SCHEMA	VARCHAR(128)	Schema of the sequence
SEQUENCE_NAME	VARCHAR(128)	Sequence name
MIN_VALUE	NUMBER	Minimum value of the sequence
MAX_VALUE	NUMBER	Maximum value of the sequence
INCREMENT_BY	NUMBER	Value by which sequence is incremented
CYCLE_FLAG	VARCHAR(1)	Indicates whether the sequence wraps around on reaching the limit (Y) or not (N)
ORDER_FLAG	VARCHAR(1)	• reserved Indicates whether sequence numbers are generated in order (Y) or not (N)
CACHE_SIZE	NUMBER	Number of sequence numbers to cache
LAST_NUMBER	NUMBER	Last sequence number written to database. If a sequence uses caching, the number written to database is the last number placed in the sequence cache.
COMMENTS	VARCHAR(1024)	Comments of the sequence

USER_SEQ_PRIVS

USER_SEQ_PRIVS describes the sequence grants for which the current user is the sequence owner, grant or, or grantee.

Table 9-135 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
SEQUENCE_OWNER	VARCHAR(128)	Owner of the sequence
SEQUENCE_SCHEMA	VARCHAR(128)	Schema of the sequence
SEQUENCE_NAME	VARCHAR(128)	Name of the sequence
PRIVILEGE	VARCHAR(32)	Privilege on the sequence
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

USER_SEQ_PRIVS_MADE

USER_SEQ_PRIVS_MADE describes the sequence grants for which the current user is the sequence owner.

Table 9-136 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
SEQUENCE_OWNER	VARCHAR(128)	Owner of the sequence
SEQUENCE_SCHEMA	VARCHAR(128)	Schema of the sequence
SEQUENCE_NAME	VARCHAR(128)	Name of the sequence
PRIVILEGE	VARCHAR(32)	Privilege on the sequence
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

USER_SEQ_PRIVS_RECD

USER_SEQ_PRIVS_RECD describes the sequence grants for which the current user is the grantee.

Table 9-137 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
SEQUENCE_OWNER	VARCHAR(128)	Owner of the sequence
SEQUENCE_SCHEMA	VARCHAR(128)	Schema of the sequence
SEQUENCE_NAME	VARCHAR(128)	Name of the sequence
PRIVILEGE	VARCHAR(32)	Privilege on the sequence
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)

USER_SHARD_KEY_COLUMNS

USER_SHARD_KEY_COLUMNS describes shard key columns of shareded tables owned by the current use r in the cluster system.

It is available only on a cluster.

Table 9-138 Column information

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
COLUMN_NAME	VARCHAR(128)	Column name of the shard key
COLUMN_POSITION	NUMBER	Position of the column within the shard key

USER_SOURCE

USER_SOURCE describes the text source of the stored objects accessible to the current user.

Table 9-139 Column information

Column name	Data type	Description
SCHEMA_NAME	VARCHAR(128)	Schema Name of object
NAME	VARCHAR(128)	Name of object
TYPE	VARCHAR(32)	Type of object: FUNCTION, PROCEDURE, PACKAGE, PACKAGE BODY, TRIGGER
LINE	NUMBER	Line number of this line of source
TEXT	LONG VARCHAR	Text source of the strored object
ORIGIN_CON_ID	VARCHAR(256)	ID of the container where the data originates

USER_SYNONYMS

USER_SYNONYMS describes all synonyms owned by the current user.

Table 9-140 Column information

Column name	Data type	Description
SYNONYM_OWNER	VARCHAR(128)	Owner of the synonym
SYNONYM_SCHEMA	VARCHAR(128)	Schema of the synonym
SYNONYM_NAME	VARCHAR(128)	Synonym name
OBJECT_SCHEMA_NAME	VARCHAR(128)	Object schema name
OBJECT_NAME	VARCHAR(128)	Object name
DB_LINK	VARCHAR(128)	Reserved for future use

USER_SYS_PRIVS

USER_SYS_PRIVS describes system (database, tablespace, schema) privileges granted to the current user or PUBLIC.

Table 9-141 Column information

Column name	Data type	Description
USERNAME	VARCHAR(128)	Name of the user, or PUBLIC
PRIVILEGE	VARCHAR(256)	System(database, tablespace, schema) privilege
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTIO N (YES) or not (NO)
ADMIN_OPTION	VARCHAR(3)	equal to GRANTABLE column

USER_TABLES

USER_TABLES describes the relational tables owned by the current user.

Table 9-142 Column information

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace containing the table
CLUSTER_NAME	VARCHAR(128)	• reserved Name of the cluster
IOT_NAME	VARCHAR(128)	• reserved Name of the index-organized table
STATUS	VARCHAR(32)	• reserved If a previous DROP TABLE operation failed, indicates whether the table is unusable (UNUSABLE) or valid (VALID)
PCT_FREE	NUMBER	Minimum percentage of free space in a block
PCT_USED	NUMBER	Minimum percentage of used space in a block
INI_TRANS	NUMBER	Initial number of transactions
MAX_TRANS	NUMBER	Maximum number of transactions
INITIAL_EXTENT	NUMBER	Size of the initial extent (in bytes)
NEXT_EXTENT	NUMBER	Size of secondary extents (in bytes)
MIN_EXTENTS	NUMBER	Minimum number of extents allowed in the segment
MAX_EXTENTS	NUMBER	Maximum number of extents allowed in the segment
PCT_INCREASE	NUMBER	• reserved Percentage increase in extent size
FREELISTS	NUMBER	• reserved Number of process freelists allocated to the segment
FREELIST_GROUPS	NUMBER	• reserved Number of freelist groups allocated to the segment
LOGGING	VARCHAR(3)	Indicates whether or not changes to the table are logged
BACKED_UP	VARCHAR(1)	• reserved Indicates whether the table has been backed up since the last modification (Y) or not (N)
NUM_ROWS	NUMBER	Number of rows in the table
BLOCKS	NUMBER	Number of used blocks in the table
ANAL_BLOCKS	NUMBER	Number of used blocks in the table when most recently analyzed
EMPTY_BLOCKS	NUMBER	• reserved Number of empty (never used) blocks in the table
AVG_SPACE	NUMBER	• reserved Average available free space in the table

Column name	Data type	Description
CHAIN_CNT	NUMBER	• reserved Number of rows in the table that are chained from one data block to another or that have migrated to a new block, requiring a link to preserve the old rowid
AVG_ROW_LEN	NUMBER	• reserved Average row length, including row overhead
AVG_SPACE_FREELIST_BLOCKS	NUMBER	• reserved Average freespace of all blocks on a freelist
NUM_FREELIST_BLOCKS	NUMBER	• reserved Number of blocks on the freelist
DEGREE	VARCHAR(32)	• reserved Number of threads per instance for scanning the table, or DEFAULT
INSTANCES	VARCHAR(32)	• reserved Number of instances across which the table is to be scanned, or DEFAULT
CACHE	VARCHAR(1)	• reserved Indicates whether the table is to be cached in the buffer cache (Y) or not (N)
TABLE_LOCK	VARCHAR(32)	Indicates whether table locking is enabled (ENABLED) or disabled (DISABLED)
SAMPLE_SIZE	NUMBER	Sample size used in analyzing the table
LAST_ANALYZED	TIMESTAMP(6) WITHOUT TIME ZONE	Date on which the table was most recently analyzed
PARTITIONED	VARCHAR(3)	• reserved Indicates whether the table is partitioned (YES) or not (NO)
IOT_TYPE	VARCHAR(32)	• reserved If the table is an index-organized table, then IOT_TYPE is IOT, IOT_OVERFLOW, or IOT_MAPPING.
TEMPORARY	VARCHAR(1)	Indicates whether the table is temporary (Y) or not (N)
SECONDARY	VARCHAR(1)	• reserved Indicates whether the table is a secondary object created by cartridge
NESTED	VARCHAR(3)	• reserved Indicates whether the table is a nested table (YES) or not (NO)
BUFFER_POOL	VARCHAR(32)	• reserved Buffer pool to be used for table blocks
FLASH_CACHE	VARCHAR(32)	• reserved Database Smart Flash Cache hint to be used for table blocks
CELL_FLASH_CACHE	VARCHAR(32)	• reserved Cell flash cache hint to be used for table blocks
ROW_MOVEME	VARCHAR(32)	• reserved If a partitioned table, indicates whether row movement is enabled (ENABLE

Column name	Data type	Description
NT		D) or disabled (DISABLED)
GLOBAL_STATS	VARCHAR(3)	• reserved For partitioned tables, indicates whether statistics for the table as a whole (global statistics) are accurate (YES)
USER_STATS	VARCHAR(3)	• reserved Indicates whether statistics were entered directly by the user (YES) or not (NO)
DURATION	VARCHAR(32)	Indicates the duration of a temporary table, the value is in (TRANSACTION, SESSION)
SKIP_CORRUPT	VARCHAR(32)	• reserved Indicates whether Database ignores blocks marked corrupt during table an d index scans (ENABLED) or raises an error (DISABLED)
MONITORING	VARCHAR(3)	• reserved Indicates whether the table has the MONITORING attribute set (YES) or not (NO)
CLUSTER_OWNE R	VARCHAR(128)	• reserved Owner of the cluster, if any
DEPENDENCIES	VARCHAR(32)	• reserved Indicates whether row-level dependency tracking is enabled (ENABLED) or disabled (DISABLED)
COMPRESSION	VARCHAR(32)	• reserved Indicates whether table compression is enabled (ENABLED) or not (DISABLE D)
COMPRESS_FOR	VARCHAR(32)	• reserved Default compression for what kind of operations
DROPPED	VARCHAR(3)	Indicates whether the table has been dropped and is in the recycle bin (YES) or not (NO)
READ_ONLY	VARCHAR(3)	Indicates whether the table IS READ-ONLY (YES) or not (NO)
SEGMENT_CREA TED	VARCHAR(3)	Indicates whether the table segment has been created (YES) or not (NO)
RESULT_CACHE	VARCHAR(32)	• reserved Result cache mode annotation for the table: the value in (NULL, DEFAULT, FORCE, MANUAL)

USER_TABLESPACES

USER_TABLESPACES describes the tablespaces accessible to the current user.

Table 9-143 Column information

Column name	Data type	Description
TABLESPACE_NAME	VARCHAR(128)	Name of the tablespace
BLOCK_SIZE	NUMBER	Tablespace block size
INITIAL_EXTENT	NUMBER	• reserved Default initial extent size (in bytes)
NEXT_EXTENT	NUMBER	• reserved Default incremental extent size (in bytes)
MIN_EXTENTS	NUMBER	• reserved Default minimum number of extents
MAX_EXTENTS	NUMBER	• reserved Default maximum number of extents
MAX_SIZE	NUMBER	• reserved Default maximum size of segments
PCT_INCREASE	NUMBER	• reserved Default percent increase for extent size
MIN_EXTLEN	NUMBER	• reserved Minimum extent size for this tablespace (in bytes)
STATUS	VARCHAR(32)	Tablespace status: the value in (ONLINE, OFFLINE, READ ONLY)
CONTENTS	VARCHAR(32)	Tablespace contents: the value in (SYSTEM, DATA, TEMPORARY, UNDO)
LOGGING	VARCHAR(32)	Default logging attribute: LOGGING, NOLOGGING
FORCE_LOGGING	VARCHAR(3)	• reserved Indicates whether the tablespace is under force logging mode (YES) or not (NO)
EXTENT_MANAGEMENT	VARCHAR(32)	• reserved Indicates whether the extents in the tablespace are dictionary managed (DICTIONARY) or locally managed (LOCAL)
ALLOCATION_TYPE	VARCHAR(32)	• reserved Type of extent allocation in effect for the tablespace: the value in (SYSTEM, UNIFORM, USER)
SEGMENT_SPACE_MANAGEMENT	VARCHAR(32)	• reserved Indicates whether the free and used segment space in the tablespace is managed using free lists (MANUAL) or bitmaps (AUTO)
DEF_TAB_COMPRESSION	VARCHAR(32)	• reserved Indicates whether default table compression is enabled (ENABLED) or not (DISABLED)
		• reserved

Column name	Data type	Description
RETENTION	VARCHAR(32)	Undo tablespace retention: the value in (GUARANTEE, NOGUARANTEE, NOT APPLY)
BIGFILE	VARCHAR(3)	• reserved Indicates whether the tablespace is a bigfile tablespace (YES) or a smallfile tablespace (NO)
PREDICATE_EVALUATION	VARCHAR(32)	• reserved Indicates whether predicates are evaluated by host (HOST) or by storage (STORAGE)
ENCRYPTED	VARCHAR(3)	• reserved Indicates whether the tablespace is encrypted (YES) or not (NO)
COMPRESS_FOR	VARCHAR(32)	• reserved Indicates whether the tablespace is encrypted (YES) or not (NO)

USER_TAB_COLS

USER_TAB_COLS describes the columns (including hidden columns) of the tables, views, and clusters owned by the current user.

Table 9-144 Column information

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Column name
DATA_TYPE	VARCHAR(128)	Datatype of the column
DATA_TYPE_MOD	VARCHAR(3)	• reserved Datatype modifier of the column
DATA_TYPE_OWNER	VARCHAR(128)	• reserved Owner of the datatype of the column
DATA_LENGTH	NUMBER	Length of the column (in bytes)
DATA_PRECISION	NUMBER	Decimal precision for NUMBER datatype; binary precision for FLOAT datatype; NULL for all other datatypes
DATA_SCALE	NUMBER	Digits to the right of the decimal point in a number
NULLABLE	VARCHAR(1)	Indicates whether a column allows NULLs.
COLUMN_ID	NUMBER	Sequence number of the column as created
DEFAULT_LENGTH	NUMBER	Length of the default value for the column
DATA_DEFAULT	LONG VARCHAR	Default value for the column
NUM_DISTINCT	NUMBER	Number of distinct values in the column
LOW_VALUE	VARBINARY(32)	Low value in the column
HIGH_VALUE	VARBINARY(32)	High value in the column
DENSITY	NUMBER	• reserved If a histogram is available on COLUMN_NAME, then this column displays the selectivity of a value that spans fewer than 2 endpoints in the histogram.
NUM_NULLS	NUMBER	Number of NULLs in the column
NUM_BUCKETS	NUMBER	• reserved Number of buckets in the histogram for the column
LAST_ANALYZED	TIMESTAMP(6) WITHOUT TIME ZONE	Date on which this column was most recently analyzed
SAMPLE_SIZE	NUMBER	Sample size used in analyzing this column
CHARACTER_SET_NAME	VARCHAR(128)	• reserved Name of the character set
CHAR_COL_DECL_LENGTH	NUMBER	Declaration length of the character type column
GLOBAL_STATS	VARCHAR(3)	• reserved For partitioned tables, indicates whether column statistics were collected for the table

Column name	Data type	Description
USER_STATS	VARCHAR(3)	• reserved Indicates whether statistics were entered directly by the user (YES) or not (NO)
AVG_COL_LEN	NUMBER	Average length of the column (in bytes)
CHAR_LENGTH	NUMBER	Displays the length of the column in characters.
CHAR_USED	VARCHAR(1)	Indicates that the column uses BYTE length semantics (B) or C HAR length semantics (C)
V80_FMT_IMAGE	VARCHAR(3)	• reserved Indicates whether the column data is in release older image f ormat (YES) or not (NO)
DATA_UPGRADED	VARCHAR(3)	• reserved Indicates whether the column data has been upgraded to the latest type version format (YES) or not (NO)
HIDDEN_COLUMN	VARCHAR(3)	Indicates whether the column is a hidden column (YES) or not (NO)
VIRTUAL_COLUMN	VARCHAR(3)	• reserved Indicates whether the column is a virtual column (YES) or not (NO)
SEGMENT_COLUMN_ID	NUMBER	Sequence number of the column in the segment
INTERNAL_COLUMN_ID	NUMBER	Internal sequence number of the column
HISTOGRAM	VARCHAR(32)	• reserved Indicates existence/type of histogram
QUALIFIED_COL_NAME	VARCHAR(4000)	Qualified column name
IDENTITY_COLUMN	VARCHAR(3)	Indicates whether this is an identity column (YES) or not (NO)

USER_TAB_COLUMNS

USER_TAB_COLUMNS describes the columns of the tables, views, and clusters owned by the current user.

Table 9-145 Column information

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COLUMN_NAME	VARCHAR(128)	Column name
DATA_TYPE	VARCHAR(128)	Datatype of the column
DATA_TYPE_MOD	VARCHAR(3)	• reserved Datatype modifier of the column
DATA_TYPE_OWNER	VARCHAR(128)	• reserved Owner of the datatype of the column
DATA_LENGTH	NUMBER	Length of the column (in bytes)
DATA_PRECISION	NUMBER	Decimal precision for NUMBER datatype; binary precision for FLOAT datatype; NULL for all other datatypes
DATA_SCALE	NUMBER	Digits to the right of the decimal point in a number
NULLABLE	VARCHAR(1)	Indicates whether a column allows NULLs.
COLUMN_ID	NUMBER	Sequence number of the column as created
DEFAULT_LENGTH	NUMBER	Length of the default value for the column
DATA_DEFAULT	LONG VARCHAR	Default value for the column
NUM_DISTINCT	NUMBER	Number of distinct values in the column
LOW_VALUE	VARBINARY(32)	Low value in the column
HIGH_VALUE	VARBINARY(32)	High value in the column
DENSITY	NUMBER	• reserved If a histogram is available on COLUMN_NAME, then this column displays the selectivity of a value that spans fewer than 2 endpoints in the histogram.
NUM_NULLS	NUMBER	Number of NULLs in the column
NUM_BUCKETS	NUMBER	• reserved Number of buckets in the histogram for the column
LAST_ANALYZED	TIMESTAMP(6) WITHOUT TIME ZONE	Date on which this column was most recently analyzed
SAMPLE_SIZE	NUMBER	Sample size used in analyzing this column
CHARACTER_SET_NAME	VARCHAR(128)	• reserved Name of the character set
CHAR_COL_DECL_LENGTH	NUMBER	Declaration length of the character type column
GLOBAL_STATS	VARCHAR(3)	• reserved For partitioned tables, indicates whether column statistics were collected for the table

Column name	Data type	Description
USER_STATS	VARCHAR(3)	• reserved Indicates whether statistics were entered directly by the user (YES) or not (NO)
AVG_COL_LEN	NUMBER	Average length of the column (in bytes)
CHAR_LENGTH	NUMBER	Displays the length of the column in characters.
CHAR_USED	VARCHAR(1)	Indicates that the column uses BYTE length semantics (B) or CHAR l ength semantics (C)
V80_FMT_IMAGE	VARCHAR(3)	• reserved Indicates whether the column data is in release older image format (YES) or not (NO)
DATA_UPGRADED	VARCHAR(3)	• reserved Indicates whether the column data has been upgraded to the latest type version format (YES) or not (NO)
HISTOGRAM	VARCHAR(32)	• reserved Indicates existence/type of histogram
IDENTITY_COLUMN	VARCHAR(3)	Indicates whether this is an identity column (YES) or not (NO)

USER_TAB_COMMENTS

USER_TAB_COMMENTS displays comments on the tables and views owned by the current user.

Table 9-146 Column information

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
TABLE_TYPE	VARCHAR(32)	Type of the object
COMMENTS	VARCHAR(1024)	Comment on the object

USER_TAB_IDENTITY_COLS

USER_TAB_IDENTITY_COLS describes all table identity columns.

Table 9-147 Column information

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
COLUMN_NAME	VARCHAR(128)	Name of the identity column
GENERATION_TYPE	VARCHAR(32)	Generation type of the identity column. Possible values are ALWAYS or BY DEFAULT
IDENTITY_OPTIONS	VARCHAR(1024)	Options for the identity column sequence generator

USER_TAB_PLACE

USER_TAB_PLACE describes node placement of cluster tables owned by the current user in the cluster sys tem.


It is available only on a cluster.

Table 9-148 Column information

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
GROUP_ID	NUMBER	Group identifier of the node where the table placed
GROUP_NAME	VARCHAR(128)	Group name of the node where the table placed
MEMBER_ID	NUMBER	Member identifier of the node where the table placed
MEMBER_NAME	VARCHAR(128)	Member name of the node where the table placed
MEMBER_POSITION	NUMBER	Member position of the node where the table placed
MEMBER_OFFLINE	BOOLEAN	data of the cluster member is offline or not
IS_UPDATE_MASTER	BOOLEAN	whether the cluster member is update master or not
DROPPED	VARCHAR(3)	Indicates whether the table has been dropped and is in the recycle b in (YES) or not (NO)
SCN	VARCHAR(64)	table scn of the node where the table placed
NUM_ROWS	NUMBER	Number of rows in the table
BLOCKS	NUMBER	Number of used blocks of the node where the table placed
LAST_ANALYZED	TIMESTAMP(6) WITH OUT TIME ZONE	Date on which the table was most recently analyzed

USER_TAB_PRIVS

USER_TAB_PRIVS describes the object grants for which the current user is the object owner, grantor, or g rantee.

Table 9-149 Column information

Column name	Data type	Description
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
PRIVILEGE	VARCHAR(32)	Privilege on the object
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTIO N (YES) or not (NO)
HIERARCHY	VARCHAR(3)	Indicates whether the privilege was granted with the HIERARCHY O PTION (YES) or not (NO)

USER_TAB_PRIVS_MADE

USER_TAB_PRIVS_MADE describes the object grants for which the current user is the object owner.

Table 9-150 Column information

Column name	Data type	Description
GRANTEE	VARCHAR(128)	Name of the user or role to whom access was granted
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
PRIVILEGE	VARCHAR(32)	Privilege on the object
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTIO N (YES) or not (NO)
HIERARCHY	VARCHAR(3)	Indicates whether the privilege was granted with the HIERARCHY O PTION (YES) or not (NO)

USER_TAB_PRIVS_RECD

USER_TAB_PRIVS_RECD describes the object grants for which the current user is the grantee.

Table 9-151 Column information

Column name	Data type	Description
OWNER	VARCHAR(128)	Owner of the object
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
GRANTOR	VARCHAR(128)	Name of the user who performed the grant
PRIVILEGE	VARCHAR(32)	Privilege on the object
GRANTABLE	VARCHAR(3)	Indicates whether the privilege was granted with the GRANT OPTION (YES) or not (NO)
HIERARCHY	VARCHAR(3)	Indicates whether the privilege was granted with the HIERARCHY OPTION (YES) or not (NO)

USER_TAB_SHARDS

USER_TAB_SHARDS describes shard information of sharded tables owned by the current user in the clust er system.


It is available only on a cluster.

Table 9-152 Column information

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the table
TABLE_NAME	VARCHAR(128)	Name of the table
SHARD_STRATEGY	VARCHAR(32)	Sharding strategy of the table: the value in (HASH SHARDING, RANGE SHARDING, LIST SHARDING)
SHARD_NAME	VARCHAR(128)	Shard name
SHARD_NUMBER	NUMBER	Shard number
SHARD_DEFINITION	LONG VARCHAR	Shard definition (if hash sharded, the value is null)
GROUP_ID	NUMBER	Group identifier where the shard placed
GROUP_NAME	VARCHAR(128)	Group Name where the shard placed
DROPPED	VARCHAR(3)	Indicates whether the table has been dropped and is in the recycle bin (YES) or not (NO)

USER_USERS

USER_USERS describes the current user.

Table 9-153 Column information

Column name	Data type	Description
USERNAME	VARCHAR(128)	Name of the user
USER_ID	NUMBER	ID number of the user
ACCOUNT_STATUS	VARCHAR(32)	Account status: the value in (OPEN, EXPIRED, EXPIRED(GRACE), LOCKED(TIMED), LOCKED, EXPIRED & LOCKED(TIMED), EXPIRED(GRACE) & LOCKED(TIMED), EXPIRED & LOCKED, EXPIRED(GRACE) & LOCKED)
LOCK_DATE	TIMESTAMP(6) WITHOUT TIME ZONE	Timestamp the account was locked if account status was LOCKED
EXPIRY_DATE	TIMESTAMP(6) WITHOUT TIME ZONE	Timestamp of expiration of the account
DEFAULT_TABLESPACE	VARCHAR(128)	Default tablespace for data
TEMPORARY_TABLESPACE	VARCHAR(128)	Name of the default tablespace for temporary tables or the name of a tablespace group
INDEX_TABLESPACE	VARCHAR(128)	Default tablespace for index
CREATED	TIMESTAMP(6) WITHOUT TIME ZONE	User creation timestamp
INITIAL_RSRC_CONSUMER_GROUP	VARCHAR(128)	• reserved Initial resource consumer group for the user
EXTERNAL_NAME	VARCHAR(128)	• reserved User external name

USER_VIEWS

USER_VIEWS describes the views owned by the current user.

Table 9-154 Column information

Column name	Data type	Description
VIEW_SCHEMA	VARCHAR(128)	Schema of the view
VIEW_NAME	VARCHAR(128)	Name of the view
TEXT_LENGTH	NUMBER	Length of the view text
TEXT	LONG VARCHAR	View text
TYPE_TEXT_LENGTH	NUMBER	• reserved Length of the type clause of the typed view
TYPE_TEXT	VARCHAR(4000)	• reserved Type clause of the typed view
OID_TEXT_LENGTH	NUMBER	• reserved Length of the WITH OID clause of the typed view
OID_TEXT	VARCHAR(4000)	• reserved WITH OID clause of the typed view
VIEW_TYPE_OWNER	VARCHAR(128)	• reserved Owner of the type of the view if the view is a typed view
VIEW_TYPE	VARCHAR(32)	• reserved Type of the view if the view is a typed view
SUPERVIEW_NAME	VARCHAR(128)	• reserved Name of the superview
EDITIONING_VIEW	VARCHAR(1)	Reserved for future use
READ_ONLY	VARCHAR(1)	Indicates whether the view is read-only (Y) or not (N)

Other Views

There are other views or tables which are none of All-family, DBA-family or USER-family.

AUDIT_POLICIES

AUDIT_POLICIES contains one row for each audit policy.

Table 9-155 Column information

Column name	Data type	Description
POLICY_NAME	VARCHAR(128)	audit policy name
ENABLED	VARCHAR(3)	indicates whether the audit policy is enabled (YES) or not (NO)
CREATED_TIME	TIMESTAMP(6) WITH OUT TIME ZONE	created time of the audit policy
MODIFIED_TIME	TIMESTAMP(6) WITH OUT TIME ZONE	last modified time of the audit policy
COMMENTS	VARCHAR(1024)	comments of the audit policy

AUDIT_POLICY_OPTIONS

AUDIT_POLICY_OPTIONS describes all audit policies created in the database.

Table 9-156 Column information

Column name	Data type	Description
POLICY_NAME	VARCHAR(128)	audit policy name
AUDIT_OPTION	VARCHAR(32)	auditing option defined in the audit policy
AUDIT_OPTION_TYPE	VARCHAR(32)	The values of AUDIT_OPTION_TYPE_NAME in ('DATABASE PRIVILEGE', 'SYSTEM ACTION', 'OBJECT ACTION')
OBJECT_SCHEMA	VARCHAR(128)	schema name, for an object-specific auditing option
OBJECT_NAME	VARCHAR(128)	object name, for an object-specific auditing option
OBJECT_TYPE	VARCHAR(32)	object type name, for an object-specific auditing option
DROPPED	VARCHAR(3)	Indicates whether the object has been dropped and is in the recycle bin (YES) or not (NO)

AUDIT_POLICY_ENABLED

AUDIT_POLICY_ENABLE describes all the audit policies that are enable in the database.

Table 9-157 Column information

Column name	Data type	Description
POLICY_NAME	VARCHAR(128)	audit policy name
ENABLED_OPT	VARCHAR(32)	enable option of the audit policy, the possible values are BY, EXCEPT
USER_NAME	VARCHAR(128)	user name for whom the audit policy is enable
WHEN_SUCCESS	VARCHAR(3)	indicates whether the audit policy is enable for auditing successful events or not
WHEN_FAILURE	VARCHAR(3)	indicates whether the audit policy is enable for auditing unsuccessful events or not

AUDIT_TRAIL

AUDIT_TRAIL displays audit records from the audit trail.

Table 9-158 Column information

Column name	Data type	Description
MEMBER_NAME	VARCHAR(128)	cluster member name
SESSION_ID	NUMBER	session identifier
SESSION_SERIAL	NUMBER	session serial number
LOGON_USERNAME	VARCHAR(128)	logon user name of the user whose actions were audited
CURRENT_USERNAME	VARCHAR(128)	effective user for the statement execution
SERVER_PROCESS	NUMBER	server process identifer for the session
CLIENT_PROGRAM_NAME	VARCHAR(128)	client program used for session
CLIENT_USERNAME	VARCHAR(128)	client operating system user name for the session
CLIENT_PROCESS	NUMBER	client process identifer for the session
CLIENT_HOST	VARCHAR(128)	client host ip address for the session
CLIENT_PORT	NUMBER	client port number for the session
CLIENT_TERMINAL	VARCHAR(128)	client terminal name for the session
TRANSACTION_ID	NUMBER	transaction identifier
SCN	VARCHAR(128)	system change number (SCN) string of the query at the time of the event
GCN	NUMBER	global change number (GCN) of the query at the time of the event
DCN	NUMBER	domain change number (DCN) of the query at the time of the event
LCN	NUMBER	local change number (LCN) of the query at the time of the event
STMT_NO	NUMBER	numeric number for each statement run in a session
SQL_TEXT	LONG VARCHAR	SQL associated with the event
SQL_BINDS	LONG VARCHAR	list of bind variables, if any, associated with SQL_TEXT
RETURN_CODE	NUMBER	error code generated by the action, zero if the action succeeded
ERROR_MESSAGE	VARCHAR(1024)	error message generated by the action, null if the action succeeded
ENTRY_ID	NUMBER	audit trail entry identifier in the session
EVENT_TIMESTAMP	TIMESTAMP(6) WITHOUT TIME ZONE	timestamp of the creation of the audit trail entry in local time zone
POLICY_NAME	VARCHAR(128)	audit policy name that caused the current audit record
PRIVILEGE_USED	VARCHAR(32)	database privilege used to execute the action
ACTION_NAME	VARCHAR(32)	action name executed by the user

Column name	Data type	Description
OBJECT_TYPE	VARCHAR(32)	object type of object affected by the action
OBJECT_SCHEMA	VARCHAR(128)	schema name of object affected by the action
OBJECT_NAME	VARCHAR(128)	object name of object affected by the action

DATABASE_PROPERTIES

DATABASE_PROPERTIES lists permanent database properties.

Table 9-159 Column information

Column name	Data type	Description
PROPERTY_NAME	VARCHAR(128)	Property name
PROPERTY_VALUE	VARCHAR(4000)	Property value
DESCRIPTION	VARCHAR(4000)	Property description

DBC_TABLE_TYPE_INFO

Identify the ODBC/JDBC table types available in this database.

Table 9-160 Column information

Column name	Data type	Description
DBC_TABLE_TYPE_ID	NUMBER	number identifier of the table type in ODBC/JDBC
DBC_TABLE_TYPE	VARCHAR(128)	name of the table type in ODBC/JDBC
IS_SUPPORTED	BOOLEAN	is supported feature
COMMENTS	VARCHAR(1024)	comments of the table type

DICTIONARY

DICTIONARY contains descriptions of data dictionary tables and views.

Table 9-161 Column information

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
COMMENTS	VARCHAR(1024)	Text comment on the object

DICT_COLUMNS

DICT_COLUMNS contains descriptions of columns in data dictionary tables and views.

Table 9-162 Column information

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the object that contains the column
TABLE_NAME	VARCHAR(128)	Name of the object that contains the column
COLUMN_NAME	VARCHAR(128)	Name of the column
COMMENTS	VARCHAR(1024)	Text comment on the column

IMPLEMENTATION_INFO

IMPLEMENTATION_INFO contains information about various aspects that are left implementation-defined.

Table 9-163 Column information

Column name	Data type	Description
IMPLEMENTATION_INFO_ID	NUMBER	identifier of the implementation item
IMPLEMENTATION_INFO_NAME	VARCHAR(1024)	descriptive name of the implementation item
INTEGER_VALUE	NUMBER	Value of the implementation item, or null if the value is contained in the column CHARACTER_VALUE
CHARACTER_VALUE	VARCHAR(1024)	Value of the implementation item, or null if the value is contained in the column INTEGER_VALUE
COMMENTS	VARCHAR(1024)	possibly a comment pertaining to the implementation item

IMPLEMENTATION_INFO_BASE

The IMPLEMENTATION_INFO_BASE table has one row for each implementation information item.

Table 9-164 Column information

Column name	Data type	Description
ID	VARCHAR(32)	identifier string of the implementation item
SUB_ID	VARCHAR(32)	identifier string of the implementation item
NAME	VARCHAR(1024)	descriptive name of the implementation item
SUB_NAME	VARCHAR(1024)	descriptive name of the implementation item
IS_SUPPORTED	BOOLEAN	TRUE if the implementation item is supported, FALSE if not
INTEGER_VALUE	NUMBER	Value of the implementation item, or null if the value is contained in the column CHARACTER_VALUE
CHARACTER_VALUE	VARCHAR(1024)	Value of the implementation item, or null if the value is contained in the column INTEGER_VALUE
COMMENTS	VARCHAR(1024)	possibly a comment pertaining to the implementation item

JDBC_CLIENT_PROPS

JDBC_CLIENT_PROPS is the set of jdbc client properties.

Table 9-165 Column information

Column name	Data type	Description
NAME	VARCHAR(128)	property name
MAX_LEN	NATIVE_INTEGER	max length of a value
DEFAULT_VALUE	VARCHAR(128)	default value
DESCRIPTION	VARCHAR(256)	descrption on that property

PRODUCT

PRODUCT is about the product name, version for ODBC, JDBC interface.

Table 9-166 Column information

Column name	Data type	Description
NAME	VARCHAR(32)	the product name
VERSION	VARCHAR(128)	product full version information
PRODUCT_VERSION	NUMBER	product version
MAJOR_VERSION	NUMBER	major version
MINOR_VERSION	NUMBER	minor version
PATCH_VERSION	NUMBER	patch version

SESSION_PRIVS

SESSION_PRIVS describes the privileges that are currently available to the user.

Table 9-167 Column information

Column name	Data type	Description
PRIVILEGE	VARCHAR(256)	Name of the privilege

SUPPLEMENTAL_LOG_TABLE_INFO

SUPPLEMENTAL_LOG_TABLE_INFO describes table-level supplemental logging status.

Table 9-168 Column information

Column name	Data type	Description
TABLE_SCHEMA	VARCHAR(128)	Schema of the object
TABLE_NAME	VARCHAR(128)	Name of the object
SUPPLEMENTAL_LOG_DATA_PK	VARCHAR(32)	Status of table-level PRIMARY KEY COLUMNS supplemental logging: IMPLICIT, EXPLICIT, NO
DROPPED	VARCHAR(3)	Indicates whether the object has been dropped and is in the recycle bin (YES) or not (NO)

Aliased Synonym

It is the public synonym which indicates the view or table in DICTIONARY_SCHEMA.

COLS

COLS is a public synonym for USER_TAB_COLUMNS.

DICT

DICT is a public synonym for DICTIONARY.

IND

IND is a public synonym for USER_INDEXES.

OBJ

OBJ is a public synonym for USER_OBJECTS.

SEQ

SEQ is a public synonym for USER_SEQUENCES.

TABS

TABS is a public synonym for USER_TABLES.

RECYCLEBIN

RECYCLEBIN is a public synonym for USER_RECYCLEBIN.

9.2 INFORMATION_SCHEMA

Views of INFORMATION_SCHEMA schema provides the same information as those defined in the SQL sta
ndard.

Execute *InformationSchema.sql* as follows to use the views.

- For standalone

```
% gsql sys gliese --as sysdba --import
$GOLDILOCKS_HOME/admin/standalone/InformationSchema.sql
```

- For cluster

```
% gsql sys gliese --as sysdba --import  $GOLDILOCKS_HOME/admin/cluster/InformationSchema.sql
```

- Views and tables of INFORMATION_SCHEMA can be retrieved from OPEN phase.
- Objects stored in the recyclebin can not be retrieved in the view of INFORMATION_SCHEM
  A.

COLUMNS

Identify the columns of tables defined in this cataog that are accessible to given user or role.

Table 9-169 Column information

Column name	Data type	Description
TABLE_CATALOG	VARCHAR(128)	catalog name of the column
TABLE_OWNER	VARCHAR(128)	owner name of the column
TABLE_SCHEMA	VARCHAR(128)	schema name of the column
TABLE_NAME	VARCHAR(128)	table name of the column
COLUMN_NAME	VARCHAR(128)	column name
ORDINAL_POSITION	NUMBER	the ordinal position (> 0) of the column in the table
COLUMN_DEFAULT	LONG VARCHAR	the default for the column
IS_NULLABLE	BOOLEAN	is nullable of the column
DATA_TYPE	VARCHAR(128)	the standard name of the data type
CHARACTER_MAXIMUM_LENGTH	NUMBER	the maximum length in characters
CHARACTER_OCTET_LENGTH	NUMBER	the maximum length in octets
NUMERIC_PRECISION	NUMBER	the numeric precision of the numerical Data type
NUMERIC_PRECISION_RADIX	NUMBER	the radix (2 or 10) of the precision of the numerical data type
NUMERIC_SCALE	NUMBER	the numeric scale of the exact numerical data type
DATETIME_PRECISION	NUMBER	for a datetime or interval type, the value is the fractional seconds precision
INTERVAL_TYPE	VARCHAR(32)	for a interval type, the value is in (YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, YEAR TO MONTH, DAY TO HOUR, DAY TO MINUTE, DAY TO SECOND, HOUR TO MINUTE, HOUR TO SECOND, MINUTE TO SECOND)
INTERVAL_PRECISION	NUMBER	for a interval type, the value is the leading precision
CHARACTER_SET_CATALOG	VARCHAR(128)	catalog name of the character set if is is a character string type
CHARACTER_SET_SCHEMA	VARCHAR(128)	schema name of the character set if is is a character string type
CHARACTER_SET_NAME	VARCHAR(128)	character set name of the character set if is is a character string type
COLLATION_CATALOG	VARCHAR(128)	catalog name of the applicable collation if is is a character string type
COLLATION_SCHEMA	VARCHAR(128)	schema name of the applicable collation if is is a character string type
COLLATION_NAME	VARCHAR(128)	collation name of the applicable collation if is is a character

Column name	Data type	Description
		string type
DOMAIN_CATALOG	VARCHAR(128)	catalog name of the domain used by the column being described
DOMAIN_SCHEMA	VARCHAR(128)	schema name of the domain used by the column being described
DOMAIN_NAME	VARCHAR(128)	domain name of the domain used by the column being described
UDT_CATALOG	VARCHAR(128)	catalog name of the user-defined type of the data type being described
UDT_SCHEMA	VARCHAR(128)	schema name of the user-defined type of the data type being described
UDT_NAME	VARCHAR(128)	user-defined type name of the user-defined type of the data type being described
SCOPE_CATALOG	VARCHAR(128)	catalog name of the referenceable table if DATA_TYPE is REF
SCOPE_SCHEMA	VARCHAR(128)	schema name of the referenceable table if DATA_TYPE is REF
SCOPE_NAME	VARCHAR(128)	scope name of the referenceable table if DATA_TYPE is REF
MAXIMUM_CARDINALITY	NUMBER	maximum cardinality if DATA_TYPE is ARRAY
DTD_IDENTIFIER	NUMBER	data type descriptor identifier
IS_SELF_REFERENCING	BOOLEAN	is a self-referencing column
IS_IDENTITY	BOOLEAN	is an identity column
IDENTITY_GENERATION	VARCHAR(32)	for an identity column, the value is in (ALWAYS, BY DEFAULT)
IDENTITY_START	NUMBER	for an identity column, the start value of the identity column
IDENTITY_INCREMENT	NUMBER	for an identity column, the increment of the identity column
IDENTITY_MAXIMUM	NUMBER	for an identity column, the maximum value of the identity column
IDENTITY_MINIMUM	NUMBER	for an identity column, the minimum value of the identity column
IDENTITY_CYCLE	BOOLEAN	for an identity column, the cycle option
IS_GENERATED	BOOLEAN	is a generated column
GENERATION_EXPRESSION	VARCHAR(128)	for a generated column, the text of the generation expression
IS_SYSTEM_VERSION_START	BOOLEAN	is a system-version start column
IS_SYSTEM_VERSION_END	BOOLEAN	is a system-version end column
SYSTEM_VERSION_TIMESTAMP_GENERATION	VARCHAR(32)	for a system-version column, the value is ALWAYS
IS_UPDATABLE	BOOLEAN	is an updatable column
DECLARED_DATA_TYPE	VARCHAR(128)	the data type name that a user declared

Column name	Data type	Description
DECLARED_NUMERIC_PRECISION	NUMBER	the precision value that a user declared
DECLARED_NUMERIC_SCALE	NUMBER	the scale value that a user declared
COMMENTS	VARCHAR(1024)	comments of the column

COLUMN_PRIVILEGES

Identify the privileges on columns of tables defined in this catalog that are available to or granted by a giv en user or role.

Table 9-170 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	authorization name of the user who granted column privileges
GRANTEE	VARCHAR(128)	authorization name of some user or role, or PUBLIC to indicate all us ers, to whom the column privilege being described is granted
TABLE_CATALOG	VARCHAR(128)	catalog name of the column on which the privilege being described was granted
TABLE_OWNER	VARCHAR(128)	table owner name of the column on which the privilege being descri bed was granted
TABLE_SCHEMA	VARCHAR(128)	schema name of the column on which the privilege being described was granted
TABLE_NAME	VARCHAR(128)	table name of the column on which the privilege being described w as granted
COLUMN_NAME	VARCHAR(128)	column name of the column on which the privilege being described was granted
PRIVILEGE_TYPE	VARCHAR(32)	the value is in (SELECT, INSERT, UPDATE, REFERENCES)
IS_GRANTABLE	BOOLEAN	is grantable

CONSTRAINT_COLUMN_USAGE

Identify the columns used by referential constraints, unique constraints, check constraints, and assertions defined in this catalog and owned by a given user or role.

Table 9-171 Column information

Column name	Data type	Description
TABLE_CATALOG	VARCHAR(128)	catalog name of the column that participates in the constraint being described
TABLE_OWNER	VARCHAR(128)	owner name of the column that participates in the constraint being described
TABLE_SCHEMA	VARCHAR(128)	schema name of the column that participates in the constraint bein g described
TABLE_NAME	VARCHAR(128)	table name of the column that participates in the constraint being d escribed
COLUMN_NAME	VARCHAR(128)	column name that participates in the constraint being described
CONSTRAINT_CATAL OG	VARCHAR(128)	catalog name of the constraint
CONSTRAINT_OWNE R	VARCHAR(128)	owner name of the constraint
CONSTRAINT_SCHE MA	VARCHAR(128)	schema name of the constraint
CONSTRAINT_NAME	VARCHAR(128)	constraint name

CONSTRAINT_TABLE_USAGE

Identify the tables that are used by referential constraints, unique constraints, check constraints, and asse rtions defined in this catalog and owned by a given user or role.

Table 9-172 Column information

Column name	Data type	Description
TABLE_CATALOG	VARCHAR(128)	catalog name of the table that participates in the constraint being d escribed
TABLE_OWNER	VARCHAR(128)	owner name of the table that participates in the constraint being de scribed
TABLE_SCHEMA	VARCHAR(128)	schema name of the table that participates in the constraint being d escribed
TABLE_NAME	VARCHAR(128)	table name that participates in the constraint being described
CONSTRAINT_CATAL OG	VARCHAR(128)	catalog name of the constraint
CONSTRAINT_OWNE R	VARCHAR(128)	owner name of the constraint
CONSTRAINT_SCHE MA	VARCHAR(128)	schema name of the constraint
CONSTRAINT_NAME	VARCHAR(128)	constraint name

INFORMATION_SCHEMA_CATALOG_NAME

Identify the catalog that contains the Information Schema

Table 9-173 Column information

Column name	Data type	Description
CATALOG_NAME	VARCHAR(128)	the name of catalog in which this Information Schema resides

KEY_COLUMN_USAGE

Identify the columns defined in this catalog that are constrained as keys and that are accessible by a given user or role.

Table 9-174 Column information

Column name	Data type	Description
CONSTRAINT_CATALOG	VARCHAR(128)	catalog name of the constraint
CONSTRAINT_OWNER	VARCHAR(128)	owner name of the constraint
CONSTRAINT_SCHEMA	VARCHAR(128)	schema name of the constraint
CONSTRAINT_NAME	VARCHAR(128)	constraint name
TABLE_CATALOG	VARCHAR(128)	catalog name of the column that participates in the constraint being described
TABLE_OWNER	VARCHAR(128)	owner name of the column that participates in the constraint being described
TABLE_SCHEMA	VARCHAR(128)	schema name of the column that participates in the constraint being described
TABLE_NAME	VARCHAR(128)	table name of the column that participates in the constraint b eing described
COLUMN_NAME	VARCHAR(128)	column name that participates in the constraint being describe d
ORDINAL_POSITION	NUMBER	the ordinal position of the specific column in the constraint be ing described. If the constraint described is a key of cardinality 1 (one), then the value of ORDINAL_POSITION is always 1 (on e).
POSITION_IN_UNIQUE_CO NSTRAINT	NUMBER	If the constraint being described is a foreign key constraint, th en the value of POSITION_IN_UNIQUE_CONSTRAINT is the ord inal position of the referenced column corresponding to the re ferencing column being described, in the corresponding uniqu e key constraint.

MODULES

Identify the SQL-server modules in this catalog that are accessible to a given user or role.

Table 9-175 Column information

Column name	Data type	Description
MODULE_CATALOG	VARCHAR(128)	catalog name of the SQL-server module
MODULE_OWNER	VARCHAR(128)	owner name of the SQL-server module
MODULE_SCHEMA	VARCHAR(128)	schema name of the SQL-server module
MODULE_NAME	VARCHAR(128)	name of the SQL-server module
DEFAULT_CHARACTER_SET_CATALOG	VARCHAR(128)	default character set catalog name of the SQL-server module
DEFAULT_CHARACTER_SET_SCHEMA	VARCHAR(128)	default character set schema name of the SQL-server module
DEFAULT_CHARACTER_SET	VARCHAR(128)	default character set name of the SQL-server module
DEFAULT_SCHEMA_CATALOG	VARCHAR(128)	catalog name of default schema of SQL-server module
DEFAULT_SCHEMA_NAME	VARCHAR(128)	default scheam name of the SQL-server module
MODULE_DEFINITION	LONG VARCHAR	definition of the SQL-server module
MODULE_AUTHORIZATION	VARCHAR(32)	authorization of the SQL-server module(DEFINER/INVOKER)
SQL_PATH	VARCHAR(1024)	described SQL PATH when the SQL-server module is defined
CREATED	TIMESTAMP(6) WITHOUT TIME ZONE	creation time of the SQL-server module
LAST_ALTERED	TIMESTAMP(6) WITHOUT TIME ZONE	most lately altered time of the SQL-server module

MODULE_BODY

Identify the SQL-server module bodies in this catalog that are accessible to a given user or role.

Table 9-176 Column information

Column name	Data type	Description
MODULE_CATALOG	VARCHAR(128)	catalog name of the SQL-server module
MODULE_OWNER	VARCHAR(128)	owner name of the SQL-server module'
MODULE_SCHEMA	VARCHAR(128)	schema name of the SQL-server module
MODULE_NAME	VARCHAR(128)	name of the SQL-server module
MODULE_DEFINITION	LONG VARCHAR	definition of the SQL-server module body
CREATED	TIMESTAMP(6) WITH OUT TIME ZONE	creation time of the SQL-server module body
LAST_ALTERED	TIMESTAMP(6) WITH OUT TIME ZONE	most lately altered time of the SQL-server module body

MODULE_BODY_MODULE_USAGE

Identify the SQL-server modules owned by a given user or role on which SQL-server module bodies define d in this catalog are dependent.

Table 9-177 Column information

Column name	Data type	Description
MODULE_CATALOG	VARCHAR(128)	catalog name of the SQL-server module
MODULE_OWNER	VARCHAR(128)	owner name of the SQL-server module
MODULE_SCHEMA	VARCHAR(128)	schema name of the SQL-server module
MODULE_NAME	VARCHAR(128)	name of the SQL-server module
REF_MODULE_CATALOG	VARCHAR(128)	catalog name of the SQL-server module of contained in definit ion text of the SQL-server module body
REF_MODULE_OWNER	VARCHAR(128)	owner name of the SQL-server module of contained in definiti on text of the SQL-server module body
REF_MODULE_SCHEMA	VARCHAR(128)	schema name of the SQL-server module of contained in defini tion text of the SQL-server module body
REF_MODULE_NAME	VARCHAR(128)	SQL-server module name of contained in definition text of the SQL-server module body

MODULE_BODY_ROUTINE_USAGE

Identify the SQL-invoked routines owned by a given user or role on which SQL-server module bodies defi ned in this catalog are dependent.

Table 9-178 Column information

Column name	Data type	Description
MODULE_CATALOG	VARCHAR(128)	catalog name of the SQL-server module
MODULE_OWNER	VARCHAR(128)	owner name of the SQL-server module
MODULE_SCHEMA	VARCHAR(128)	schema name of the SQL-server module
MODULE_NAME	VARCHAR(128)	name of the SQL-server module
ROUTINE_CATALOG	VARCHAR(128)	catalog name of the SQL-invoked routine of contained in definition t ext of the SQL-server module body
ROUTINE_OWNER	VARCHAR(128)	owner name of the SQL-invoked routine of contained in definition t ext of the SQL-server module body
ROUTINE_SCHEMA	VARCHAR(128)	schema name of the SQL-invoked routine of contained in definition text of the SQL-server module body
ROUTINE_NAME	VARCHAR(128)	SQL-invoked routine name of contained in definition text of the SQL -server module body

MODULE_BODY_SEQUENCE_USAGE

Identify the sequences owned by a given user or role on which SQL-server module bodies defined in this c atalog are dependent.

Table 9-179 Column information

Column name	Data type	Description
MODULE_CATALOG	VARCHAR(128)	catalog name of the SQL-server module
MODULE_OWNER	VARCHAR(128)	owner name of the SQL-server module
MODULE_SCHEMA	VARCHAR(128)	schema name of the SQL-server module
MODULE_NAME	VARCHAR(128)	name of the SQL-server module
SEQUENCE_CATALOG	VARCHAR(128)	catalog name of the sequence of contained in definition text of t he SQL-server module body
SEQUENCE_OWNER	VARCHAR(128)	owner name of the sequence of contained in definition text of t he SQL-server module body
SEQUENCE_SCHEMA	VARCHAR(128)	schema name of the sequence of contained in definition text of the SQL-server module body
SEQUENCE_NAME	VARCHAR(128)	sequence name of contained in definition text of the SQL-server module body

MODULE_BODY_TABLE_USAGE

Identify the tables owned by a given user or role on which SQL-server module bodies defined in this catal og are dependent.

Table 9-180 Column information

Column name	Data type	Description
MODULE_CATALOG	VARCHAR(128)	catalog name of the SQL-server module
MODULE_OWNER	VARCHAR(128)	owner name of the SQL-server module
MODULE_SCHEMA	VARCHAR(128)	schema name of the SQL-server module
MODULE_NAME	VARCHAR(128)	name of the SQL-server module
TABLE_CATALOG	VARCHAR(128)	catalog name of the table of contained in definition text of the SQL-server module body
TABLE_OWNER	VARCHAR(128)	owner name of the table of contained in definition text of the SQL-s erver module body
TABLE_SCHEMA	VARCHAR(128)	schema name of the table of contained in definition text of the SQL-server module body
TABLE_NAME	VARCHAR(128)	table name of contained in definition text of the SQL-server module body

MODULE_MODULE_USAGE

Identify the SQL-server modules owned by a given user or role on which SQL-server modules defined in th is catalog are dependent.

Table 9-181 Column information

Column name	Data type	Description
MODULE_CATALOG	VARCHAR(128)	catalog name of the SQL-server module
MODULE_OWNER	VARCHAR(128)	owner name of the SQL-server module
MODULE_SCHEMA	VARCHAR(128)	schema name of the SQL-server module
MODULE_NAME	VARCHAR(128)	name of the SQL-server module
REF_MODULE_CATALOG	VARCHAR(128)	catalog name of the SQL-server module of contained in definit ion text of the SQL-server module
REF_MODULE_OWNER	VARCHAR(128)	owner name of the SQL-server module of contained in definiti on text of the SQL-server module
REF_MODULE_SCHEMA	VARCHAR(128)	schema name of the SQL-server module of contained in defini tion text of the SQL-server module
REF_MODULE_NAME	VARCHAR(128)	SQL-server module name of contained in definition text of the SQL-server module

MODULE_PRIVILEGES

Identify the privileges on SQL-server modules defined in this catalog that are available to or granted by a given user or role.

Table 9-182 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	authorization name of the user who granted SQL-server module privileges
GRANTEE	VARCHAR(128)	authorization name of some user or role, or PUBLIC to indicate all users, to whom the SQL-server module privilege being described is granted
MODULE_CATALOG	VARCHAR(128)	catalog name of the SQL-server module on which the privilege being described was granted
MODULE_OWNER	VARCHAR(128)	owner name of the the SQL-server module on which the privilege being described was granted
MODULE_SCHEMA	VARCHAR(128)	schema name of the the SQL-server module on which the privilege being described was granted
MODULE_NAME	VARCHAR(128)	name of the the SQL-server module on which the privilege being described was granted
PRIVILEGE_TYPE	VARCHAR(32)	the value is in (EXECUTE)
IS_GRANTABLE	BOOLEAN	is grantable

MODULE_ROUTINE_USAGE

Identify the SQL-invoked routines owned by a given user or role on which SQL-server modules defined in this catalog are dependent.

Table 9-183 Column information

Column name	Data type	Description
MODULE_CATALOG	VARCHAR(128)	catalog name of the SQL-server module
MODULE_OWNER	VARCHAR(128)	owner name of the SQL-server module
MODULE_SCHEMA	VARCHAR(128)	schema name of the SQL-server module
MODULE_NAME	VARCHAR(128)	name of the SQL-server module
ROUTINE_CATALOG	VARCHAR(128)	catalog name of the SQL-invoked routine of contained in definition text of the SQL-server module
ROUTINE_OWNER	VARCHAR(128)	owner name of the SQL-invoked routine of contained in definition text of the SQL-server module
ROUTINE_SCHEMA	VARCHAR(128)	schema name of the SQL-invoked routine of contained in definition text of the SQL-server module
ROUTINE_NAME	VARCHAR(128)	SQL-invoked routine name of contained in definition text of the SQL-server module

MODULE_SEQUENCE_USAGE

Identify the sequences owned by a given user or role on which SQL-server modules defined in this catalog are dependent.

Table 9-184 Column information

Column name	Data type	Description
MODULE_CATALOG	VARCHAR(128)	catalog name of the SQL-server module
MODULE_OWNER	VARCHAR(128)	owner name of the SQL-server module
MODULE_SCHEMA	VARCHAR(128)	schema name of the SQL-server module
MODULE_NAME	VARCHAR(128)	name of the SQL-server module
SEQUENCE_CATALOG	VARCHAR(128)	catalog name of the sequence of contained in definition text of t he SQL-server module
SEQUENCE_OWNER	VARCHAR(128)	owner name of the sequence of contained in definition text of t he SQL-server module
SEQUENCE_SCHEMA	VARCHAR(128)	schema name of the sequence of contained in definition text of the SQL-server module
SEQUENCE_NAME	VARCHAR(128)	sequence name of contained in definition text of the SQL-server module

MODULE_TABLE_USAGE

Identify the tables owned by a given user or role on which SQL-server modules defined in this catalog are dependent.

Table 9-185 Column information

Column name	Data type	Description
MODULE_CATALOG	VARCHAR(128)	catalog name of the SQL-server module
MODULE_OWNER	VARCHAR(128)	owner name of the SQL-server module
MODULE_SCHEMA	VARCHAR(128)	schema name of the SQL-server module
MODULE_NAME	VARCHAR(128)	name of the SQL-server module
TABLE_CATALOG	VARCHAR(128)	catalog name of the table of contained in definition text of the SQL-server module
TABLE_OWNER	VARCHAR(128)	owner name of the table of contained in definition text of the SQL-server module
TABLE_SCHEMA	VARCHAR(128)	schema name of the table of contained in definition text of the SQL-server module
TABLE_NAME	VARCHAR(128)	table name of contained in definition text of the SQL-server module

PARAMETERS

Identify the SQL parameters of SQL-invoked routines defined in this catalog that are accessible to a given user or role.

Table 9-186 Column information

Column name	Data type	Description
SPECIFIC_CATALOG	VARCHAR(128)	catalog name of the specific name of the SQL- invoke d routine that contains the SQL parameter being descr ibed
SPECIFIC_OWNER	VARCHAR(128)	owner name of the specific name of the SQL- invoked routine that contains the SQL parameter being describ ed
SPECIFIC_SCHEMA	VARCHAR(128)	schema name of the specific name of the SQL- invoke d routine that contains the SQL parameter being descr ibed
SPECIFIC_NAME	VARCHAR(128)	specific name of the SQL- invoked routine that contai ns the SQL parameter being described
ORDINAL_POSITION	NUMBER	ordinal position of the SQL- invoked routine that cont ains the SQL parameter being described
PARAMETER_MODE	VARCHAR(32)	parameter mode of the SQL parameter being describe d
IS_RESULT	BOOLEAN	the parameter is RESULT parameter of type-preserving function
AS_LOCATOR	BOOLEAN	the parameter is passed as locator
PARAMETER_NAME	VARCHAR(128)	name of the SQL parameter being descaibed
FROM_SQL_SPECIFIC_CATALOG	VARCHAR(128)	specific catalog name of the from-sql routine for the i nput parameter being described
FROM_SQL_SPECIFIC_SCHEMA	VARCHAR(128)	specific schema name of the from-sql routine for the i nput parameter being described
FROM_SQL_SPECIFIC_NAME	VARCHAR(128)	specific name of the from-sql routine for the input par ameter being described
TO_SQL_SPECIFIC_CATALOG	VARCHAR(128)	specific catalog name of the to-sql routine for the inp ut parameter being described
TO_SQL_SPECIFIC_SCHEMA	VARCHAR(128)	specific schema name of the to-sql routine for the inp ut parameter being described
TO_SQL_SPECIFIC_NAME	VARCHAR(128)	specific name of the to-sql routine for the input para meter being described
DATA_TYPE	VARCHAR(128)	data type of the SQL parameter being described
CHARACTER_MAXIMUM_LENGTH	NUMBER	maximum length of the SQL parameter being describe d
		maximum length in octets of the SQL parameter bein

Column name	Data type	Description
CHARACTER_OCTET_LENGTH	NUMBER	g described
CHARACTER_SET_CATALOG	VARCHAR(128)	character set catalog name of the data type of the SQL parameter being described
CHARACTER_SET_SCHEMA	VARCHAR(128)	character set schema name of the data type of the SQL parameter being described
CHARACTER_SET_NAME	VARCHAR(128)	character set name of the data type of the SQL parameter being described
COLLATION_CATALOG	VARCHAR(128)	collation catalog name of the data type of the SQL parameter being described
COLLATION_SCHEMA	VARCHAR(128)	collation schema name of the data type of the SQL parameter being described
COLLATION_NAME	VARCHAR(128)	collation name of the data type of the SQL parameter being described
NUMERIC_PRECISION	NUMBER	precision of the data type of the SQL parameter being described
NUMERIC_PRECISION_RADIX	NUMBER	precision radix of the data type of the SQL parameter being described
NUMERIC_SCALE	NUMBER	scale of the data type of the SQL parameter being described
DATETIME_PRECISION	NUMBER	fractional second precisions of the data type of the SQL parameter being described
INTERVAL_TYPE	VARCHAR(32)	interval qualifier of the data type of the SQL parameter being described
INTERVAL_PRECISION	NUMBER	interval precision of the data type of the SQL parameter being described
UDT_CATALOG	VARCHAR(128)	catalog name of UDT of the data type of the SQL parameter being described
UDT_SCHEMA	VARCHAR(128)	schema name of UDT of the data type of the SQL parameter being described
UDT_NAME	VARCHAR(128)	name of UDT of the data type of the SQL parameter being described
SCOPE_CATALOG	VARCHAR(128)	catalog name of referenceable tables of the data type of the SQL parameter being described
SCOPE_SCHEMA	VARCHAR(128)	schema name of referenceable tables of the data type of the SQL parameter being described
SCOPE_NAME	VARCHAR(128)	name of referenceable tables of the data type of the SQL parameter being described
MAXIMUM_CARDINALITY	NUMBER	maximum cardinality of the data type of the SQL parameter being described
DTD_IDENTIFIER	NUMBER	dtd identifier of the data type of the SQL parameter being described
		declared data type of the SQL parameter being descri

Column name	Data type	Description
DECLARED_DATA_TYPE	VARCHAR(128)	bed
DECLARED_NUMERIC_PRECISION	NUMBER	precision of declared data type of the SQL parameter being described
DECLARED_NUMERIC_SCALE	NUMBER	scale of declared data type of the SQL parameter bein g described
PARAMETER_DEFAULT	LONG VARCHAR	default value of the SQL parameter being described

REFERENTIAL_CONSTRAINTS

Identify the referential constraints defined on tables in this catalog that are accssible to a given user or rol e.

Table 9-187 Column information

Column name	Data type	Description
CONSTRAINT_CATALOG	VARCHAR(128)	catalog name of the referential constraint
CONSTRAINT_OWNER	VARCHAR(128)	owner name who owns the referential constraint
CONSTRAINT_SCHEMA	VARCHAR(128)	schema name of the referential constraint being described
CONSTRAINT_NAME	VARCHAR(128)	referential constraint name
CONSTRAINT_TABLE_NAME	VARCHAR(128)	name of the table to which the referential constraint being described applies
CONSTRAINT_COLUMN_NAME	VARCHAR(128)	column name of the table to which the referential constraint being described applies
ORDINAL_POSITION	NUMBER	the ordinal position of the specific column in the referentail constraint being described.
UNIQUE_CONSTRAINT_CATALOG	VARCHAR(128)	catalog name of the unique or primary key constraint applied to the referenced column list being described
UNIQUE_CONSTRAINT_OWNER	VARCHAR(128)	owner name of the unique or primary key constraint applied to the referenced column list being described
UNIQUE_CONSTRAINT_SCHEMA	VARCHAR(128)	schema name of the unique or primary key constraint applied to the referenced column list being described
UNIQUE_CONSTRAINT_NAME	VARCHAR(128)	constraint name of the unique or primary key constraint applied to the referenced column list being described
UNIQUE_CONSTRAINT_TABLE_NAME	VARCHAR(128)	table name of the unique or primary key constraint applied to the referenced column list being described
UNIQUE_CONSTRAINT_COLUMN_NAME	VARCHAR(128)	column name of the unique or primary key constraint applied to the referenced column list being described
IS_PRIMARY_KEY	BOOLEAN	whether the constraint applied to the referenced column list being described, is primary key or not
MATCH_OPTION	VARCHAR(32)	the referential constraint that has a match option: the value in (SIMPLE, PARTIAL, FULL)
UPDATE_RULE	VARCHAR(32)	the referential constraint that has an update rule: the value in (NO ACTION, RESTRICT, CASCADE, SET NULL, SET DEFAULT)
		the referential constraint that has a delete rule: the value in (NO ACTION, R

Column name	Data type	Description
DELETE_RULE	VARCHAR(32)	ESTRICT, CASCADE, SET NULL, SET DEFAULT)
IS_DEFERRABLE	BOOLEAN	is a deferrable constraint
INITIALLY_DEFERRED	BOOLEAN	is an initially deferred constraint

ROUTINES

Identify the SQL-invoked routines in this catalog that are accessible to a given user or role.

Table 9-188 Column information

Column name	Data type	Description
SPECIFIC_CATALOG	VARCHAR(128)	specific catalog name of the routine
SPECIFIC_OWNER	VARCHAR(128)	specific owner name of the routine
SPECIFIC_SCHEMA	VARCHAR(128)	specific schema name of the routine
SPECIFIC_NAME	VARCHAR(128)	specific name of the routine
ROUTINE_CATALOG	VARCHAR(128)	catalog name of the routine
ROUTINE_OWNER	VARCHAR(128)	owner name of the routine
ROUTINE_SCHEMA	VARCHAR(128)	schema name of the routine
ROUTINE_NAME	VARCHAR(128)	null
ROUTINE_TYPE	VARCHAR(128)	name of the routine
MODULE_CATALOG	VARCHAR(128)	module name of the routine
MODULE_SCHEMA	VARCHAR(128)	schema name of the module in which the routine is defined
MODULE_NAME	VARCHAR(128)	name of the module in which the routine is defined
UDT_CATALOG	VARCHAR(128)	catalog name of the user-defined data type which defined the routine as a method function
UDT_SCHEMA	VARCHAR(128)	schema name of the user-defined data type which defined the routine as a method function
UDT_NAME	VARCHAR(128)	name of the user-defined data type which defined the routine as a method function
DATA_TYPE	VARCHAR(128)	data type the routine returns
CHARACTER_MAXIMUM_LENGTH	NUMBER	maximum character length of data type the routine returns
CHARACTER_OCTET_LENGTH	NUMBER	maximum character length in octets of data type the routine returns
CHARACTER_SET_CATALOG	VARCHAR(128)	character set catalog name of data type the routine returns
CHARACTER_SET_SCHEMA	VARCHAR(128)	character set schema name of data type the routine returns
CHARACTER_SET_NAME	VARCHAR(128)	character set name of data type the routine returns
COLLATION_CATALOG	VARCHAR(128)	collation catalog name of data type the routine returns
COLLATION_SCHEMA	VARCHAR(128)	collation schema name of data type the routine returns
COLLATION_NAME	VARCHAR(128)	collation name of data type the routine returns
NUMERIC_PRECISION	NUMBER	precision of data type the routine returns
NUMERIC_PRECISION_RADIX	NUMBER	precision radix of data type the routine returns
NUMERIC_SCALE	NUMBER	scale of data type the routine returns

Column name	Data type	Description
DATETIME_PRECISION	NUMBER	fractional seconds precision of data type the routine returns
INTERVAL_TYPE	VARCHAR(32)	interval qualifier for data type the routine returns
INTERVAL_PRECISION	NUMBER	interval leading field precision of data type the routine returns
TYPE_UDT_CATALOG	VARCHAR(128)	catalog name of the user-defined data type, which is the data type the routine returns
TYPE_UDT_SCHEMA	VARCHAR(128)	schema name of the user-defined data type, which is the data type the routine returns
TYPE_UDT_NAME	VARCHAR(128)	name of the user-defined data type, which is the data type the routine returns
SCOPE_CATALOG	VARCHAR(128)	catalog name of referenceable table
SCOPE_SCHEMA	VARCHAR(128)	schema name of referenceable table
SCOPE_NAME	VARCHAR(128)	name of referenceable table
MAXIMUM_CARDINALITY	NUMBER	maximum cardinality of data type the routine returns
DTD_IDENTIFIER	NUMBER	dtd ientifier of data type the routine returns
ROUTINE_BODY	VARCHAR(32)	type of the routine body
ROUTINE_DEFINITION	LONG VARCHAR	catalog name of the routine
EXTERNAL_NAME	VARCHAR(128)	external name of the external routine
EXTERNAL_LANGUAGE	VARCHAR(32)	language of the external routine
PARAMETER_STYLE	VARCHAR(32)	SQL parameter passing style of the external routine
IS_DETERMINISTIC	BOOLEAN	the routine is deterministic or not
SQL_DATA_ACCESS	VARCHAR(32)	routine possibly contains SQL or access data
IS_NULL_CALL	BOOLEAN	routine returns NULL if any of parameter values are NULL
SQL_PATH	VARCHAR(1024)	described SQL PATH when the routine is defined
SCHEMA_LEVEL_ROUTINE	BOOLEAN	the routine is schema-level routine
MAX_DYNAMIC_RESULT_SETS	NUMBER	max result set count of the routine
IS_USER_DEFINED_CAST	BOOLEAN	the routine is a function that is a user-defined cast function
IS_IMPLICITLY_INVOCABLE	BOOLEAN	the user-defined cast function is implicitly invocable
SECURITY_TYPE	VARCHAR(32)	security type of the routine(DEFINER/INVOKER)
TO_SQL_SPECIFIC_CATALOG	VARCHAR(128)	catalog name of the to-sql routine of the result type of routine
TO_SQL_SPECIFIC_SCHEMA	VARCHAR(128)	schema name of the to-sql routine of the result type of routine
TO_SQL_SPECIFIC_NAME	VARCHAR(128)	name of the to-sql routine of the result type of routine
AS_LOCATOR	BOOLEAN	return value of the routine is passed as locator
CREATED	TIMESTAMP(6) WITHOUT TIME ZONE	creation time of the routine
LAST_ALTERED	TIMESTAMP(6) WITHOUT TIME ZONE	most lately altered time of the routine
NEW_SAVEPOINT_LEVEL	BOOLEAN	specifiy new savepoint level or not

Column name	Data type	Description
IS_UDT_DEPENDENT	BOOLEAN	routine is dependent
RESULT_CAST_FROM_DATA_TYPE	VARCHAR(128)	data type which is specificed in result cast clause of the routine definition
RESULT_CAST_AS_LOCATOR	BOOLEAN	locator indication which is specificed in result cast clause of the routine definition
RESULT_CAST_CHAR_MAX_LENGTH	NUMBER	maximum character length of data type which is specificed in result cast clause of the routine definition
RESULT_CAST_CHAR_OCTET_LENGTH	NUMBER	maximum character length in octets of data type which is specificed in result cast clause of the routine definition
RESULT_CAST_CHAR_SET_CATALOG	VARCHAR(128)	character set catalog name of data type which is specificed in result cast clause of the routine definition
RESULT_CAST_CHAR_SET_SCHEMA	VARCHAR(128)	character set schema name of data type which is specificed in result cast clause of the routine definition
RESULT_CAST_CHARACTER_SET_NAME	VARCHAR(128)	character set name of data type which is specificed in result cast clause of the routine definition
RESULT_CAST_COLLATION_CATALOG	VARCHAR(128)	collation catalog name of data type which is specificed in result cast clause of the routine definition
RESULT_CAST_COLLATION_SCHEMA	VARCHAR(128)	collation schema name of data type which is specificed in result cast clause of the routine definition
RESULT_CAST_COLLATION_NAME	VARCHAR(128)	collation name of data type which is specificed in result cast clause of the routine definition
RESULT_CAST_NUMERIC_PRECISION	NUMBER	precision of data type which is specificed in result cast clause of the routine definition
RESULT_CAST_NUMERIC_RADIX	NUMBER	precision radix of data type which is specificed in result cast clause of the routine definition
RESULT_CAST_NUMERIC_SCALE	NUMBER	scale of data type which is specificed in result cast clause of the routine definition
RESULT_CAST_DATETIME_PRECISION	NUMBER	fractional seconds precision of data type which is specificed in result cast clause of the routine definition
RESULT_CAST_INTERVAL_TYPE	VARCHAR(32)	interval qualifier of data type which is specificed in result cast clause of the routine definition
RESULT_CAST_INTERVAL_PRECISION	NUMBER	interval precision of data type which is specificed in result cast clause of the routine definition
RESULT_CAST_TYPE_UDT_CATALOG	VARCHAR(128)	UDT catalog name of data type which is specificed in result cast clause of the routine definition
RESULT_CAST_TYPE_UDT_SCHEMA	VARCHAR(128)	UDT schema name of data type which is specificed in result cast clause of the routine definition
RESULT_CAST_TYPE_UDT_NAME	VARCHAR(128)	UDT name of data type which is specificed in result cast clause of the routine definition
RESULT_CAST_SCOPE_CATALOG	VARCHAR(128)	catalog name of referenceable table described in result cast clause of the routine definition
RESULT_CAST_SCOPE_SCH		schema name of referenceable table described in result cast cl

Column name	Data type	Description
EMA	VARCHAR(128)	ause of the routine definition
RESULT_CAST_SCOPE_NAME	VARCHAR(128)	name of referenceable table described in result cast clause of the routine definition
RESULT_CAST_MAX_CARDINALITY	NUMBER	maximum cardinality of data type which is specificed in result cast clause of the routine definition
RESULT_CAST_DTD_IDENTIFIER	NUMBER	dtd identifier of data type which is specificed in result cast clause of the routine definition
DECLARED_DATA_TYPE	VARCHAR(128)	declared data type of the routine returns
DECLARED_NUMERIC_PRECISION	NUMBER	declared data type precision of the routine returns
DECLARED_NUMERIC_SCALE	NUMBER	declared data type scale of the routine returns
RESULT_CAST_FROM_DECLARED_DATA_TYPE	VARCHAR(128)	declared data type which is specificed in result cast clause of the routine definition
RESULT_CAST_DECLARED_NUMERIC_PRECISION	NUMBER	declared data type precision which is specificed in result cast clause of the routine definition
RESULT_CAST_DECLARED_NUMERIC_SCALE	NUMBER	declared data type scale which is specificed in result cast clause of the routine definition

ROUTINE_MODULE_USAGE

Identify the SQL-server modules owned by a given user or role on which SQL routines defined in this catal og are dependent.

Table 9-189 Column information

Column name	Data type	Description
SPECIFIC_CATALOG	VARCHAR(128)	specific catalog name of the routine
SPECIFIC_OWNER	VARCHAR(128)	specific owner name of the routine
SPECIFIC_SCHEMA	VARCHAR(128)	specific schema name of the routine
SPECIFIC_NAME	VARCHAR(128)	specific name of the routine
MODULE_CATALOG	VARCHAR(128)	catalog name of the SQL-server module of contained in routine bod y of the SQL-invoked routine
MODULE_OWNER	VARCHAR(128)	owner name of the SQL-server module of contained in routine body of the SQL-invoked routine
MODULE_SCHEMA	VARCHAR(128)	schema name of the SQL-server module of contained in routine bod y of the SQL-invoked routine
MODULE_NAME	VARCHAR(128)	SQL-server module name of contained in routine body of the SQL-in voked routine

ROUTINE_PRIVILEGES

Identify the privileges on SQL-invoked routines defined in this catalog that are available to or granted by a given user or role.

Table 9-190 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	authorization name of the user who granted routine privileges
GRANTEE	VARCHAR(128)	authorization name of some user or role, or PUBLIC to indicate all users, to whom the routine privilege being described is granted
SPECIFIC_CATALOG	VARCHAR(128)	specific catalog name of the SQL-invoked routine on which the privilege being described was granted
SPECIFIC_OWNER	VARCHAR(128)	specific owner name of the the SQL-invoked routine on which the privilege being described was granted
SPECIFIC_SCHEMA	VARCHAR(128)	specific schema name of the the SQL-invoked routine on which the privilege being described was granted
SPECIFIC_NAME	VARCHAR(128)	specific name of the the SQL-invoked routine on which the privilege being described was granted
ROUTINE_CATALOG	VARCHAR(128)	routine catalog name of the SQL-invoked routine on which the privilege being described was granted
ROUTINE_OWNER	VARCHAR(128)	null
ROUTINE_SCHEMA	VARCHAR(128)	routine schema name of the the SQL-invoked routine on which the privilege being described was granted
ROUTINE_NAME	VARCHAR(128)	routine name of the the SQL-invoked routine on which the privilege being described was granted
PRIVILEGE_TYPE	VARCHAR(32)	the value is in (EXECUTE)
IS_GRANTABLE	BOOLEAN	is grantable

ROUTINE_ROUTINE_USAGE

Identify each SQL-invoked routine owned by a given user or role on which an SQL routine defined in this catalog is dependent.

Table 9-191 Column information

Column name	Data type	Description
SPECIFIC_CATALOG	VARCHAR(128)	specific catalog name of the routine
SPECIFIC_OWNER	VARCHAR(128)	specific owner name of the routine
SPECIFIC_SCHEMA	VARCHAR(128)	specific schema name of the routine
SPECIFIC_NAME	VARCHAR(128)	specific name of the routine
ROUTINE_CATALOG	VARCHAR(128)	routine catalog name of a routine contained in routine body of the SQL-invoked routine
ROUTINE_OWNER	VARCHAR(128)	routine owner name of a routine contained in routine body of the SQL-invoked routine
ROUTINE_SCHEMA	VARCHAR(128)	routine schema name of a routine contained in routine body of the SQL-invoked routine
ROUTINE_NAME	VARCHAR(128)	routine name of a routine contained in routine body of the SQL-invoked routine

ROUTINE_SEQUENCE_USAGE

Identify each external sequence generator owned by a given user or role on which some SQL routine defi ned in this catalog is dependent.

Table 9-192 Column information

Column name	Data type	Description
SPECIFIC_CATALOG	VARCHAR(128)	specific catalog name of the routine
SPECIFIC_OWNER	VARCHAR(128)	specific owner name of the routine
SPECIFIC_SCHEMA	VARCHAR(128)	specific schema name of the routine
SPECIFIC_NAME	VARCHAR(128)	specific name of the routine
SEQUENCE_CATALOG	VARCHAR(128)	catalog name of the sequence of contained in routine body of the SQL-invoked routine
SEQUENCE_OWNER	VARCHAR(128)	owner name of the sequence of contained in routine body of the SQL-invoked routine
SEQUENCE_SCHEMA	VARCHAR(128)	schema name of the sequence of contained in routine body of th e SQL-invoked routine
SEQUENCE_NAME	VARCHAR(128)	sequence name of contained in routine body of the SQL-invoked r outine

ROUTINE_TABLE_USAGE

Identify the tables owned by a given user or role on which SQL routines defined in this catalog are depen dent.

Table 9-193 Column information

Column name	Data type	Description
SPECIFIC_CATALOG	VARCHAR(128)	specific catalog name of the routine
SPECIFIC_OWNER	VARCHAR(128)	specific owner name of the routine
SPECIFIC_SCHEMA	VARCHAR(128)	specific schema name of the routine
SPECIFIC_NAME	VARCHAR(128)	specific name of the routine
TABLE_CATALOG	VARCHAR(128)	catalog name of the table of contained in routine body of the SQL-i nvoked routine
TABLE_OWNER	VARCHAR(128)	owner name of the table of contained in routine body of the SQL-in voked routine
TABLE_SCHEMA	VARCHAR(128)	schema name of the table of contained in routine body of the SQL-i nvoked routine
TABLE_NAME	VARCHAR(128)	table name of contained in routine body of the SQL-invoked routine

SCHEMATA

Identify the schemata in a catalog that are owned by given user or accessible to given user or role.

Table 9-194 Column information

Column name	Data type	Description
CATALOG_NAME	VARCHAR(128)	catalog name of the schema
SCHEMA_NAME	VARCHAR(128)	schema name
SCHEMA_OWNER	VARCHAR(128)	authorization name who owns the schema
DEFAULT_CHARACTER_SET_CATALOG	VARCHAR(128)	catalog name of the default character set for columns and domains in the schemata
DEFAULT_CHARACTER_SET_SCHEMA	VARCHAR(128)	schema name of the default character set for columns and domains in the schemata
DEFAULT_CHARACTER_SET_NAME	VARCHAR(128)	character set name of the default character set for columns and domains in the schemata
SQL_PATH	VARCHAR(1024)	character representation of schema path specification
CREATED_TIME	TIMESTAMP(6) WITHOUT TIME ZONE	created time of the schema
MODIFIED_TIME	TIMESTAMP(6) WITHOUT TIME ZONE	last modified time of the schema
COMMENTS	VARCHAR(1024)	comments of the schema

SEQUENCES

Identify the external sequence generators defined in this catalog that are accesible to a given user or role.

Table 9-195 Column information

Column name	Data type	Description
SEQUENCE_CATALOG	VARCHAR(128)	catalog name of the sequence
SEQUENCE_OWNER	VARCHAR(128)	owner name of the sequence
SEQUENCE_SCHEMA	VARCHAR(128)	schema name of the sequence
SEQUENCE_NAME	VARCHAR(128)	sequence name
DATA_TYPE	VARCHAR(128)	the standard name of the data type
NUMERIC_PRECISION	NUMBER	the numeric precision of the numerical data type
NUMERIC_PRECISION_RADIX	NUMBER	the radix (2 or 10) of the precision of the numerical data type
NUMERIC_SCALE	NUMBER	the numeric scale of the exact numerical data type
START_VALUE	NUMBER	the start value of the sequence generator
MINIMUM_VALUE	NUMBER	the minimum value of the sequence generator
MAXIMUM_VALUE	NUMBER	the maximum value of the sequence generator
INCREMENT	NUMBER	the increment of the sequence generator
CYCLE_OPTION	BOOLEAN	cycle option
CACHE_SIZE	NATIVE_INTEGER	number of sequence numbers to cache
DECLARED_DATA_TYPE	VARCHAR(128)	the data type name that a user declared
DECLARED_NUMERIC_PRECISION	NUMBER	the precision value that a user declared
DECLARED_NUMERIC_SCALE	NUMBER	the scale value that a user declared
CREATED_TIME	TIMESTAMP(6) WITHOUT TIME ZONE	created time of the sequence generator
MODIFIED_TIME	TIMESTAMP(6) WITHOUT TIME ZONE	last modified time of the sequence generator
COMMENTS	VARCHAR(1024)	comments of the sequence generator

SQL_FEATURES

List the features and subfeatures of this ISO/IEC 9075 standard, and indicate which of these the SQL-implementation supports.

Table 9-196 Column information

Column name	Data type	Description
FEATURE_ID	VARCHAR(32)	identifier string of the conformance element
FEATURE_NAME	VARCHAR(1024)	descriptive name of the conformance element
SUB_FEATURE_ID	VARCHAR(32)	identifier string of the subfeature, or a single space if not a subfeature
SUB_FEATURE_NAME	VARCHAR(1024)	descriptive name of the subfeature, or a single space if not a subfeature
IS_SUPPORTED	BOOLEAN	TRUE if an SQL-implementation fully supports that conformance element described when SQL-data in the identified catalog is accessed through that implementation, FALSE if not
IS_VERIFIED_BY	VARCHAR(1024)	If full support for the conformance element described has been verified by testing, then the IS_VERIFIED_BY column shall contain information identifying the conformance test used to verify the conformance claim; otherwise, IS_VERIFIED_BY shall be the null value
COMMENTS	VARCHAR(1024)	possibly a comment pertaining to the conformance element

SQL_IMPLEMENTATION_INFO

List the SQL-implementation information items defined in this ISO/IEC 9075 standard and, for each of the se, indicate the value supported by the SQL-implementation.

Table 9-197 Column information

Column name	Data type	Description
IMPLEMENTATION_INFO_ID	VARCHAR(32)	identifier string of the implementation information item
IMPLEMENTATION_INFO_NAME	VARCHAR(1024)	descriptive name of the implementation information item
INTEGER_VALUE	NATIVE_INTEGER	value of the implementation information item, or null if the value is contained in the column CHARACTER_VALUE
CHARACTER_VALUE	VARCHAR(32)	value of the implementation information item, or null if the value is contained in the column INTEGER_VALUE
COMMENTS	VARCHAR(1024)	possibly a comment pertaining to the implementation information it em

SQL_PACKAGES

List the packages of this ISO/IEC 9075 standard, and indicate which of these the SQL-implementation sup ports.

Table 9-198 Column information

Column name	Data type	Description
ID	VARCHAR(32)	identifier string of the conformance element
NAME	VARCHAR(1024)	descriptive name of the conformance element
IS_SUPPORTED	BOOLEAN	TRUE if an SQL-implementation fully supports that conformance ele ment described when SQL-data in the identified catalog is accessed through that implementation, FALSE if not
IS_VERIFIED_BY	VARCHAR(1024)	If full support for the conformance element described has been verif ied by testing, then the IS_VERIFIED_BY column shall contain inform ation identifying the conformance test used to verify the conforman ce claim; otherwise, IS_VERIFIED_BY shall be the null value
COMMENTS	VARCHAR(1024)	possibly a comment pertaining to the conformance element

SQL_PARTS

List the parts of this ISO/IEC 9075 standard, and indicate which of these the SQL-implementation supports.

Table 9-199 Column information

Column name	Data type	Description
ID	VARCHAR(32)	identifier string of the conformance element
NAME	VARCHAR(1024)	descriptive name of the conformance element
IS_SUPPORTED	BOOLEAN	TRUE if an SQL-implementation fully supports that conformance element described when SQL-data in the identified catalog is accessed through that implementation, FALSE if not
IS_VERIFIED_BY	VARCHAR(1024)	If full support for the conformance element described has been verified by testing, then the IS_VERIFIED_BY column shall contain information identifying the conformance test used to verify the conformance claim; otherwise, IS_VERIFIED_BY shall be the null value
COMMENTS	VARCHAR(1024)	possibly a comment pertaining to the conformance element

SQL_SIZING

List the sizing items of this ISO/IEC 9075 standard, for each of these, indicate the size supported by the SQL-implementation.

Table 9-200 Column information

Column name	Data type	Description
SIZING_ID	NATIVE_INTEGER	identifier of the sizing item
SIZING_NAME	VARCHAR(1024)	descriptive name of the sizing item
SUPPORTED_VALUE	NATIVE_INTEGER	value of the sizing item, or 0 if the size is unlimited or cannot be determined, or null if the features for which the sizing item is applicable are not supported
COMMENTS	VARCHAR(1024)	possibly a comment pertaining to the sizing item

STATISTICS

Provide a list of statistics about a single table and the indexes associated with the table that are accessible to a given user or role.

Table 9-201 Column information

Column name	Data type	Description
TABLE_CATALOG	VARCHAR(128)	catalog name of the table
TABLE_OWNER	VARCHAR(128)	owner name of the table
TABLE_SCHEMA	VARCHAR(128)	schema name of the table
TABLE_NAME	VARCHAR(128)	table name of the table
STAT_TYPE	VARCHAR(32)	statistics type: the value in (TABLE STAT, INDEX CLUSTERED, INDEX HASHED, INDEX OTHER)
NON_UNIQUE	BOOLEAN	indicates whether the index does not allow duplicate values
INDEX_CATALOG	VARCHAR(128)	catalog name of the index
INDEX_OWNER	VARCHAR(128)	owner name of the index
INDEX_SCHEMA	VARCHAR(128)	schema name of the index
INDEX_NAME	VARCHAR(128)	name of the index
COLUMN_NAME	VARCHAR(128)	column name that participates in the index
ORDINAL_POSITION	NUMBER	ordinal position of the specific column in the index described
IS_ASCENDING_ORDER	BOOLEAN	index key column being described is sorted in ASCENDING(TRUE) or DESCENDING(FALSE) order
IS_NULLS_FIRST	BOOLEAN	the null values of the key column are sorted before(TRUE) or after(FALSE) non-null values
CARDINALITY	NUMBER	if STAT_TYPE is (TABLE TYPE), then this is the number of rows in the table; otherwise, it is the number of unique values in the index
PAGES	NUMBER	if STAT_TYPE is (TABLE TYPE), then this is the number of pages used for the table; otherwise, it is the number of pages used for the current index.
FILTER_CONDITION	VARCHAR(1024)	filter condition, if any.
COMMENTS	VARCHAR(1024)	if STAT_TYPE is (TABLE TYPE), then this is the table comments; otherwise, it is the index comments.

TABLES

Identify the tables defined in this catalog that are accessible to a given user or role

Table 9-202 Column information

Column name	Data type	Description
TABLE_CATALOG	VARCHAR(128)	catalog name of the table
TABLE_OWNER	VARCHAR(128)	owner name of the table
TABLE_SCHEMA	VARCHAR(128)	schema name of the table
TABLE_NAME	VARCHAR(128)	table name of the table
TABLE_TYPE	VARCHAR(32)	the value is in (BASE TABLE, VIEW, GLOBAL TEMPORARY, LOCAL TEMPORARY, SYSTEM VERSIONED, FIXED TABLE, DUMP TABLE)
DBC_TABLE_TYPE	VARCHAR(32)	ODBC/JDBC table type: the value is in (TABLE, VIEW, GLOBAL TEMPORARY, LOCAL TEMPORARY, IMMUTABLE TABLE, SYSTEM TABLE, ALIAS, SYNONYM)
TABLESPACE_NAME	VARCHAR(128)	tablespace name of the table, NULL if view
SYSTEM_VERSION_START_COLUMN_NAME	VARCHAR(128)	if the table is a system-versioned table, then the name of the system-version start column of the table
SYSTEM_VERSION_END_COLUMN_NAME	VARCHAR(128)	if the table is a system-versioned table, then the name of the system-version end column of the table
SYSTEM_VERSION_RETENTION_PERIOD	VARCHAR(32)	if the table is a system-versioned table, then the character representation of the value of the retention period of the table
SELF_REFERENCING_COLUMN_NAME	VARCHAR(128)	if the table is a typed table, then the name of the self-referencing column of the table
REFERENCE_GENERATION	VARCHAR(32)	if the table has a self-referencing column, the value is in (SYSTEM GENERATED, USER GENERATED, DERIVED)
USER_DEFINED_TYPE_CATALOG	VARCHAR(128)	if the table being described is a table of a structured type, the catalog name of the structured type
USER_DEFINED_TYPE_SCHEMA	VARCHAR(128)	if the table being described is a table of a structured type, the schema name of the structured type
USER_DEFINED_TYPE_NAME	VARCHAR(128)	if the table being described is a table of a structured type, the name of the structured type
IS_INSERTABLE_INTO	BOOLEAN	is an insertable-into table
IS_TYPED	BOOLEAN	is a typed table
COMMIT_ACTION	VARCHAR(32)	if the table is a temporary table, the value is in (DELETE, PRESERVE)
CREATED_TIME	TIMESTAMP(6) WITHOUT TIME ZONE	created time of the table
MODIFIED_TIME	TIMESTAMP(6) WITHOUT TIME ZONE	last modified time of the table
COMMENTS	VARCHAR(1024)	comments of the table

TABLE_CONSTRAINTS

Identify the table constraints defined on tables in this catalog that are accessible to a given user or role

Table 9-203 Column information

Column name	Data type	Description
CONSTRAINT_CATALOG	VARCHAR(128)	catalog name of the constraint
CONSTRAINT_OWNER	VARCHAR(128)	authorization name who owns the constraint
CONSTRAINT_SCHEMA	VARCHAR(128)	schema name of the constraint being described
CONSTRAINT_NAME	VARCHAR(128)	constraint name
TABLE_CATALOG	VARCHAR(128)	catalog name of the table to which the table constraint being described applies
TABLE_OWNER	VARCHAR(128)	authorization name who owns the table to to which the table constraint being described applies
TABLE_SCHEMA	VARCHAR(128)	schema name of the table to to which the table constraint bei ng described applies
TABLE_NAME	VARCHAR(128)	table name of the table to to which the table constraint being described applies
CONSTRAINT_TYPE	VARCHAR(32)	the value is in (PRIMARY KEY, UNIQUE, FOREIGN KEY, NOT N ULL, CHECK)
IS_DEFERRABLE	BOOLEAN	is a deferrable constraint
INITIALLY_DEFERRED	BOOLEAN	is an initially deferred constraint
ENFORCED	BOOLEAN	is an enforced constraint
CREATED_TIME	TIMESTAMP(6) WITH OUT TIME ZONE	created time of the constraint
MODIFIED_TIME	TIMESTAMP(6) WITH OUT TIME ZONE	last modified time of the constraint
COMMENTS	VARCHAR(1024)	comments of the constraint

TABLE_PRIVILEGES

Identify the privileges on tables of tables defined in this catalog that are available to or granted by a given user or role.

Table 9-204 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	authorization name of the user who granted table privileges
GRANTEE	VARCHAR(128)	authorization name of some user or role, or PUBLIC to indicate all users, to whom the table privilege being described is granted
TABLE_CATALOG	VARCHAR(128)	catalog name of the table on which the privilege being described was granted
TABLE_OWNER	VARCHAR(128)	table owner name of the table on which the privilege being described was granted
TABLE_SCHEMA	VARCHAR(128)	schema name of the table on which the privilege being described was granted
TABLE_NAME	VARCHAR(128)	table name on which the privilege being described was granted
PRIVILEGE_TYPE	VARCHAR(32)	the value is in (CONTROL, SELECT, INSERT, UPDATE, DELETE, REFERENCES, LOCK, INDEX, ALTER)
IS_GRANTABLE	BOOLEAN	is grantable
WITH_HIERARCHY	BOOLEAN	whether the privilege was granted WITH HIERARCHY OPTION or not

USAGE_PRIVILEGES

Identify the USAGE privileges on objects defined in this catalog that are available to or granted by a given user or role.

Table 9-205 Column information

Column name	Data type	Description
GRANTOR	VARCHAR(128)	authorization name of the user who granted usage privileges, on the object of the type identified by OBJECT_TYPE
GRANTEE	VARCHAR(128)	authorization identifier of some user or role, or PUBLIC to indicate all users, to whom the usage privilege being described is granted
OBJECT_CATALOG	VARCHAR(128)	catalog name of the object of the type identified by OBJECT_TYPE on which the privilege being described was granted
OBJECT_OWNER	VARCHAR(128)	owner name of the object of the type identified by OBJECT_TYPE on which the privilege being described was granted
OBJECT_SCHEMA	VARCHAR(128)	schema name of the object of the type identified by OBJECT_TYPE on which the privilege being described was granted
OBJECT_NAME	VARCHAR(128)	object name of the type identified by OBJECT_TYPE on which the privilege being described was granted
OBJECT_TYPE	VARCHAR(32)	the value is in (DOMAIN, CHARACTER SET, COLLATION, TRANSLATION, SEQUENCE)
PRIVILEGE_TYPE	VARCHAR(32)	the value is in (USAGE)
IS_GRANTABLE	BOOLEAN	is grantable

VIEWS

Identify the viewed tables defined in this catalog that are accessible to a given user or role.

Table 9-206 Column information

Column name	Data type	Description
TABLE_CATALOG	VARCHAR(128)	catalog name of the viewed table
TABLE_OWNER	VARCHAR(128)	owner name of the viewed table
TABLE_SCHEMA	VARCHAR(128)	schema name of the viewed table
TABLE_NAME	VARCHAR(128)	view name of the viewed table
VIEW_DEFINITION	LONG VARCHAR	the character representation of the user-specified query expression contained in the corresponding view descriptor
CHECK_OPTION	VARCHAR(32)	the value is in (CASCADED, LOCAL, NONE)
IS_UPDATABLE	BOOLEAN	is an updatable view
INSERTABLE_INTO	BOOLEAN	is an insertable view
IS_TRIGGER_UPDATABLE	BOOLEAN	whether an update INSTEAD OF trigger is defined on the view or not
IS_TRIGGER_DELETABLE	BOOLEAN	whether a delete INSTEAD OF trigger is defined on the view or not
IS_TRIGGER_INSERTABLE_INTO	BOOLEAN	whether an insert INSTEAD OF trigger is defined on the view or not
IS_COMPILED	BOOLEAN	whether the view is compiled or not
IS_AFFECTED	BOOLEAN	whether the view is affected by modification of underlying object or not
COMMENTS	VARCHAR(1024)	comments of the view

VIEW_MODULE_USAGE

Identify the SQL-server modules owned by a given user or role on which views defined in this catalog are dependent.

Table 9-207 Column information

Column name	Data type	Description
TABLE_CATALOG	VARCHAR(128)	catalog name of the viewed table
TABLE_OWNER	VARCHAR(128)	owner name of the viewed table
TABLE_SCHEMA	VARCHAR(128)	schema name of the viewed table
TABLE_NAME	VARCHAR(128)	view name of the viewed table
MODULE_CATALOG	VARCHAR(128)	catalog name of the SQL-server module of contained in definition te xt of the view
MODULE_OWNER	VARCHAR(128)	owner name of the SQL-server module of contained in definition tex t of the view
MODULE_SCHEMA	VARCHAR(128)	schema name of the SQL-server module of contained in definition te xt of the view'
MODULE_NAME	VARCHAR(128)	SQL-server module name of contained in definition text of the view

VIEW_ROUTINE_USAGE

Identify each routine owned by a given user or role on which a view defined in this catalog is dependent.

Table 9-208 Column information

Column name	Data type	Description
TABLE_CATALOG	VARCHAR(128)	catalog name of the viewed table
TABLE_OWNER	VARCHAR(128)	owner name of the viewed table
TABLE_SCHEMA	VARCHAR(128)	schema name of the viewed table
TABLE_NAME	VARCHAR(128)	view name of the viewed table
SPECIFIC_CATALOG	VARCHAR(128)	specific catalog name of a routine contained in the query expression of the view being described
SPECIFIC_OWNER	VARCHAR(128)	specific owner name of a routine contained in the query expression of the view being described
SPECIFIC_SCHEMA	VARCHAR(128)	specific schema name of a routine contained in the query expression of the view being described
SPECIFIC_NAME	VARCHAR(128)	specific name of a routine contained in the query expression of the v iew being described

VIEW_TABLE_USAGE

Identify the tables on which viewed tables defined in this catalog and owned by a given user or role are dependent.

Table 9-209 Column information

Column name	Data type	Description
VIEW_CATALOG	VARCHAR(128)	catalog name of the viewed table
VIEW_OWNER	VARCHAR(128)	owner name of the viewed table
VIEW_SCHEMA	VARCHAR(128)	schema name of the viewed table
VIEW_NAME	VARCHAR(128)	view name of the viewed table
TABLE_CATALOG	VARCHAR(128)	catalog name of a table that is explicitly or implicitly referenced in the original query expression of the compiled view being described
TABLE_OWNER	VARCHAR(128)	owner name of a table that is explicitly or implicitly referenced in the original query expression of the compiled view being described
TABLE_SCHEMA	VARCHAR(128)	schema name of a table that is explicitly or implicitly referenced in the original query expression of the compiled view being described
TABLE_NAME	VARCHAR(128)	table name of a table that is explicitly or implicitly referenced in the original query expression of the compiled view being described

9.3 PERFORMANCE_VIEW_SCHEMA

PERFORMANCE_VIEW_SCHEMA schema consists of views which can retrieve the current state of the syst em.

Execute *PerformanceViewSchema.sql* as follows to use the views.

- For standalone

```
% gsql sys gliese --as sysdba --import
$GOLDILOCKS_HOME/admin/standalone/PerformanceViewSchema.sql
```

- For cluster

```
% gsql sys gliese --as sysdba --import
$GOLDILOCKS_HOME/admin/cluster/PerformanceViewSchema.sql
```

The retrievable information of PERFORMANCE_VIEW_SCHEMA views vary upon its phase of the startup. (nomount, mount, open)
Execute the followings to figure out the startup phase in which each views are retrieved.

```
gSQL> select table_name, startup_phase from v$tables order by 1;
TABLE_NAME                 STARTUP_PHASE
-------------------------- -------------
V$AGABLE_INFO              OPEN
V$ARCHIVELOG               MOUNT
V$AUDITABLE_DB_PRIVILEGES  NO_MOUNT
V$AUDITABLE_SYSTEM_ACTIONS NO_MOUNT
V$BACKUP                   MOUNT
V$BALANCER                 OPEN
V$BCH                      MOUNT
V$BUFFER_STAT              MOUNT
V$COLUMNS                  OPEN
V$CONTROLFILE              MOUNT
V$DATAFILE                 MOUNT
V$DB_CHANGE_TRACKING       MOUNT
V$DB_FILE                  MOUNT
V$DB_PROPERTY              OPEN
V$DISPATCHER               OPEN
V$ERROR_CODE               NO_MOUNT
V$INCREMENTAL_BACKUP       MOUNT
```

```
V$INSTANCE                NO_MOUNT
V$KEYWORDS                NO_MOUNT
V$LATCH                   NO_MOUNT
V$LOCK_WAIT               OPEN
TABLE_NAME             STARTUP_PHASE
---------------------- -------------
V$LOGFILE              MOUNT
V$OPEN_CURSOR          NO_MOUNT
V$PLAN_HISTORY         OPEN
V$PLAN_HISTORY_LATEST  OPEN
V$PROCESS_MEM_STAT     NO_MOUNT
V$PROCESS_SQL_STAT     NO_MOUNT
V$PROCESS_STAT         NO_MOUNT
V$PROPERTY             NO_MOUNT
V$PROPERTY_ALIAS       NO_MOUNT
V$PSM_RESERVED_WORDS   NO_MOUNT
V$QUEUE                OPEN
V$RESERVED_WORDS       NO_MOUNT
V$SEQUENCE             OPEN
V$SESSION              NO_MOUNT
V$SESSION_AUDIT        OPEN
V$SESSION_CONNECT_INFO NO_MOUNT
V$SESSION_EVENT        OPEN
V$SESSION_MEM_STAT     NO_MOUNT
V$SESSION_SQL_STAT     NO_MOUNT
V$SESSION_STAT         NO_MOUNT
V$SESSION_WAIT         OPEN
TABLE_NAME              STARTUP_PHASE
---------------------- -------------
V$SHARED_MODE           OPEN
V$SHARED_SERVER         OPEN
V$SHM_SEGMENT           NO_MOUNT
V$SPROPERTY             NO_MOUNT
V$SQLFN_METADATA        NO_MOUNT
V$SQL_CACHE             NO_MOUNT
V$SQL_COMMAND           NO_MOUNT
V$SQL_HISTORY           NO_MOUNT
V$STATEMENT             NO_MOUNT
V$SYSTEM_EVENT          OPEN
V$SYSTEM_MEM_STAT       NO_MOUNT
V$SYSTEM_SQL_STAT       NO_MOUNT
```

```
V$SYSTEM_STAT           NO_MOUNT
V$TABLES                NO_MOUNT
V$TABLESPACE            MOUNT
V$TABLESPACE_STAT       OPEN
V$TRANSACTION           OPEN
V$WAIT_EVENT_CLASS_NAME OPEN
V$WAIT_EVENT_NAME       OPEN
V$XA_TRANSACTION        OPEN
62 rows selected.
```

GV$ Global View

The cluster provides GV$ view corresponding to almost every V$ views.
V$ view retrieves the information of the currently connected server, but GV$ view retrieves the informati on of all servers.
GV$ view includes all column information of V$ view, and it additionally has ORIGIN_MEMBER_NAME co lumn which is a server having acquired the data (cluster member).


It is available only on a cluster.

For example, V$TRANSACTION information retrieves the transaction information of the currently connect ed server as follows.

```
gSQL> SELECT TRANS_ID, SESSION_ID, TRANS_VIEW_SCN, START_TIME FROM V$TRANSACTION;
TRANS_ID SESSION_ID TRANS_VIEW_SCN START_TIME
-------- ---------- -------------- --------------------------
40501296         48 1098.1.26      2017-04-07 17:14:01.912637
```

On the other hand, GV$TRANSACTION information retrieves the transaction information of all servers as follows.

```
gSQL> SELECT ORIGIN_MEMBER_NAME, TRANS_ID, SESSION_ID, TRANS_VIEW_SCN, START_TIME FROM
GV$TRANSACTION;
ORIGIN_MEMBER_NAME TRANS_ID SESSION_ID TRANS_VIEW_SCN START_TIME
------------------ -------- ---------- -------------- --------------------------
G1N1               40501296         48 1098.1.26      2017-04-07 17:14:01.912637
G2N2               40304688         48 1098.0.888     2017-04-07 17:14:55.134015
G2N1               42205232         48 1098.0.888     2017-04-07 17:14:55.135996
G1N2               40435760         48 1098.1.889     2017-04-07 17:14:01.910138
```

In the example above, ORIGIN_MEMBER_NAME information acquired each transaction information from the cluster member corresponding to G1N1, G2N1, G1N2, G2N2 each.

The information of a specific remote server can be retrieved by using the condition for ORIGIN_MEMBER_ NAME column as follows.

```
gSQL>
SELECT ORIGIN_MEMBER_NAME, TRANS_ID, SESSION_ID, TRANS_VIEW_SCN, START_TIME
  FROM GV$TRANSACTION
WHERE ORIGIN_MEMBER_NAME IN ( 'G2N1', 'G3N2' );
```

```
ORIGIN_MEMBER_NAME TRANS_ID SESSION_ID TRANS_VIEW_SCN START_TIME
------------------ -------- ---------- -------------- --------------------------
G3N2               32178224         48 1099.0.888     2017-04-07 17:33:10.934752
G2N1               42270768         48 1099.0.888     2017-04-07 17:31:14.726007
2 rows selected.
```

V$AGABLE_INFO

The V$AGABLE_INFO displays the system agable information.

Table 9-210 Column information

Column name	Data type	Description
SCN	VARCHAR(32)	system scn
AGABLE_SCN	VARCHAR(32)	system agable scn
AGABLE_SCN_GAP	VARCHAR(32)	gap between system scn and agable scn
OLDEST_SESSION_ID	NUMBER	identifier of session blocking aging

V$ARCHIVELOG

The V$ARCHIVELOG displays information of log archiving.

Table 9-211 Column information

Column name	Data type	Description
ARCHIVELOG_MODE	VARCHAR(32)	database log mode: the value in (NOARCHIVELOG, ARCHIVELOG)
LAST_ARCHIVED_LOG	NUMBER	sequence number of last archived log file
ARCHIVELOG_DIR	VARCHAR(1024)	archive destination path
ARCHIVELOG_FILE_PREFIX	VARCHAR(128)	file prefix name of the archived log

V$AUDITABLE_DB_PRIVILEGES

The V$AUDITABLE_DB_PRIVILEGES displays auditable database privileges.

Table 9-212 Column information

Column name	Data type	Description
PRIVILEGE_ID	NUMBER	database privilege identifier
PRIVILEGE_NAME	VARCHAR(128)	database privilege name

V$AUDITABLE_SYSTEM_ACTIONS

The V$AUDITABLE_SYSTEM_ACTIONS displays auditable system actions.

Table 9-213 Column information

Column name	Data type	Description
ACTION_ID	NUMBER	auditable system action identifier
ACTION_NAME	VARCHAR(128)	auditable system action name

V$BACKUP

The V$BACKUP displays information of backup.

Table 9-214 Column information

Column name	Data type	Description
TBS_NAME	VARCHAR(128)	tablespace name
BACKUP_STATUS	VARCHAR(16)	indicates whether the tablespace begin backup (ACTIVE) or not (INACTIVE)
BACKUP_LSN	NUMBER	the last checkpoint lsn of tablespace when backup started

V$BALANCER

The V$BALANCER displays information of balancer.

Table 9-215 Column information

Column name	Data type	Description
PROCESS_ID	NUMBER	balancer process identifier
CUR_CONNECTIONS	NUMBER	current number of connections
CONNECTIONS	NUMBER	total number of connections
CONNECTIONS_HIGHWATER	NUMBER	highest number of connections
MAX_CONNECTIONS	NUMBER	maximum connections
STATUS	VARCHAR(16)	status

V$BCH

The V$BCH displays information of database buffer control header array.

Table 9-216 Column information

Column name	Data type	Description
BCH_SEQ	NUMBER	bch sequence
TABLESPACE_ID	NUMBER	tablespace identifier of the page cached in the frame of bch
PAGE_ID	NUMBER	page identifier of the page cached in the frame of bch
LOGICAL_ADDRESS	VARCHAR(18)	logical address of the frame of bch
DIRTY	BOOLEAN	dirty state of the page cached in the frame of bch
PGAE_TYPE	VARCHAR(20)	page type of the page cached in the frame of bch
FIRST_DIRTY_LSN	NUMBER	first dirty lsn of the page cached in the frame of bch
RECOVERY_LSN	NUMBER	recovery lsn of the page cached in the frame of bch
LAST_FLUSHED_LSN	NUMBER	last flushed lsn of the page cached in the frame of bch
FIXED_COUNT	NUMBER	fixed count of the page cached in the frame of bch
TOUCHED_COUNT	NUMBER	touched count of the page cached in the frame of bch
RECENT_TOUCH_COUNT_INCREASED_TIME	TIMESTAMP(6) WITHOUT TIME ZONE	timestamp that touch count of the page cached in the frame of bch increased most recently
BCH_LIST_TYPE	VARCHAR(16)	list type to which the bch belongs
BCH_STATE	VARCHAR(16)	bch state

V$BUFFER_STAT

The V$BUFFER_STAT displays database buffer statistics.

Table 9-217 Column information

Column name	Data type	Description
BUFFER_POOL_SIZE	NUMBER	total buffer frame size (page count)
HASH_BUCKET_COUNT	NUMBER	buffer hash bucket count
LRU_LIST_COUNT	NUMBER	buffer lru list count
HOT_REGION_PERCENTAGE	NUMBER	percentage of lru hot region
HOT_REGION_CRITERIA	NUMBER	touch count criteria of lru hot region
CHECKPOINT_LIST_COUNT	NUMBER	buffer checkpoint list count
FLUSH_LIST_COUNT	NUMBER	buffer flush list count
FREE_LIST_COUNT	NUMBER	buffer free list count
FREE_BUFFER_WAIT	NUMBER	total number of waiting for free list
READ_COMPLETE_WAIT	NUMBER	total number of waiting for read page complete
BUFFER_LOOKUPS	NUMBER	total number of lookups in the buffer for requested pages
BUFFER_HIT	NUMBER	total number of hits in the buffer for requested pages
BUFFER_MISS	NUMBER	total number of misses in the buffer for requested pages
TOTAL_WRITES	NUMBER	total number of physical writes
TOTAL_READS	NUMBER	total number of physical reads
FLUSH_PER_SECOND	NUMBER	total number of disk writes per one second
READ_PER_SECOND	NUMBER	total number of disk reads per one second
AVERAGE_WRITE_LATENCY	NUMBER	average latency of disk writes
AVERAGE_READ_LATENCY	NUMBER	average latency of disk reads

V$CLUSTER_DISPATCHER

The V$CLUSTER_DISPATCHER displays cluster dispatcher information.

It is available only on a cluster.

Table 9-218 Column information

Column name	Data type	Description
DISPATCHER_ID	NUMBER	dispatcher identifier
IS_SYNC	BOOLEAN	whether the dispatcher is sync or not
RX_BYTES	NUMBER	total amount of data that has received through the dispatcher
TX_BYTES	NUMBER	total amount of data that has transmitted through the dispatcher
RX_JOBS	NUMBER	the total number of jobs received
TX_JOBS	NUMBER	the total number of jobs transmitted

V$CLUSTER_LOCATION

The V$CLUSTER_LOCATION displays cluster location information.


It is available only on a cluster.

Table 9-219 Column information

Column name	Data type	Description
MEMBER_NAME	VARCHAR(128)	member name
HOST	VARCHAR(128)	host name or IP address of a member
PORT	NUMBER	host port of a member

V$CLUSTER_MEMBER

The V$CLUSTER_MEMBER displays cluster member information.

It is available only on a cluster.

Table 9-220 Column information

Column name	Data type	Description
MEMBER_ID	NUMBER	member identifier
MEMBER_POSITION	NUMBER	member position
STATUS	VARCHAR(64)	status of the member: the value in (ACTIVE, INACTIVE)
IS_GLOBAL_COORD	BOOLEAN	indicates whether a member is global coordnator (TRUE) or not (FALSE)
IS_GROUP_COORD	BOOLEAN	indicates whether a member is group coordnator (TRUE) or not (FALSE)

V$COLUMNS

The V$COLUMNS has one row for each column of all the performance views (views beginning with V$).

Execute \desc as follows to retrieve the column information of performance view in nomount or mount p hase in which V$COLUMNS is not available.

```
gSQL> \desc V$INSTANCE
COLUMN_NAME     TYPE                            IS_NULLABLE
--------------- ------------------------------- -----------
RELEASE_VERSION VARCHAR(64)          FALSE
STARTUP_TIME    TIMESTAMP(6) WITHOUT TIME ZONE FALSE
INSTANCE_STATUS VARCHAR(16)          FALSE
```

Table 9-221 Column information

Column name	Data type	Description
TABLE_OWNER	VARCHAR(128)	owner name who owns the performance view
TABLE_SCHEMA	VARCHAR(128)	schema name of the performance view
TABLE_NAME	VARCHAR(128)	name of the performance view
COLUMN_NAME	VARCHAR(128)	column name
ORDINAL_POSITION	NUMBER	the ordinal position (> 0) of the column in the performance view
DATA_TYPE	VARCHAR(128)	the data type name that a user declared
DATA_PRECISION	NUMBER	the precision value that a user declared
DATA_SCALE	NUMBER	the scale value that a user declared
COMMENTS	VARCHAR(1024)	comments of the column

V$CONTROLFILE

This view displays information about GOLDILOCKS control files.

Table 9-222 Column information

Column name	Data type	Description
STATUS	VARCHAR(16)	control file status (VALID, CORRUPTED)
CONTROLFILE_NAME	VARCHAR(1152)	control file name (absolute path)
LAST_CHECKPOINT_LSN	NATIVE_BIGINT	the last checkpoint lsn
IS_PRIMARY	BOLLEAN	indicates whether the control file is primary

V$DATAFILE

The V$DATAFILE displays information of all datafiles.

Table 9-223 Column information

Column name	Data type	Description
TBS_NAME	VARCHAR(128)	tablepsace name
DATAFILE_NAME	VARCHAR(1024)	datafile name (absolute path)
CHECKPOINT_LSN	NUMBER	LSN at last checkpoint (null if temporary tablespace)
CREATION_TIME	TIMESTAMP(6) WITH OUT TIME ZONE	timestamp of the datafile creation
FILE_SIZE	NUMBER	datafile size (in bytes)
LOADED_CHECKPOINT_LSN	NUMBER	checkpoint LSN of the datafile loaded in memory
CORRUPT_PAGE_COUNT	NUMBER	number of corrupt pages in the datafile

V$DB_CHANGE_TRACKING

The V$DB_CHANGE_TRACKING displays information of database change tracking.

Table 9-224 Column information

Column name	Data type	Description
TABLESPACE_ID	NUMBER	tablespage identifier
DATAFILE_ID	NUMBER	datafile identifier
CHANGE_TRACKING_STATE	VARCHAR(32)	state of dtafile change tracking
CHANGE_TRACKING_CHUNK_SEQ	NUMBER	sequence of change tracking chunk for datafile
MAX_SIZE	NUMBER	maximum size of datafile (byte)
BITMAP_BLOCK_COUNT	NUMBER	bitmap block count of change tracking chunk
LAST_PAGE_SEQ	NUMBER	the last page sequence of change tracking chunk

V$DB_FILE

The V$DB_FILE displays a list of all files using in database.

Table 9-225 Column information

Column name	Data type	Description
FILE_NAME	VARCHAR(1024)	file name
FILE_TYPE	VARCHAR(16)	file type

V$DB_PROPERTY

The V$DB_PROPERTY displays a list of permanent property.

Table 9-226 Column information

Column name	Data type	Description
PROPERTY_NAME	VARCHAR(128)	name of the property
DESCRIPTION	VARCHAR(2048)	description of the property
DATA_TYPE	VARCHAR(32)	data type of the property
VALUE_UNIT	VARCHAR(32)	unit of the property value: the value in (NONE, BYTE, MS(milisec))
PROPERTY_VALUE	VARCHAR(2048)	property value for the session. otherwise, the instance-wide value
MIN_VALUE	NUMBER	minimum value for property. null if type is varchar
MAX_VALUE	NUMBER	maximum value for property. null if type is varchar
SES_MODIFIABLE	VARCHAR(32)	property can be changed with ALTER SESSION or not: the value in (TRUE, FALSE)
SYS_MODIFIABLE	VARCHAR(32)	property can be changed with ALTER SYSTEM and when the change takes effect: the value in (NONE, FALSE, IMMEDIATE, DEFERRED)
IS_MODIFIABLE	VARCHAR(32)	property can be changed or not: the value in (TRUE, FALSE)
IS_DEPRECATED	VARCHAR(32)	whether a property is deprecated or not: the value in (TRUE, FALSE)
IS_GLOBAL	VARCHAR(32)	whether a property scope is global or not: the value in (TRUE, FALSE)

V$DISPATCHER

The V$DISPATCHER displays information of dispatchers.

Table 9-227 Column information

Column name	Data type	Description
PROCESS_ID	NUMBER	dispatcher process identifier
RESPONSE_JOB_COUNT	NUMBER	response job count
ACCEPT	NUMBER	indicates whether this dispatcher is accepting new connections
START_TIME	NUMBER	process start time
CUR_CONNECTIONS	NUMBER	current number of connections
CONNECTIONS	NUMBER	total number of connections
CONNECTIONS_HIGHWATER	NUMBER	highest number of connections
MAX_CONNECTIONS	NUMBER	maximum connections
RECV_STATUS	VARCHAR(16)	receive status
RECV_BYTES	NUMBER	total bytes of received
RECV_UNITS	NUMBER	total units of received
RECV_IDLE	NUMBER	total idle time of receive (1/100 second)
RECV_BUSY	NUMBER	total busy time of receive (1/100 second)
SEND_STATUS	VARCHAR(16)	send status
SEND_BYTES	NUMBER	total bytes of sent
SEND_UNITS	NUMBER	total units of sent
SEND_IDLE	NUMBER	total idle time of send (1/100 second)
SEND_BUSY	NUMBER	total busy time of send (1/100 second)

V$ERROR_CODE

The V$ERROR_CODE displays a list of all GOLDILOCKS error codes.

Table 9-228 Column information

Column name	Data type	Description
ERROR_CODE	NUMBER	GOLDILOCKS error code
SQL_STATE	VARCHAR(32)	standard SQLSTATE code
ERROR_MESSAGE	VARCHAR(1024)	error message

V$GLOBAL_TRANSACTION

The V$GLOBAL_TRANSACTION displays information on the currently active global transactions.

Table 9-229 Column information

Column name	Data type	Description
GLOBAL_TRANS_ID	VARCHAR(1024)	global transaction identifier
LOCAL_TRANS_ID	NUMBER	local transaction identifier
GLOBAL_TRANS_STATE	VARCHAR(32)	state of the global transaction: the value in (NOTR, ACTIVE, IDLE, PREPARED, ROLLBACK_ONLY, HEURISTIC_COMPLETED)
ASSO_STATE	VARCHAR(32)	associate state of the global transaction: the value in (NOT_ASSOCIATED, ASSOCIATED, ASSOCIATION_SUSPENDED)
START_TIME	TIMESTAMP(6) WITHOUT TIME ZONE	global transaction start time
IS_REPREPARABLE	BOOLEAN	indicates whether the global transaction is repreparable

V$INCREMENTAL_BACKUP

The V$INCREMENTAL_BACKUP displays information about control files and datafiles in backup sets from the control file.

Table 9-230 Column information

Column name	Data type	Description
BACKUP_NAME	VARCHAR(1024)	backup file name (absolute path)
BACKUP_SCOPE	VARCHAR(128)	incremental backup scope: the value in (database, tablespace, control)
INCREMENTAL_LEVEL	NUMBER	incremental backup level: the value in (0, 1, 2, 3, 4)
INCREMENTAL_TYPE	VARCHAR(32)	incremental backup type: the value in (DIFFERENTIAL, CUMULATIVE)
LSN	NUMBER	all changes up to checkpoint LSN are included in this backup
BEGIN_TIME	TIMESTAMP(6) WITHOUT TIME ZONE	incremental backup beginning time
COMPLETION_TIME	TIMESTAMP(6) WITHOUT TIME ZONE	incremental backup completion time

V$INSTANCE

This view displays the state of the current instance.

Table 9-231 Column information

Column name	Data type	Description
RELEASE_VERSION	VARCHAR(64)	release version
STARTUP_TIME	TIMESTAMP(6) WITH OUT TIME ZONE	time when the instance was started
INSTANCE_STATUS	VARCHAR(16)	status of the instance: the value in (STARTED, MOUNTED, OPEN)

V$JOURNALING

The V$JOURNALING displays journaling information.


It is available only on a cluster.

Table 9-232 Column information

Column name	Data type	Description
TABLE_NAME	VARCHAR(128)	table name
SHARD_ID	NUMBER	shard identifier
RECORD_COUNT	NUMBER	journaled record count
TOTAL_SIZE	NUMBER	total size of journaled records (byte)

V$KEYWORDS

The V$KEYWORDS displays a list of all SQL keywords.

Table 9-233 Column information

Column name	Data type	Description
KEYWORD_NAME	VARCHAR(128)	name of keyword
KEYWORD_LENGTH	NUMBER	length of the keyword
IS_RESERVED	BOOLEAN	indicates whether the keyword cannot be used as an identifier (TRUE) or whether the keyword is not reserved (FALSE)

V$LATCH

The V$LATCH shows latch information.

Table 9-234 Column information

Column name	Data type	Description
LATCH_DESCRIPTION	VARCHAR(64)	latch description
REF_COUNT	NUMBER	reference count
SPIN_LOCK	VARCHAR(3)	indicates whether the spin lock is locked (YES) or not (NO)
WAIT_COUNT	NUMBER	wait count
CURRENT_MODE	VARCHAR(32)	current latch mode: the value in (INITIAL, SHARED, EXCLUSIVE)

V$LICENSE

The V$LICENSE displays information of current license.

Table 9-235 Column information

Column name	Data type	Description
LICENSE_TYPE	VARCHAR(8)	license type
START_DATE	DATE	start date of the license
EXPIRE_DATE	DATE	expire date of the license

V$LOGFILE

The V$LOGFILE displays information of all redo log members.

Table 9-236 Column information

Column name	Data type	Description
GROUP_ID	NUMBER	redo log group identifier
FILE_NAME	VARCHAR(1024)	name of the log member
GROUP_STATE	VARCHAR(32)	state of the log group: the value in (UNUSED, ACTIVE, CURRENT, IN ACTIVE)
FILE_SEQ	NUMBER	file sequence number of the log member
FILE_SIZE	NUMBER	file size of the log member (in bytes)

V$LOCK_WAIT

This view lists the locks currently held and outstanding requests for a lock.

Table 9-237 Column information

Column name	Data type	Description
GRANT_TRANS_ID	NUMBER	transaction identifier that holds the lock
REQUEST_TRANS_ID	NUMBER	transaction identifier that requests the lock

V$LOCKED_OBJECT

This view shows locked object information.

Table 9-238 Column information

Column name	Data type	Description
LOCK_SLOT_ID	NUMBER	lock slot identifier
TABLE_OWNER	VARCHAR(128)	owner name who owns the locked table
TABLE_SCHEMA	VARCHAR(128)	schema of the locked table
TABLE_NAME	VARCHAR(128)	locked table name
LOCK_MODE	VARCHAR(8)	granted lock mode (IS, IX, S, X, SIX)

V$OPEN_CURSOR

The view lists display cursor status information for each current session.

Table 9-239 Column information

Column name	Data type	Description
SESSION_ID	NUMBER	ID of the session
USER_NAME	VARCHAR(128)	NAME of the user
CURSOR_NAME	VARCHAR(128)	NAME of the cursor
PSM_CURSOR_ID	NUMBER	ID of the PSM cursor
SQL_TEXT	LONG VARCHAR	SQL text for the cursor
IS_PSM_CURSOR	BOOLEAN	is PSM cursor
IS_OPEN	BOOLEAN	is open
OPEN_TIME	TIMESTAMP(6) WITHOUT TIME ZONE	cursor open time
LAST_EXEC_TIME	NATIVE_BIGINT	last execution time(us)
IS_SENSITIVE	BOOLEAN	is sensitive
IS_SCROLLABLE	BOOLEAN	is scrollable
IS_HOLDABLE	BOOLEAN	is holdable
IS_UPDATABLE	BOOLEAN	is updatable

V$PLAN_HISTORY

The V$PLAN_HISTORY displays information of SQL plans.

Table 9-240 Column information

Column name	Data type	Description
DRIVER_MEMBER_POS	NUMBER	driver member position
DRIVER_SESSION_ID	NUMBER	driver session identifier
SESSION_ID	NUMBER	session identifier
STMT_ID	NUMBER	statement identifier in a session
CL_STMT_ID	NUMBER	cluster statement identifier in a session
DRIVER_CL_STMT_ID	NUMBER	driver cluster statement identifier in a session
PLAN_HISTORY_POS	NUMBER	plan history position
PLAN_HISTORY_ID	NUMBER	plan history identifier
SQL_TEXT	LONG VARCHAR	SQL text for the statement
PLAN_TEXT	LONG VARCHAR	plan text for the statement
LAST_EXEC_TIME	TIMESTAMP(6) WITHOUT TIME ZONE	statement last execution time

V$PLAN_HISTORY_LATEST

The V$PLAN_HISTORY_LATEST displays information of the latest SQL plan.

Table 9-241 Column information

Column name	Data type	Description
DRIVER_MEMBER_POS	NUMBER	driver member position
DRIVER_SESSION_ID	NUMBER	driver session identifier
SESSION_ID	NUMBER	session identifier
STMT_ID	NUMBER	statement identifier in a session
CL_STMT_ID	NUMBER	cluster statement identifier in a session
DRIVER_CL_STMT_ID	NUMBER	driver cluster statement identifier in a session
PLAN_HISTORY_POS	NUMBER	plan history position
PLAN_HISTORY_ID	NUMBER	plan history identifier
SQL_TEXT	LONG VARCHAR	SQL text for the statement
PLAN_TEXT	LONG VARCHAR	plan text for the statement
LAST_EXEC_TIME	TIMESTAMP(6) WITHOUT TIME ZONE	statement last execution time

V$PROCESS_STAT

The V$PROCESS_STAT displays goldilocks process statistics.

Table 9-242 Column information

Column name	Data type	Description
STAT_NAME	VARCHAR(128)	statistic name
PROC_ID	NUMBER	goldilocks process identifier
STAT_VALUE	NUMBER	statistic value

V$PROCESS_MEM_STAT

The V$PROCESS_MEM_STAT displays goldilocks process memory statistics.

Table 9-243 Column information

Column name	Data type	Description
STAT_NAME	VARCHAR(128)	statistic name
PROC_ID	NUMBER	goldilocks process identifier
STAT_VALUE	NUMBER	statistic value

V$PROCESS_SQL_STAT

The V$PROCESS_SQL_STAT displays goldilocks process SQL statistics.

Table 9-244 Column information

Column name	Data type	Description
STAT_NAME	VARCHAR(128)	statistic name
PROC_ID	NUMBER	goldilocks process identifier
STAT_VALUE	NUMBER	statistic value

V$PROPERTY

The V$PROPERTY displays a list of all properties at current session. Otherwise, the instance-wide value.

Table 9-245 Column information

Column name	Data type	Description
PROPERTY_NAME	VARCHAR(128)	name of the property
DESCRIPTION	VARCHAR(2048)	description of the property
DATA_TYPE	VARCHAR(32)	data type of the property
STARTUP_PHASE	VARCHAR(32)	modifiable startup-phase: the value IN (NO MOUNT / MOUNT / OPEN & [BELOW|ABOVE])
VALUE_UNIT	VARCHAR(32)	unit of the property value: the value in (NONE, BYTE, MS(milisec))
PROPERTY_VALUE	VARCHAR(2048)	property value for the session. otherwise, the instance-wide value
PROPERTY_SOURCE	VARCHAR(32)	source of the current property value: the value IN (USER, DEFAULT, ENV_VAR, BINARY_FILE, FILE, SYSTEM)
INIT_VALUE	VARCHAR(2048)	property init value for the session
INIT_SOURCE	VARCHAR(32)	source of the current property INIT_VALUE: the value IN (USER, DEFAULT, ENV_VAR, BINARY_FILE, FILE, SYSTEM)
MIN_VALUE	NUMBER	minimum value for property. null if type is varchar
MAX_VALUE	NUMBER	maximum value for property. null if type is varchar
SES_MODIFIABLE	VARCHAR(32)	property can be changed with ALTER SESSION or not: the value in (TRUE, FALSE)
SYS_MODIFIABLE	VARCHAR(32)	property can be changed with ALTER SYSTEM and when the change takes effect: the value in (NONE, FALSE, IMMEDIATE, DEFERRED)
IS_MODIFIABLE	VARCHAR(32)	property can be changed or not: the value in (TRUE, FALSE)
IS_DEPRECATED	VARCHAR(32)	whether a property is deprecated or not: the value in (TRUE, FALSE)
IS_GLOBAL	VARCHAR(32)	whether a property scope is global or not: the value in (TRUE, FALSE)

V$PROPERTY_ALIAS

The V$PROPERTY_ALIAS displays a list of all properties alias.

Table 9-246 Column information

Column name	Data type	Description
PROPERTY_NAME	VARCHAR(128)	original name of the property
PROPERTY_ALIAS	VARCHAR(128)	alias name of the property

V$PSM_RESERVED_WORDS

The V$PSM_RESERVED_WORDS displays a list of all PSM reserved keywords. Reserved words cannot be used in variable name or procedure name.

Table 9-247 Column information

Column name	Data type	Description
KEYWORD_NAME	VARCHAR(128)	name of keyword
KEYWORD_LENGTH	NUMBER	length of the keyword

V$QUEUE

The V$QUEUE displays information of queue.

Table 9-248 Column information

Column name	Data type	Description
TYPE	NUMBER	queue type (COMMON or DISPATCHER)
INDEX	NUMBER	index
QUEUED	NUMBER	number of items in the queue
WAIT	NUMBER	total time that all items in this queue have waited (1/100 second)
TOTALQ	VARCHAR(128)	total number of items that have ever been in the queue

V$RESERVED_WORDS

The V$RESERVED_WORDS displays a list of all SQL reserved keywords. Reserved words cannot be used in table name or column name.

Table 9-249 Column information

Column name	Data type	Description
KEYWORD_NAME	VARCHAR(128)	name of keyword
KEYWORD_LENGTH	NUMBER	length of the keyword

V$SEQUENCE

The V$SEQUENCE displays information of sequences

Table 9-250 Column information

Column name	Data type	Description
SEQUENCE_NAME	VARCHAR(128)	sequence name
PHYSICAL_ID	NUMBER	sequence physical identifier
START_WITH	NUMBER	start with value
INCREMENT_BY	NUMBER	increment value
MAXVALUE	NUMBER	maximum value
MINVALUE	NUMBER	minimum value
CACHE_SIZE	NUMBER	cache size
LOCAL_NEXT_VALUE	NUMBER	local next value
LOCAL_CURR_VALUE	NUMBER	local current value
RESTART_VALUE	NUMBER	restart value
CYCLE	BOOLEAN	allow cycle
USE_LAST_VALUE	BOOLEAN	use last value or not
LOCAL_CACHE_COUNT	NUMBER	current local cache count
GLOBAL_NEXT_VALUE	NUMBER	global next cache chunk start value
SYNC_COMPARE_SN	NUMBER	serial number for global sequence synchronization
GLOBAL_LATCH_SESSION_ID	NUMBER	identifier of the session acquiring the global latch (-1 if the latch is not acquired)
GLOBAL_LATCH_SESSION_SERIAL	NUMBER	serial number of the session acquiring the global latch (-1 if the latch is not acquired)
DDL_LATCH_SESSION_ID	NUMBER	identifier of the session acquiring the ddl latch (-1 if the latch is not acquired)
DDL_LATCH_SESSION_SERIAL	NUMBER	serial number of the session acquiring the ddl latch (-1 if the latch is not acquired)
LOCAL_LATCH_SESSION_ID	NUMBER	identifier of the session acquiring the local latch (-1 if the latch is not acquired)
LOCAL_LATCH_SESSION_SERIAL	NUMBER	serial number of the session acquiring the local latch (-1 if the latch is not acquired)
IS_ONLINE	BOOLEAN	is online
LAST_SYNC_TIME	TIMESTAMP(6) WITHOUT TIME ZONE	last time the sequence was synchronized

V$SESSION

The V$SESSION displays session information for each current session.

Table 9-251 Column information

Column name	Data type	Description
SESSION_ID	NUMBER	session identifier
SERIAL_NO	NUMBER	session serial number
TRANS_ID	NUMBER	transaction identifier (-1 if inactive transaction)
CONNECTION_TYPE	VARCHAR(32)	connection type: the value in (DA, TCP)
USER_NAME	VARCHAR(128)	user name
SESSION_STATUS	VARCHAR(32)	status of the session: the value in (CONNECTED, SIGNALED, SNIPED, DEAD)
SERVER_TYPE	VARCHAR(32)	server type: the value in (DEDICATED, SHARED)
PROCESS_ID	NUMBER	client process identifier
LOGON_TIME	TIMESTAMP(6) WITH OUT TIME ZONE	logon time
PROGRAM_NAME	VARCHAR(128)	program name
CLIENT_ADDRESS	VARCHAR(1024)	client address (null if DA)
CLIENT_PORT	NUMBER	client port (0 if DA)
FAILOVER_TYPE	VARCHAR(13)	indicates whether and to what extent transparent application failover (TAF) is enabled for the session (NONE, SESSION)
FAILED_OVER	VARCHAR(3)	indicates whether the session is running in failover mode and failover has occurred (YES) or not (NO)
IS_AUDITED	VARCHAR(3)	indicates whether the session is audited (YES) or not (NO)

V$SESSION_AUDIT

The V$SESSION_AUDIT displays audited session information.

Table 9-252 Column information

Column name	Data type	Description
SESSION_ID	NUMBER	session identifier
SERIAL_NO	NUMBER	session serial number
POLICY_NAME	VARCHAR(128)	active audit policy name
WHEN_SUCCESS	VARCHAR(3)	indicates whether the audit policy is enable for auditing successful events or not
WHEN_FAILURE	VARCHAR(3)	indicates whether the audit policy is enable for auditing unsuccessful events or not

V$SESSION_CONNECT_INFO

The V$SESSION_CONNECT_INFO displays information about network connections for the current session.

Table 9-253 Column information

Column name	Data type	Description
SESSION_ID	NUMBER	session identifier
SERIAL_NO	NUMBER	session serial number
CLIENT_CHARSET	VARCHAR(40)	client character set

V$SESSION_EVENT

The V$SESSION_EVENT displays information on waits for an event by a session.

Table 9-254 Column information

Column name	Data type	Description
SESSION_ID	NUMBER	ID of the session
WAIT_EVENT_ID	NUMBER	Identifier of the wait event
WAIT_EVENT_NAME	VARCHAR(64)	Name of the wait event
TOTAL_WAITS	NUMBER	Total number of waits for the event
TOTAL_TIMEOUTS	NUMBER	Total number of timeouts for the event
TIME_WAITED	NUMBER	Total amount of time waited for the event (microsecond)
AVERAGE_WAIT	NUMBER	Average amount of time waited for the event (microsecond)
MAX_WAIT	NUMBER	Maximum time waited for the event by the session (microsecond)
CLASS_NAME	VARCHAR(64)	Name of the class of the wait event

V$SESSION_STAT

The V$SESSION_STAT displays session statistics.

Table 9-255 Column information

Column name	Data type	Description
STAT_NAME	VARCHAR(128)	statistic name
SESS_ID	NUMBER	session identifier
STAT_VALUE	NUMBER	statistic value

V$SESSION_MEM_STAT

The V$SESSION_MEM_STAT displays session memory statistics.

Table 9-256 Column information

Column name	Data type	Description
STAT_NAME	VARCHAR(128)	statistic name
SESS_ID	NUMBER	session identifier
STAT_VALUE	NUMBER	statistic value

V$SESSION_MEM_USAGE

The V$SESSION_MEM_USAGE displays session memory usage for each session.

Table 9-257 Column information

Column name	Data type	Description
SESSION_ID	NUMBER	session identifier
ALLOCATOR_ID	NUMBER	memory allocator identifier
ALLOCATOR_TYPE	VARCHAR(7)	memory allocator type (REGION or DYNAMIC)
MEMORY_TYPE	VARCHAR(4)	memory type (HEAP, SHM)
TOTAL_SIZE	NUMBER	total memory size

V$SESSION_SQL_STAT

The V$SESSION_SQL_STAT displays session SQL statistics.

Table 9-258 Column information

Column name	Data type	Description
STAT_NAME	VARCHAR(128)	statistic name
SESS_ID	NUMBER	session identifier
STAT_VALUE	NUMBER	statistic value

V$SESSION_WAIT

The V$SESSION_WAIT displays the current or last wait for each session.

Table 9-259 Column information

Column name	Data type	Description
SESSION_ID	NUMBER	ID of the session
SEQ_NO	NUMBER	Identifier of the wait event
WAIT_EVENT_ID	NUMBER	Name of the wait event
WAIT_EVENT_NAME	VARCHAR(64)	A number that uniquely identifies the current or last wait (incremented for each wait)
P1TEXT	VARCHAR(64)	Description of the first parameter for the wait event
P1	NUMBER	First wait event parameter (in decimal)
P1HEX	VARCHAR(32)	First wait event parameter (in hex)
P2TEXT	VARCHAR(64)	Description of the second parameter for the wait event
P2	NUMBER	Second wait event parameter (in decimal)
P2HEX	VARCHAR(32)	Second wait event parameter (in hex)
P3TEXT	VARCHAR(64)	Description of the third parameter for the wait event
P3	NUMBER	Third wait event parameter (in decimal)
P3HEX	VARCHAR(32)	Third wait event parameter (in hex)
STATE	VARCHAR(64)	Wait state
WAIT_TIME	NUMBER	If the session is currently waiting, then the value is time waited for the current wait. If the session is not in a wait, then the value is the duration of the last wait (in microseconds)
TIME_SINCE_LAST_WAIT	NUMBER	Time elapsed since the end of the last wait (in microseconds). If the session is currently in a wait, then the value is 0.
CLASS_NAME	VARCHAR(64)	Name of the class of the wait event

V$SHARED_MODE

The V$SHARED_MODE displays information of shared mode.

Table 9-260 Column information

Column name	Data type	Description
NAME	VARCHAR(128)	name
VALUE	VARCHAR(128)	value

V$SHARED_SERVER

The V$SHARED_SERVER displays information of shared servers.

Table 9-261 Column information

Column name	Data type	Description
PROCESS_ID	NUMBER	shared server process identifier
PROCESSED_JOB_COUNT	NUMBER	processed job count
STATUS	VARCHAR(128)	status
IDLE	NUMBER	total idle time (1/100 second)
BUSY	NUMBER	total busy time (1/100 second)

V$SHM_SEGMENT

The V$SHM_SEGMENT displays a list of all shared memory segments.

Table 9-262 Column information

Column name	Data type	Description
SHM_NAME	VARCHAR(32)	shared memory segment name
SHM_ID	NUMBER	shared memory segment identifier
SHM_SIZE	NUMBER	shared memory segment size (in bytes)
SHM_KEY	NUMBER	shared memory segment key
SHM_SEQ	NUMBER	shared memory segment sequence
SHM_ADDR	VARCHAR(32)	start address of the shared memory segment
LARGE_PAGES	BOOLEAN	indicates whether the shared memory segment use large pages

V$SPROPERTY

The V$SPROPERTY displays a list of Properties. This is store a binary property file.

Table 9-263 Column information

Column name	Data type	Description
PROPERTY_NAME	VARCHAR(128)	name of the property
DESCRIPTION	VARCHAR(2048)	description of the property
DATA_TYPE	VARCHAR(32)	data type of the property
STARTUP_PHASE	VARCHAR(32)	modifiable startup-phase: the value IN (NO MOUNT / MOUNT / OPEN & [BELOW|ABOVE])
VALUE_UNIT	VARCHAR(32)	unit of the property value: the value in (NONE, BYTE, MS(milisec))
PROPERTY_VALUE	VARCHAR(2048)	property value stored in the binary property file
PROPERTY_SOURCE	VARCHAR(32)	source of the current property value: the value is BINARY_FILE
INIT_VALUE	VARCHAR(2048)	property init value for the system
INIT_SOURCE	VARCHAR(32)	source of the current property INIT_VALUE: the value IN (USER, DEFAULT, ENV_VAR, BINARY_FILE, FILE, SYSTEM)
MIN_VALUE	NUMBER	minimum value for property. null if type is varchar
MAX_VALUE	NUMBER	maximum value for property. null if type is varchar
SES_MODIFIABLE	VARCHAR(32)	property can be changed with ALTER SESSION or not: the value in (TRUE, FALSE)
SYS_MODIFIABLE	VARCHAR(32)	property can be changed with ALTER SYSTEM and when the change takes effect: the value in (NONE, FALSE, IMMEDIATE, DEFERRED)
IS_MODIFIABLE	VARCHAR(32)	property can be changed or not: the value in (TRUE, FALSE)

V$SQLFN_METADATA

The V$SQLFN_METADATA contains metadata about operators and built-in functions

Table 9-264 Column information

Column name	Data type	Description
FUNC_NAME	VARCHAR(128)	name of the built-in function
MINARGS	NUMBER	minimum number of arguments for the function
MAXARGS	NUMBER	maximum number of arguments for the function
IS_AGGREGATE	BOOLEAN	indicates whether the function is an aggregate function (TRUE) or not (FALSE)

V$SQL_CACHE

The V$SQL_CACHE lists statistics of shared SQL plan.

Table 9-265 Column information

Column name	Data type	Description
SQL_HANDLE	NUMBER	SQL handle
HASH_VALUE	NUMBER	hash value of the SQL statement
REF_COUNT	NUMBER	count of prepared statements referencing the statement
PLAN_SIZE	NUMBER	the total plan size of the SQL statement (in bytes)
CLOCK_ID	NUMBER	clock identifier
PLAN_AGE	NUMBER	plan age
USER_NAME	VARCHAR(128)	user name
BIND_PARAM_COUNT	NUMBER	count of bind parameters
SQL_TEXT	LONG VARCHAR	SQL full text
PLAN_COUNT	NUMBER	physical plan count of the SQL statement
PLAN_ID	NUMBER	plan identifier
PLAN_SIZE	NUMBER	the total plan size of the SQL statement (in bytes)
PLAN_IS_ATOMIC	BOOLEAN	plan is atomic array insert or not
PLAN_TEXT	LONG VARCHAR	plan text for SQL statement

V$SQL_COMMAND

The V$SQL_COMMAND lists attribute information of each SQL command.

Table 9-266 Column information

Column name	Data type	Description
COMMAND	VARCHAR(128)	SQL command
FROM_PHASE	VARCHAR(32)	executable from start-up phase
UNTIL_PHASE	VARCHAR(32)	executable until start-up phase
ACCESS_MODE	VARCHAR(32)	database access mode: values in (NONE, READ & WRITE, READ, READ & LOCK)
NEED_FETCH	VARCHAR(32)	the command is a query which has result set and need fetch
IS_DDL	VARCHAR(3)	the command is a DDL(Data Defintion Language) or not
CLUSTER_LOCK_MODE	VARCHAR(32)	cluster lock mode: values in (NONE, SERIAL, MANUAL)
AUTO_COMMIT	VARCHAR(3)	the command is auto-commit or not
IS_CACHEABLE	VARCHAR(3)	the command is plan-cacheable or not
AUDIT_ACTION	VARCHAR(128)	auditiable action name for the SQL command

V$SQL_HISTORY

The V$SQL_HISTORY displays information of SQLs.

Table 9-267 Column information

Column name	Data type	Description
DRIVER_MEMBER_POS	NUMBER	driver member position
SESSION_ID	NUMBER	session identifier
START_TIME	TIMESTAMP(6) WITHOUT TIME ZONE	statement start time
EXEC_TIME	NUMBER	execution time(us)
PREPARED	BOOLEAN	indicates whether the statement is prepared (YES) or not (NO)
SUCCESS	BOOLEAN	indicates whether the statement is success (YES) or not (NO)
STATUS	CHARACTER VARYING(16)	status of the statement: the value in (RUNNING, DONE)
SQL_TEXT	CHARACTER VARYING(1024)	first 1024 bytes of the SQL text for the statement

V$STATEMENT

The V$STATEMENT lists all statements.

Table 9-268 Column information

Column name	Data type	Description
SESSION_ID	NUMBER	session identifier
STMT_ID	NUMBER	statement identifier in a session
STMT_VIEW_SCN	NUMBER	statement view scn
SQL_TEXT	VARCHAR(1024)	first 1024 bytes of the SQL text for the statement
START_TIME	TIMESTAMP(6) WITHOUT TIME ZONE	statement start time
TOTAL_EXEC_TIME	NATIVE_BIGINT	total execution time(us)
LAST_EXEC_TIME	NATIVE_BIGINT	last execution time(us)
EXECUTIONS	NATIVE_BIGINT	number of executions

V$SYSTEM_EVENT

The V$SYSTEM_EVENT displays information on total waits for an event.

Table 9-269 Column information

Column name	Data type	Description
WAIT_EVENT_ID	NUMBER	Identifier of the wait event
WAIT_EVENT_NAME	VARCHAR(64)	Name of the wait event
TOTAL_WAITS	NUMBER	Total number of waits for the event
TOTAL_TIMEOUTS	NUMBER	Total number of timeouts for the event
TIME_WAITED	NUMBER	Total amount of time waited for the event (microsecond)
AVERAGE_WAIT	NUMBER	Average amount of time waited for the event (microsecond)
CLASS_NAME	VARCHAR(64)	Name of the class of the wait event

V$SYSTEM_STAT

The V$SYSTEM_STAT displays system statistics.

Table 9-270 Column information

Column name	Data type	Description
STAT_NAME	VARCHAR(128)	statistic name
STAT_VALUE	NUMBER	statistic value
COMMENTS	VARCHAR(1024)	comments

V$SYSTEM_MEM_STAT

The V$SYSTEM_MEM_STAT displays system memory statistics.

Table 9-271 Column information

Column name	Data type	Description
STAT_NAME	VARCHAR(128)	statistic name
STAT_VALUE	NUMBER	statistic value
COMMENTS	VARCHAR(1024)	comments

V$SYSTEM_SQL_STAT

The V$SYSTEM_SQL_STAT displays system SQL statistics.

Table 9-272 Column information

Column name	Data type	Description
STAT_NAME	VARCHAR(128)	statistic name
STAT_VALUE	NUMBER	statistic value
COMMENTS	VARCHAR(1024)	comments

V$TABLES

The V$TABLES contains the definitions of all the performance views (views beginning with V$).

Table 9-273 Column information

Column name	Data type	Description
TABLE_OWNER	VARCHAR(128)	owner name who owns the performance view
TABLE_SCHEMA	VARCHAR(128)	schema name of the performance view
TABLE_NAME	VARCHAR(128)	name of the performance view
STARTUP_PHASE	VARCHAR(32)	visible startup phase of the performance view
CREATED_TIME	TIMESTAMP(6) WITHOUT TIME ZONE	• available only in OPEN phase created time of the performance view
MODIFIED_TIME	TIMESTAMP(6) WITHOUT TIME ZONE	• available only in OPEN phase modified time of the performance view
COMMENTS	VARCHAR(1024)	• available only in OPEN phase comments of the performance view

V$TABLESPACE

This view displays tablespace information.

Table 9-274 Column information

Column name	Data type	Description
TBS_NAME	VARCHAR(128)	tablespace name
TBS_ID	NUMBER	tablespace identifier
TBS_ATTR	VARCHAR(128)	tablespace attribute: the value in (device attribute (MEMORY) | temporary attribute (TEMPORARY, PERSISTENT) | usage attribute(DICT, UNDO, DATA, TEMPORARY))
IS_LOGGING	BOOLEAN	indicates whether the tablespace is a logging tablespace (YES) or not (NO)
IS_ONLINE	BOOLEAN	indicates whether the tablespace is ONLINE (YES) or OFFLINE (NO)
OFFLINE_STATE	VARCHAR(32)	indicates whether the tablespace can be taken online normally (CONSISTENT) or not (INCONSISTENT). null if the tablespace is ONLINE
EXTENT_SIZE	NUMBER	extent size of the tablespace (in bytes)
PAGE_SIZE	NUMBER	page size of the tablespace (in bytes)

V$TABLESPACE_STAT

This view displays tablespace statistical information.

Table 9-275 Column information

Column name	Data type	Description
TBS_NAME	VARCHAR(128)	tablespace name
TBS_ID	NUMBER	tablespace identifier
TOTAL_EXT_COUNT	NUMBER	total extent count of the tablespace
USED_META_EXT_COUNT	NUMBER	meta extent count currently used on the tablespace
USED_DATA_EXT_COUNT	NUMBER	data extent count currently used on the tablespace
FREE_EXT_COUNT	NUMBER	free extent count of the tablespace
EXTENT_SIZE	NUMBER	extent size of the tablespace (in bytes)

V$TRANSACTION

The V$TRANSACTION lists the active transactions in the system.

Table 9-276 Column information

Column name	Data type	Description
TRANS_ID	NUMBER	transaction identifier
SESSION_ID	NUMBER	session identifier (null if the global transaction is unassociated
TRANS_SLOT_ID	NUMBER	transaction slot identifier
PHYSICAL_TRANS_ID	NUMBER	physical transaction identifier
TRANS_STATE	VARCHAR(32)	transaction state: the value in (ACTIVE, BLOCK, PREPARE, COMMIT, ROLLBACK, IDLE, PRECOMMIT)
IS_GLOBAL	BOOLEAN	indicates whether the transaction is global or not
TRANS_ATTRIBUTE	VARCHAR(32)	transaction attribute: the value in (READ_ONLY, UPDATABLE, LOCKABLE, UPDATABLE | LOCKABLE)
ISOLATION_LEVEL	VARCHAR(32)	transaction isolation level: the value in (READ COMMITTED, SERIALIZABLE)
TRANS_VIEW_SCN	NUMBER	transaction view scn
TCN	NUMBER	transaction change number
TRANS_SEQ	NUMBER	transaction sequence number
START_TIME	TIMESTAMP(6) WITHOUT TIME ZONE	transaction start time
UNDO_SEGMENT_ID	NUMBER	undo segment identifier

V$WAIT_EVENT_CLASS_NAME

The V$WAIT_EVENT_CLASS_NAME displays information about Class of wait event.

Table 9-277 Column information

Column name	Data type	Description
CLASS_ID	NUMBER	Identifier of the class of the wait event
NAME	VARCHAR(64)	Name of the class of the wait event
DESCRIPTION	VARCHAR(128)	Description of the class of the wait event

V$WAIT_EVENT_NAME

The V$WAIT_EVENT_NAME displays information about wait events.

Table 9-278 Column information

Column name	Data type	Description
CLASS_ID	NUMBER	Identifier of the wait event
NAME	VARCHAR(64)	Name of the wait event
DESCRIPTION	VARCHAR(128)	Description of the wait event
PARAMETER1	NUMBER	Description of the first parameter for the wait event
PARAMETER1	NUMBER	Description of the second parameter for the wait event
PARAMETER1	NUMBER	Description of the third parameter for the wait event
CLASS_ID	NUMBER	Identifier of the class of the wait event
CLASS_NAME	VARCHAR(64)	Name of the class of the wait event

V$XA_TRANSACTION

The V$XA_TRANSACTION displays information on the currently active XA transactions.

Table 9-279 Column information

Column name	Data type	Description
XA_TRANS_ID	VARCHAR(1024)	XA transaction identifier
LOCAL_TRANS_ID	NUMBER	local transaction identifier
DRIVER_TRANS_ID	NUMBER	driver transaction identifier
DRIVER_MEMBER_POS	NUMBER	driver member position
XA_TRANS_STATE	VARCHAR(32)	state of the XA transaction: the value in (NOTR, ACTIVE, IDLE, PREPARED, ROLLBACK_ONLY, HEURISTIC_COMPLETED)
ASSO_STATE	VARCHAR(32)	associate state of the XA transaction: the value in (NOT_ASSOCIATED, ASSOCIATED, ASSOCIATION_SUSPENDED)
START_TIME	TIMESTAMP(6) WITHOUT TIME ZONE	XA transaction start time
IS_REPREPARABLE	BOOLEAN	indicates whether the XA transaction is repreparable

10.

Server Property

10.1 Server Property Information

For more information about SQL syntax to change properties, refer to the followings.

- **ALTER SYSTEM SET property_name**
- **ALTER SESSION SET property_name**

For more information about property types, refer to the followings.

- **V$PROPERTY**: It displays the property list which can be altered while the system is operating or is rest arting.
- **V$SPROPERTY**: It is either the property which was set by reading the binary file, or the property list w hich is stored in the binary file.
- **V$DB_PROPERTY**: It is a read-only property list which can be altered only when creating the database, and it can not be altered afterwards.

The followings describe basic information items of property in this manual.

Item	Description
Name	Property name
Summary	Short description of the property
Data type	Data type of the property value
Applicable phase	A startup phase which can be updated with ALTER SYSTEM or ALTER SESSION • NONE: Applicable phase does not exist. (If it can be updated, but applicable pha se is NONE, then use *SCOPE = FILE* option.)
Updatable	Whether property is updatable or not • If the property value is TRUE, it is updatable. • If the property value is FALSE, only the read-only is possible.
ALTER SESSION	Whether property is updatable or not by using **ALTER SESSION SET property_name**
ALTER SYSTEM	Whether property is updatable or not by using **ALTER SYSTEM SET property_name** • IMMEDIATE: The updated value is immediately reflected in all session after exec ution. • DEFERRED: The updated value is reflected only in the session which is connected after execution. However, it is not reflected in already connected session. • FALSE: The updated value is not reflected in the session during execution. Howe ver, the updated value is reflected after restart, (Properties are updatable by usin g only *SCOPE=FILE* option.) • NONE: It is not updatable.
MIN	If the data type is BIGINT, it is the minimum value of property. If the data type is VARCHAR, the minimum value of property is N/A.
MAX	If the data type is BIGINT, it is the maximum value of property. If the data type is VARCHAR, the maximum value of property is N/A.

Item	Description
Default value	Default value of the property

10.2 Property Alias Information

The information about the property alias is viewed through **V$PROPERTY_ALIAS**.

The basic information about the property alias written in this manual is as follows.

Item	Description
Original name	It is the original name of the property.
ALIAS	It is the name of the property alias.

For more information about property alias list, refer to **Property Alias**.

CDISPATCHER_THREADS

It is an alias of **CDISPATCHER_LOCKABLE_THREADS**.

CLUSTER_COMMIT_SLAVES

It is an alias of **CLUSTER_COMMIT_SLAVE_CSERVERS**.

CLUSTER_SERVER_RESPONSE_ QUEUE_SIZE

It is an alias of **CLUSTER_GSERVER_RESPONSE_QUEUE_SIZE**.

CSERVER

It is an alias of **CLUSTER_LOCKABLE_CSERVERS**.

INCREMENTAL_CHECKPOINT_CRITERIA

It is an alias of **BUFFER_DIRTY_PAGE_LIMIT**.

LOCKLESS_CSERVERS

It is an alias of **CLUSTER_LOCKLESS_CSERVERS**.

MEMORY_MERGE_RUN_COUNT

It is an alias of **INDEX_MERGE_RUN_COUNT**.

MEMORY_SORT_RUN_SIZE

It is an alias of **INDEX_SORT_RUN_SIZE**.

SYSTEM_LOGGER_DIR

It is an alias of **TRACE_SYSTEM_DIR**.

10.3 ADMIN_SESSION_POOL_INIT_SIZE

Basic Information

Item	Description
Name	ADMIN_SESSION_POOL_INIT_SIZE
Summary	initial memory size for admin session pool
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	0
MAX	1099511627776 (1T)
Default value	10485760 (10M)

Description

It sets the initial memory size of the admin session pool.

10.4 ADMIN_SESSION_POOL_NEXT_SIZE

Basic Information

Item	Description
Name	ADMIN_SESSION_POOL_NEXT_SIZE
Summary	memory size to be expanded in admin session pool
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	131072 (128K)
MAX	1073741824 (1G)
Default value	1048576 (1M)

Description

It sets how much to extend the memory size in the session pool when expanding the admin session pool space.
It is valid only when ADMIN_SESSION_POOL_INIT_SIZE is bigger than 0.

10.5 AGING_INTERVAL

Basic Information

Item	Description
Name	AGING_INTERVAL
Summary	aging interval time(ms)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	1
MAX	100000000
Default value	10

Description

It sets the idle time (second) when an ager thread which deletes the previous version data does not have a job to process in MVCC based database.

10.6 AGING_PLAN_INTERVAL

Basic Information

Item	Description
Name	AGING_PLAN_INTERVAL
Summary	aging plan interval time(s)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	31536000
Default value	0

Description

The SQL plan which is older than AGING_PLAN_INTERVAL becomes the aging target.

10.7 ARCHIVELOG_DIR_1 ~ ARCHIVELOG_DIR_10

Basic Information

Item	Description
Name	ARCHIVELOG_DIR_1 ~ ARCHIVELOG_DIR_10
Summary	archive log directory
Data type	VARCHAR
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE(ARCHIVELOG_DIR_1), TRUE(ARCHIVELOG_DIR_2 ~ ARCHIVELOG_DIR_10)
ALTER SYSTEM	DEFERRED
MIN	N/A
MAX	N/A
Default value	$GOLDILOCKS_DATA/archive_log

Description

It specifies archiving directory of GOLDILOCKS database's online redo log file. Also, it specifies where to r ead of archive redo log file at media recovery. The online redo log file creates archive redo log file only in ARCHIVELOG_DIR_1.

ARCHIVELOG_DIR_1 sets only the system, but ARCHIVELOG_DIR_2 ~ ARCHIVELOG_DIR_10 sets the sessi on.

10.8 ARCHIVELOG_FILE

Basic Information

Item	Description
Name	ARCHIVELOG_FILE
Summary	default archive log file
Data type	VARCHAR
Applicable phase	MOUNT or below
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	DEFERRED
MIN	N/A
MAX	N/A
Default value	archive

Description

It sets the prefix of the targeted file name stored in archive directory when archiving online redo logfile. The archive logfile's name consists of prefix '_' set in ARCHIVELOG_FILE, the file sequence and the file extension 'log'. For example, the online logfile whose file sequence is 0 is archived as 'archive_0.log'.

10.9 ARCHIVELOG_MODE

Basic Information

Item	Description
Name	ARCHIVELOG_MODE
Summary	archive log mode(0:disable, 1:enable)
Data type	BIGINT
Applicable phase	NONE
Updatable	FALSE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	0
MAX	1
Default value	0

Description

The property is applied at database creation. The archivelog mode can be set to one of the following valu e.

- 0: NOARCHIVELOG
- 1: ARCHIVELOG

It does not affect archive log mode during operation after database is created. The archive log mode can be modified by using *ALTER DATABASE {ARCHIVELOG | NOARCHIVELOG}* in MOUNT phase.

10.10 BACKUP_DIR_1 ~ BACKUP_DIR_10

Basic Information

Item	Description
Name	BACKUP_DIR_1 ~ BACKUP_DIR_10
Summary	backup directory
Data type	VARCHAR
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE(BACKUP_DIR_1), TRUE(BACKUP_DIR_2 ~ BACKUP_DIR_10)
ALTER SYSTEM	DEFERRED
MIN	N/A
MAX	N/A
Default value	$GOLDILOCKS_DATA/backup

Description

A backup file is created when incremental backup is executed. Then it sets a directory of backup file to be read when restoring files using incremental backup. Incremental backups are created only in the directory set in BACKUP_DIR_1.

BACKUP_DIR_1 sets only the system, but BACKUP_DIR_2 ~ BACKUP_DIR_10 sets the session.

10.11 BLOCK_READ_COUNT

Basic Information

Item	Description
Name	BLOCK_READ_COUNT
Summary	value count for a block read
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	DEFERRED
MIN	1
MAX	65536
Default value	20

Description

The SQL executes operation by reading row in the unit of BLOCK_READ_COUNT which is a row bundle. BLOCK_READ_COUNT means the number of rows to be processed at a time when operation is executed. It is a basic unit of pipe-lining process of execution nodes whiich are used in SQL query processing.

If BLOCK_READ_COUNT value is big the processing performance improves, but many memory resources are used.
The value between 10 and 100 is recommended.
If the value becomes bigger than 100 the resource usage increases proportionately, but, the performance improvement does not increase proportionately.

10.12 BROADCAST_INDEX_REBUILD_PROTOCOL

Basic Information

Item	Description
Name	BROADCAST_INDEX_REBUILD_PROTOCOL
Summary	broadcast index rebuild protocol
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	NO

Description

It sets whether to simultaneously rebuild the indexes on multiple members when rebuilding the index in c luster environment.

10.13 BROADCAST_REBALANCE_PROTOCOL

Basic Information

Item	Description
Name	BROADCAST_REBALANCE_PROTOCOL
Summary	broadcast rebalance protocol
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	NO

Description

It sets whether to simultaneously process protocol which can be processed on multiple members at the sa me time when performing table rebalancing in a cluster environment.

10.14 BUFFER_CACHE_SIZE

Basic Information

Item	Description
Name	BUFFER_CACHE_SIZE
Summary	buffer cache size (byte)
Data type	BIGINT
Applicable phase	NONE
Updatable	FALSE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	1 MB
MAX	1 TB
Default value	64 MB

Description

It sets the size of the buffer which cashes the page in the disk tablespace.

10.15 BUFFER_CHECKPOINT_LIST_COUNT

Basic Information

Item	Description
Name	BUFFER_CHECKPOINT_LIST_COUNT
Summary	number of buffer checkpoint lists
Data type	BIGINT
Applicable phase	NONE
Updatable	FALSE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	1
MAX	36
Default value	1

Description

It is linked to the checklist when pages of disk tablespace cached to the buffer are updated. Each checkp oint list flushes updated pages linked to the checkpoint list by its own flush thread to the disk, and BUFFE R_CHECKPOINT_LIST_COUNT sets the number of checkpoint lists and the number of flush threads.

10.16 BUFFER_DIRTY_PAGE_LIMIT

Basic Information

Item	Description
Name	BUFFER_DIRTY_PAGE_LIMIT
Summary	a limit on the number of dirty pages in the buffer cache
Data type	BIGINT
Applicable phase	NO_MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	134217728
Default value	0

Description

Pages updated in the system buffer cache is applied to the disk at checkpoint, and if the buffer cache size is big and updated pages are a lot, then the checkpoint takes a long time and it affects the service. GOLDILOCKS performs the incremental checkpoint which applies updated pages to the disk when the number of updated pages are over a specified number in the system. And BUFFER_DIRTY_PAGE_LIMIT sets the criteria to perform the incremental checkpoint.

For example, when this value is set to 1000, if the updated pages are less than 1000 in the system, then the updated pages are not applied to the disk. However, if the updated pages are 1000 or above, then the updated pages in the buffer are applied to the disk.

The default value is 0, and it means infinity. In this case, it does not perform the incremental checkpoint even when all cached pages in the buffer are updated.

Set the appropriate value for BUFFER_DIRTY_PAGE_LIMIT and **INCREMENTAL_DATAFILE_HEADER_UPDATE_CRITERIA** considering the restart recovery time and its effect on the service.

ALIAS

Item	Description
Original name	BUFFER_DIRTY_PAGE_LIMIT
ALIAS	INCREMENTAL_CHECKPOINT_CRITERIA

10.17 BUFFER_FLUSH_THREADS

Basic Information

Item	Description
Name	BUFFER_FLUSH_THREADS
Summary	number of buffer flush threads
Data type	BIGINT
Applicable phase	NONE
Updatable	FALSE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	1
MAX	36
Default value	1

Description

It is linked to the flush list then requests flush to the buffer flusher, to reuse bch which cached the update d pages in the buffer lru list. In this case, BUFFER_FLUSH_THREADS sets the number of buffer flushers an d flush lists to be used in the database.

10.18 BUFFER_FLUSHING_INTERVAL

Basic Information

Item	Description
Name	BUFFER_FLUSHING_INTERVAL
Summary	buffer flushing interval time (sec)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	1
MAX	86400 (1 day)
Default value	3

Description

It sets the idle time (sec) when the job to be processed by the buffer flusher flushing updated disk tablesp ace pages to the disk does not exist.

10.19 BUFFER_FREE_LIST_COUNT

Basic Information

Item	Description
Name	BUFFER_FREE_LIST_COUNT
Summary	number of buffer free lists
Data type	BIGINT
Applicable phase	NONE
Updatable	FALSE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	1
MAX	64
Default value	16

Description

It sets the number of buffer free lists connecting bch which are instantly available to use in the buffer cac he.

10.20 BUFFER_HASH_BUCKETS

Basic Information

Item	Description
Name	BUFFER_HASH_BUCKETS
Summary	number of buffer hash buckets
Data type	BIGINT
Applicable phase	NONE
Updatable	FALSE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	0
MAX	1073741824
Default value	0

Description

It sets the number of hash buckets for the disk tablespace pages cached in the buffer. It can be set from 0 to 1073741824, and 0 is set by calculating hash buckets as many as pages which can be cached to the b uffer which is set according to BUFFER_CACHE_SIZE. If the buffer size is smaller than the specified value, then it adjusts the number of hash buckets to the buffer size.

10.21 BUFFER_HOT_REGION_CRITERIA

Basic Information

Item	Description
Name	BUFFER_HOT_REGION_CRITERIA
Summary	threshold touch count of hot region in the buffer lru list
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	2
MAX	100
Default value	2

Description

It sets touch count to transfer pages existing in the cold region to the hot region in buffer lru list.

10.22 BUFFER_HOT_REGION_PERCENT

Basic Information

Item	Description
Name	BUFFER_HOT_REGION_PERCENT
Summary	the percentage of hot region in the buffer lru list
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	80
Default value	20

Description

It sets the proportion (percentage) of hot region pages to the entire page in the buffer lru list.

10.23 BUFFER_LRU_LIST_COUNT

Basic Information

Item	Description
Name	BUFFER_LRU_LIST_COUNT
Summary	number of buffer LRU lists
Data type	BIGINT
Applicable phase	NONE
Updatable	FALSE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	1
MAX	64
Default value	16

Description

It sets the number of lru lists to select a victim among pages in use by caching when the free buffer for ca ching disk tablespace pages does not exist.

10.24 BUFFER_LRU_SCAN_PERCENT

Basic Information

Item	Description
Name	BUFFER_LRU_SCAN_PERCENT
Summary	the percentage of buffers to inspect when looking for free
Data type	NO MOUNT or above
Applicable phase	TRUE
Updatable	FALSE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	10
MAX	100
Default value	40

Description

If the free buffer available right now does not exist in the system, then it searches for the reusable buffer from the lru list to cache the disk tablespace page to the buffer. BUFFER_LRU_SCAN_PERCENT sets the p ercent of the buffer pages set in BUFFER_CACHE_SIZE, so that it can determine the number of pages to c heck to find the reusable buffer from lru list.

10.25 BUFFER_MULTIPAGE_READ_COUNT

Basic Information

Item	Description
Name	BUFFER_MULTIPAGE_READ_COUNT
Summary	maximum number of pages read in one I/O operation during a full scan
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	DEFERRED
MIN	1
MAX	128
Default value	32

Description

It sets the maximum number of pages to be used for one time disk IO when full scanning the disk table.

10.26 BUFFER_PREFETCH_PAGE_COUNT

Basic Information

Item	Description
Name	BUFFER_PREFETCH_PAGE_COUNT
Summary	the maximum number of pages to be prefetched to the buffer per I/O operation
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	1
MAX	128
Default value	32

Description

It sets the maximum number of pages nearby to be prefetched per a disk I/O when accessing to the page in the disk tablespace which does not exist in the buffer.

10.27 BULK_IO_PAGE_COUNT

Basic Information

Item	Description
Name	BULK_IO_PAGE_COUNT
Summary	page count for bulk IO operation
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	128
MAX	131072
Default value	3840

Description

It is used when IO READ of the data file occurs during server restart, or when IO WRITE occurs during creating a data file.

The heap memory is allocated as big as BULK_IO_PAGE_COUNT * 8192 when server restarts or data file is created. If the session's PRIVATE_STATIC_AREA_SIZE is smaller than the heap memory size, an error of insufficient memory may occur. In this case, extend PRIVATE_STATIC_AREA_SIZE.

10.28 CDISPATCHER_HOT_POLICY_INTERVAL

Basic Information

Item	Description
Name	CDISPATCHER_HOT_POLICY_INTERVAL
Summary	cdispatcher dequeue interval for busy waiting (micro second)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	86400000000 (1day)
Default value	0

Description

It is the time of the busy waiting when performing the dequeue in the cdispatcher. It is a micro second un it. If this value is big, it uses more cpu but the user response time (latency) is decreased.
The default value is 0, and the busy waiting is not allowed.

10.29 CDISPATCHER_LOCKABLE_THREADS

Basic Information

Item	Description
Name	CDISPATCHER_LOCKABLE_THREADS
Summary	cdispatcher lockable sender, receiver thread count
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	1
MAX	30
Default value	1

Description

It sets the number of cdispatcher threads of lockble data sender and receiver. However, the number of cd ispatcher threads of lockless data sender and receiver is set by using **CDISPATCHER_LOCKLESS_THREADS**.

ALIAS

Item	Description
Original name	CDISPATCHER_LOCKABLE_THREADS
ALIAS	CDISPATCHER_THREADS

10.30 CDISPATCHER_LOCKLESS_THREADS

Basic Information

Item	Description
Name	CDISPATCHER_LOCKLESS_THREADS
Summary	cdispatcher lockless sender, receiver thread count
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	1
MAX	30
Default value	1

Description

It sets the number of cdispatcher threads of lockless data sender and receiver. However, the number of c dispatcher threads of lockable data sender and receiver is set by using **CDISPATCHER_LOCKABLE_THREADS**.

10.31 CDISPATCHER_MAX_PACKET_BUFFER_SIZE

Basic Information

Item	Description
Name	CDISPATCHER_MAX_PACKET_BUFFER_SIZE
Summary	maximum packet buffer size for cdipatcher sender and receiver threads
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	10 Mega
MAX	32 Giga
Default value	1 Giga

Description

It sets the maximum size of the buffer where the data sender and receiver of cdispatcher stores the sent/received packets.

10.32 CDISPATCHER_SOCKET_BUFFER_SIZE

Basic Information

Item	Description
Name	CDISPATCHER_SOCKET_BUFFER_SIZE
Summary	cdispatcher socket buffer(sender, receiver) size
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	64
MAX	100 Mega
Default value	32768

Description

It is the socket buffer(sender, receiver) size of cdispatcher.

10.33 CDISPATCHER_SYNC_THREADS

Basic Information

Item	Description
Name	CDISPATCHER_SYNC_THREADS
Summary	cdispatcher sync thread count
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	1
MAX	30
Default value	1

Description

It is the thread count of cdispatcher sync.

10.34 CHANGE_TRACKING

Basic Information

Item	Description
Name	CHANGE_TRACKING
Summary	enable change tracking for incremental backup
Data type	BOOLEAN
Applicable phase	NONE
Updatable	FALSE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	0
MAX	1
Default value	NO

Description

It sets whether to track the updated pages to perform the incremental backup of disk tablespace.

- NO: disable change tracking
- YES: enable change tracking

change tracking can be enabled by using ALTER DATABASE { ENABLE | DISABLE } CHANGE TRACKING on mount or above phase only when the database is operated in archivelog.

10.35 CHANGE_TRACKING_EXTENT_SIZE

Basic Information

Item	Description
Name	CHANGE_TRACKING_EXTENT_SIZE
Summary	number of pages to track changed of incremental backup
Data type	BIGINT
Applicable phase	NONE
Updatable	FALSE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	16
MAX	128
Default value	32

Description

It sets the number of pages to display with one dirty flag when change tracking. For example, if it is set to 32, then one dirty flag is used per 32 pages, and if it is set to 128, then then one dirty flag is used per 12 8 pages.

10.36 CHANGE_TRACKING_FILE

Basic Information

Item	Description
Name	CHANGE_TRACKING_FILE
Summary	default change tracking file
Data type	VARCHAR
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	DEFERRED
MIN	N/A
MAX	N/A
Default value	$GOLDILOCKS_DATA/backup/gl_change_tracking_file.ctf

Description

It sets the file directory which stores the change tracking, and the file name.

10.37 CHAR_LENGTH_UNITS

Basic Information

Item	Description
Name	CHAR_LENGTH_UNITS
Summary	char length units
Data type	VARCHAR
Applicable phase	NONE
Updatable	FALSE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	N/A
MAX	N/A
Default value	OCTETS

Description

It is the value of char length units used when defining character string such as CHAR, VARCHAR and omit ting char length unit as follows.

```
CREATE TABLE t1
(
   id   CHAR( 10 OCTETS ),
   name VARCHAR( 128 CHARACTERS ),
   addr VARCHAR( 128 )
);
```

- id CHAR(10 OCTETS) means 10 bytes.
- name VARCHAR(128 CHARACTERS) means 128 characters.
- addr VARCHAR(128) means that this value is referenced when char length unit is omitted.

When database is created, the property is set to either OCTETS or CHARACTERS. OCTETS is the number o f bytes, and CHARACTERS is the number of characters.

The SQL standard defines CHARACTERS as default value. Other DBMS defines the default value of char length unit as follows.

- Oracle and DB2 define OCTETS as default value.
- MS-SQL, MySQL, PostgreSQL define CHARACTERS as default value.

10.38 CHARACTER_SET

Basic Information

Item	Description
Name	CHARACTER_SET
Summary	character set
Data type	VARCHAR
Applicable phase	NONE
Updatable	FALSE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	N/A
MAX	N/A
Default value	UTF8

Description

It is a character set of database, and it is applied when database is created.
The property is set to one of the following values.

Character set	Description
SQL_ASCII	ASCII standards
UTF8	Unicode, 8-bit
UHC	Unified Hangul code
GB18030	Chinese government standards

10.39 CHECK_DEDICATE_CONNECTION_INTERVAL

Basic Information

Item	Description
Name	CHECK_DEDICATE_CONNECTION_INTERVAL
Summary	check dedicate socket
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	100000
Default value	1000

Description

It is the interval of checking for when the client forcibly cut the connection in C/S dedicate environment. The dedicate server(gserver) checks the socket, and it terminates it if it was cut. The default value is 1,000 millisecond (1 second).

10.40 CHECKPOINT_LIST_COUNT_PER_IO_GROUP

Basic Information

Item	Description
Name	CHECKPOINT_LIST_COUNT_PER_IO_GROUP
Summary	number of checkpoint lists per each io group
Data type	BIGINT
Applicable phase	NO_MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	DEFERRED
MIN	0
MAX	8192
Default value	0

Description

It sets the number of checkpoint lists to be processed by each IO slave set by **PARALLEL_IO_FACTOR**. It specifies the appropriate value to efficiently process the concurrency when connecting the updated pages to the checkpoint lists. The default value is 0, and in this case, each IO slave creates the checkpoint lists as many as the number of CPU and processes it.

10.41 CLIENT_MAX_COUNT

Basic Information

Item	Description
Name	CLIENT_MAX_COUNT
Summary	maximum session count
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	12
MAX	65535
Default value	128

Description

It sets the maximum number of sessions to connect.

10.42 CLIENT_NUMA_POLICY

Basic Information

Item	Description
Name	CLIENT_NUMA_POLICY
Summary	client numa policy(0: by modualar, 1: by statistics , 2: by manunal)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	DEFERRED
MIN	0
MAX	2
Default value	0

Description

It determines the policy to distribute client processes to NUMA nodes. This property is operated when NU MA property is set to on.

- 0: It determines the NUMA node to be connected by modularizing the session ID.
- 1: It connects to the NUMA node of which is the least connected based on the statistics information.
- 2: C/S clientis determined by TCP_CLIENT_NUMA_NODE property, D/A client is determined by DA_CL IENT_ NUMA_NODE property.

10.43 CLOSE_PSM_CHILD_STMTS

Basic Information

Item	Description
Name	CLOSE_PSM_CHILD_STMTS
Summary	close child statements of PSM at the end of each execution
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	DEFERRED
MIN	0
MAX	1
Default value	NO

Description

It closes the child statement of PSM at the end of each execution.

10.44 CLUSTER_ASYNC_COMMIT

Basic Information

Item	Description
Name	CLUSTER_ASYNC_COMMIT
Summary	enable asynchronous commit in cluster system
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	YES

Description

It determines whether to internally process the commit protocol on an async mode in cluster system.

If this property is set to on, it asynchronously commits each node, so the temporary inconsistency among nodes may occur. On the other hand, if it is set to off, it synchronizes everytime it commits, so it may reduce the performance. Therefore, it is required to determine the appropriate property depending on the purpose.

10.45 CLUSTER_CM_BUFFER_SIZE

Basic Information

Item	Description
Name	CLUSTER_CM_BUFFER_SIZE
Summary	communication buffer size for cluster
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	10 Mega
MAX	32 Giga
Default value	10 Mega

Description

It is the communication buffer size for cluster.

10.46 CLUSTER_CM_READ_BUFFER_SIZE

Basic Information

Item	Description
Name	CLUSTER_CM_READ_BUFFER_SIZE
Summary	communication read block size
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	8192
MAX	10485,760
Default value	65536

Description

It is the communication read block size.

10.47 CLUSTER_COMMIT_SLAVE_CSERVERS

Basic Information

Item	Description
Name	CLUSTER_COMMIT_SLAVE_CSERVERS
Summary	number of commit slave cservers
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	0
MAX	8
Default value	0

Description

It is the number of commit slaves.

ALIAS

Item	Description
Original name	CLUSTER_COMMIT_SLAVE_CSERVERS
ALIAS	CLUSTER_COMMIT_SLAVES

10.48 CLUSTER_COMMIT_STREAM_ISOLATION

Basic Information

Item	Description
Name	CLUSTER_COMMIT_STREAM_ISOLATION
Summary	isolate cluster dispatcher stream for commit protocol
Data type	BOOLEAN
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	0
MAX	1
Default value	0

Description

It determines whether to internally perform the commit process flow in the cluster system separately from other protocol process. The performance may be improved when seperating the commit process accordin g to the system environment.

10.49 CLUSTER_CONNECTION

Basic Information

Item	Description
Name	CLUSTER_CONNECTION
Summary	connection mode for cluster (socket:0, rdma:1)
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	0
MAX	1
Default value	0: socket

Description

It is the connection mode for cluster. (socket:0, rdma:1)

10.50 CLUSTER_CONNECTION_TIMEOUT_SEC

Basic Information

Item	Description
Name	CLUSTER_CONNECTION_TIMEOUT_SEC
Summary	connection timeout for cluster
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	86400
Default value	10

Description

It is the connection timeout for cluster.

10.51 CLUSTER_DATA_SYNC_SERVERS

Basic Information

Item	Description
Name	CLUSTER_DATA_SYNC_SERVERS
Summary	count of data synchroniztion server
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	1
MAX	128
Default value	3

Description

It is the count of data synchroniztion server.

10.52 CLUSTER_DEADLOCK_TIMEOUT

Basic Information

Item	Description
Name	CLUSTER_DEADLOCK_TIMEOUT
Summary	a time limit (sec) for resolving cluster deadlock
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	1
MAX	3600 (1 hour)
Default value	3600 (1 hour)

Description

If the competition to occupy the cluster server becomes keen due to the lack of the lockable cluster server, then the cluster deadlock may occur. When cluster deadlock occurs, it waits for the deadlock to be resolved as long as the time set in this property. However, if it is not resolved, then CLUSTER_DEADLOCK_TIMEOUT error occurs.

10.53 CLUSTER_DISPATCHER_IN_QUEUE_SIZE

Basic Information

Item	Description
Name	CLUSTER_DISPATCHER_IN_QUEUE_SIZE
Summary	in-queue size for cluster dispatcher
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	1024
MAX	32768
Default value	1024

Description

It is the in-queue size for cluster dispatcher

10.54 CLUSTER_DISPATCHER_NUMA_STREAM_MAP

Basic Information

Item	Description
Name	CLUSTER_DISPATCHER_NUMA_STREAM_MAP
Summary	numa stream map for cluster dispatcher
Data type	VARCHAR
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	N/A
MAX	N/A
Default value	'x' : no binding

Description

It determines a NUMA node to which the cluster dispatcher is to be connected. This property is operated when NUMA property is set to on.

If CLUSTER_COMMIT_STREAM_ISOLATION property is set to on, then the 0 stream is set to NUMA node of a commit stream.

The following is an example of three dispatchers. It connects number 0 stream to number 0 NUMA node, connects number 1 stream to number 1 NUMA node, and number 2 stream to number 2 NUMA node.

```
CLUSTER_DISPATCHER_NUMA_STREAM_MAP = '0:1:2'
```

10.55 CLUSTER_DISPATCHER_OUT_QUEUE_SIZE

Basic Information

Item	Description
Name	CLUSTER_DISPATCHER_OUT_QUEUE_SIZE
Summary	out-queue size for cluster dispatcher
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	1024
MAX	32768
Default value	1024

Description

It is the out-queue size for cluster dispatcher.

10.56 CLUSTER_GSERVER_RESPONSE_QUEUE_SIZE

Basic Information

Item	Description
Name	CLUSTER_GSERVER_RESPONSE_QUEUE_SIZE
Summary	response queue size for cluster server
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	30
MAX	32768
Default value	30

Description

It sets the maximum queue size to receive the response from the remote server.

ALIAS

Item	Description
Original name	CLUSTER_GSERVER_RESPONSE_QUEUE_SIZE
ALIAS	CLUSTER_SERVER_RESPONSE_QUEUE_SIZE

10.57 CLUSTER_HEARTBEAT_INTERVAL

Basic Information

Item	Description
Name	CLUSTER_HEARTBEAT_INTERVAL
Summary	interval seconds for health checking of cluster
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	86400
Default value	3

Description

It is the interval seconds for health checking of cluster. 0 means that it is disabled.

10.58 CLUSTER_HEARTBEAT_RETRY_COUNT

Basic Information

Item	Description
Name	CLUSTER_HEARTBEAT_RETRY_COUNT
Summary	retry count for health checking of cluster
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	1
MAX	65536
Default value	5

Description

It is the retry count for health checking of cluster.

10.59 CLUSTER_IGNORE_INACTIVE_MEMBER

Basic Information

Item	Description
Name	CLUSTER_IGNORE_INACTIVE_MEMBER
Summary	ignore in-active member for cluster
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	FALSE
MIN	0
MAX	1
Default value	NO

Description

It ignores in-active member for cluster.

10.60 CLUSTER_KEEPALIVE_IDLE_TIME

Basic Information

Item	Description
Name	CLUSTER_KEEPALIVE_IDLE_TIME
Summary	The number of seconds a cluster connection needs to be idle
Data type	BIGINT
Applicable phase	NO_MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	0
MAX	16383
Default value	0

Description

It is the idle time which is the duration without sending or receiving TCP packets between a cluster sessio n and a cdispatcher before sending a keep alive packet. In other words, if TCP packets are not exchanged during the seconds set in CLUSTER_KEEPALIVE_IDLE_TIME, then the keep alive mechanism is performed on cdispatcher side to detect the dead connection.

The default value is 0, in which case the keep alive feature is disabled.

10.61 CLUSTER_LOCKABLE_CSERVERS

Basic Information

Item	Description
Name	CLUSTER_LOCKABLE_CSERVERS
Summary	number of lockable cserver processes
Data type	BIGINT
Applicable phase	MOUNT or below
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	1
MAX	2048
Default value	5

Description

It sets the number of cluster server processes performing the operation which acquires lock. However, th e number of cluster server processes performing the operation which does not acquire lock is set by using **CLUSTER_LOCKLESS_CSERVERS**.

ALIAS

Item	Description
Original name	CLUSTER_LOCKABLE_CSERVERS
ALIAS	CSERVERS

10.62 CLUSTER_LOCKLESS_CSERVERS

Basic Information

Item	Description
Name	CLUSTER_LOCKLESS_CSERVERS
Summary	number of lockless cserver processes
Data type	BIGINT
Applicable phase	MOUNT or below
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	1
MAX	2048
Default value	5

Description

It sets the number of cluster server processes performing the operation which does not acquire lock. How ever, the number of cluster server processes performing the operation which acquires lock is set by using **CLUSTER_LOCKABLE_CSERVERS**.

ALIAS

Item	Description
Original name	CLUSTER_LOCKLESS_CSERVERS
ALIAS	LOCKLESS_CSERVERS

10.63 CLUSTER_MAX_PACKET_SIZE

Basic Information

Item	Description
Name	CLUSTER_MAX_PACKET_SIZE
Summary	maximum packet size for cluster session
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	10 Mega
MAX	32 Giga
Default value	100 Mega

Description

It sets the maximum packet size of which the remote protocol can transfer at a time. If the column size to be remotely transferred exceeds the property size, then the property size should be set biggerthen the col umn size.

10.64 CLUSTER_MAX_PAYLOAD_SIZE

Basic Information

Item	Description
Name	CLUSTER_MAX_PAYLOAD_SIZE
Summary	maximum packet payload size for cluster session (byte)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	524288
MAX	33554432
Default value	524288

Description

The cluster packet which is remotely transferred may be delivered in pieces, and this property sets the ma
ximum size of data to be stored in a piece.

10.65 CLUSTER_PACKET_ALLOCATION_TIMEOUT

Basic Information

Item	Description
Name	CLUSTER_PACKET_ALLOCATION_TIMEOUT
Summary	a time limit (sec) for how long statemets will wait to allocate packet memory
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	1
MAX	100000000
Default value	3

Description

It sets the maximum time (second) of waiting when allocating memory required for cluster packet config uration.

10.66 CLUSTER_PROTOCOL_FAILOVER_POLICY_TIMEOUT

Basic Information

Item	Description
Name	CLUSTER_PROTOCOL_FAILOVER_POLICY_TIMEOUT
Summary	a time limit of failover policy to wait for a response from the cluster protocol
Data type	BIGINT
Applicable phase	NO_MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	100000000
Default value	0

Description

It is the maximum time of when waiting for the response after transferring the protocol in cluster. If it does not responds within the specified time, then GOLDILOCKS, depending on the protocol, may terminate the protocol or make the remote cluster member which does not responds to be failover. Specify the time limit by using *CLUSTER_PROTOCOL_FAILOVER_POLICY_TIMEOUT* property to use the failover policy. However, specify the time limit by using **CLUSTER_PROTOCOL_SESSION_FATAL_POLICY_TIMEOUT** property to use the policy terminating the session.

10.67 CLUSTER_PROTOCOL_SESSION_FATAL_POLICY_TIMEOUT

Basic Information

Item	Description
Name	CLUSTER_PROTOCOL_SESSION_FATAL_POLICY_TIMEOUT
Summary	a time limit of session fatal policy to wait for a response from the cluster protocol
Data type	BIGINT
Applicable phase	NO_MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	100000000
Default value	0

Description

It is the maximum time of when waiting for the response after transferring the protocol in cluster. If it does not responds within the specified time, then GOLDILOCKS, depending on the protocol, may terminate the protocol or make the remote cluster member which does not responds to be failover. Specify the time limit by using *CLUSTER_PROTOCOL_SESSION_FATAL_POLICY_TIMEOUT* property to use the policy terminating the session. However, specify the time limit by using **CLUSTER_PROTOCOL_FAILOVER_POLICY_TIMEOUT** property to use the failover policy.

10.68 CLUSTER_SESSION_HASH_BUCKETS

Basic Information

Item	Description
Name	CLUSTER_SESSION_HASH_BUCKETS
Summary	Number of hash buckets for cluster sessions
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	127
MAX	1073741824
Default value	127

Description

It sets the number of hash buckets to control the cluster session.

10.69 CLUSTER_SPLIT_BRAIN_RESOLUTION_POLICY

Basic Information

Item	Description
Name	CLUSTER_SPLIT_BRAIN_RESOLUTION_POLICY
Summary	split brain resolution policy for cluster system
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	2
Default value	0

Description

It sets the policy to resolve split-brain situation in the cluster system. If the value is set to 1 or over, it enquires the solution of a locator.

If the query for a locator is timed out, it tries to enquire as many times as CLUSTER_SPLIT_BRAIN_RETRY_COUNT. If the property value after the retry failure is 1, then it forcibly proceeds the fail over. If it is 2, then it terminates the fatal.

10.70 CLUSTER_SPLIT_BRAIN_RETRY_COUNT

Basic Information

Item	Description
Name	CLUSTER_SPLIT_BRAIN_RETRY_COUNT
Summary	retry count for split brain resolution policy(1 ~ 65536)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	1
MAX	65536
Default value	1

Description

It is used when CLUSTER_SPLIT_BRAIN_RESOLUTION_POLICY is set to 1 or over in cluster system. It sets t he times of retrying to enquire when the query to a locator does not respond.

10.71 COMMITTER_HOT_POLICY_INTERVAL

Basic Information

Item	Description
Name	COMMITTER_HOT_POLICY_INTERVAL
Summary	committer deque interval for busy waiting
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	86400000000 (1 day)
Default value	0 (cold policy)

Description

It sets the timezone interval of busy waiting when the commit cserver is dequeing to read the commit pro tocol message. If it is set to 1,000,000 (1 second), and the time is not passed over 1 second from the last deque success to another deque retry, then it sets the timeout in deque to 0 and performs the busy waiti ng.

10.72 CONTROL_FILE_0 ~ CONTROL_FILE_7

Basic Information

Item	Description
Name	CONTROL_FILE_0
Summary	control file name
Data type	VARCHAR
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	N/A
MAX	N/A
Default value	$GOLDILOCKS_DATA/wal/control_0.ctl

Description

If a control file is corrupted, database can not be used. Therefore, the control file is multiplexed for stabili ty of database. It specifies the directory and file name of which stores each control file.

10.73 CONTROL_FILE_COUNT

Basic Information

Item	Description
Name	CONTROL_FILE_COUNT
Summary	control file count
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	2
MAX	8
Default value	2

Description

If a control file is corruped, database can not be used. The control file is multiplexed for stability of datab ase. CONTROL_FILE_COUNT specifies the multiplexing number of control files. A control file is multiplexe d at least 2 up to 8.

10.74 CONTROL_FILE_TEMP_NAME

Basic Information

Item	Description
Name	CONTROL_FILE_TEMP_NAME
Summary	temporary file name for control file
Data type	VARCHAR
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	DEFERRED
MIN	N/A
MAX	N/A
Default value	$GOLDILOCKS_DATA/backup/control.tmp

Description

During database operation, a control file is frequently changed, and its temporary copy can be made if ne cessary. CONTROL_FILE_TEMP_NAME specifies the directory and its file name to temporarily store the co ntrol file.

10.75 COORDINATOR_COMMIT_WRITE_MODE

Basic Information

Item	Description
Name	COORDINATOR_COMMIT_WRITE_MODE
Summary	coordinator commit write mode(0:disable, 1:wait mode)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	0

Description

It is a commit write mode applied to a coordinator. If TRANSACTION_COMMIT_WRITE_MODE is *no wait*, and its property is *wait*, then the coordinator node is operated as *wait*, and other nodes are operated as *no wait*.

10.76 DA_CLIENT_NUMA_NODE

Basic Information

Item	Description
Name	DA_CLIENT_NUMA_NODE
Summary	numa node for DA clients
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	-1
MAX	63
Default value	-1

Description

It sets the NUMA node ID to which the direct access (D/A) session is to be bound. This property is operate d when NUMA property is set to ON.

10.77 DATA_STORE_MODE

Basic Information

Item	Description
Name	DATA_STORE_MODE
Summary	data store mode(cds:1,tds:2)
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	1
MAX	2
Default value	2

Description

It sets the storing method of database.

- 1: CDS mode supports the concurrency for multiple users but it does not guarantee the durability. It d oes not record logs for all update operations such as insert/ delete/ update data, consequentially a fai lure can not be recovered.
- 2: TDS mode guarantee s the concurrency for multiple users and the durability using logs.

10.78 DATABASE_INSTANCE_NAME

Basic Information

Item	Description
Name	DATABASE_INSTANCE_NAME
Summary	database instance name
Data type	VARCHAR
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	N/A
MAX	N/A
Default value	GOLDILOCKS

Description

It is the database instance name.

10.79 DDL_AUTOCOMMIT

Basic Information

Item	Description
Name	DDL_AUTOCOMMIT
Summary	DDL auto commit
Data type	BOOL
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	NO

Description

It sets whether to autocommit DDL operations which are not autocommitted yet. For example, autocommit is not applied to the operations such as creating/altering a table, so if DDL_AUTOCOMMIT is 0, a table creation and alteration can be undone by the rollback. On the other hand, if DDL_AUTOCOMMIT is 1, DDL to which autocommit is not applied is committed immediately.

10.80 DDL_LOCK_TIMEOUT

Basic Information

Item	Description
Name	DDL_LOCK_TIMEOUT
Summary	a time limit (sec) for how long DDL statemets will wait
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	100000000
Default value	0

Description

It means the lock wait time when DDL operations occur to the same object at the same time.
The default value is 0 second, and it does not wait for a lock when the DDL operation occurs.

When operations of altering a table structure simultaneously occur, they wait for the time specified in DD L_LOCK_TIMEOUT, without waiting for other transactions termination as follows.

- Transaction A

```
ALTER TABLE t1 ADD COLUMN ( new_column NUMBER );
```

- Transaction B

```
TRUNCATE TABLE t1;
```

If the waiting time exceeds DDL_LOCK_TIMEOUT, an error occurs as follows.

```
gSQL> TRUNCATE TABLE t1;
ERR-HYT00(14026): resource busy or timeout expired
```

10.81 DEADLOCK_PRIORITY

Basic Information

Item	Description
Name	DEADLOCK_PRIORITY
Summary	importance to choose deadlock victim
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	FALSE
MIN	0
MAX	9
Default value	5

Description

When a deadlock occurs while simultaneously processing multiple transactions, a specific transaction wit h the low weight is selected as a victim among transactions which caused the deadlock, to solve the prob lem. If a deadlock occurs between a transaction started in sessions which have higher value for this prope rty and a transaction started in sessions which have lower value for this property, then latter is selected as a deadlock victim. Therefore, set this property according to the priority of each transaction.

Start the transaction after setting this property value so that this value is applied as a weight of that trans action. The transaction weight is not altered if this value is changed after the transaction already has bee n started.

10.82 DEFAULT_GLOBAL_SECONDARY_INDEX_CREATION

Basic Information

Item	Description
Name	DEFAULT_GLOBAL_SECONDARY_INDEX_CREATION
Summary	specifies whether or not create global secondary index at table creation
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	FALSE
MIN	0
MAX	1
Default value	YES

Description

It sets whether to create the global secondary index when creating a table in cluster system. A non-deterministic query for the table which did not created the global secondary index fails. The global secondary index can be separately created after creating the table when the property is set to NO.

10.83 DEFAULT_INDEX_LOGGING


It is not supported after 3.2.

Basic Information

Item	Description
Name	DEFAULT_INDEX_LOGGING
Summary	default logging flag of indexes
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	NO

Description

If LOGGING property is not explicitly set by a user when an index is created, then it is set to DEFAULT_INDEX_LOGGING value. If an index is created in LOGGING tablespace, the LOGGING property should be set.

10.84 DEFAULT_INDEX_PCTFREE

Basic Information

Item	Description
Name	DEFAULT_INDEX_PCTFREE
Summary	default pctfree value of indexes (%)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	99
Default value	10

Description

If a user does not explicitly specify PCTFREE syntax when creating an index. The PCTFREE is set to DEFAUL T_INDEX_PCTFREE property value.

10.85 DEFAULT_INITRANS

Basic Information

Item	Description
Name	DEFAULT_INITRANS
Summary	default initrans value of tables
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	1
MAX	32
Default value	4

Description

If a user does not explicitly set INITRANS syntax when creating a table or an index, then it is set to DEFAULT_INITRANS property value.

10.86 DEFAULT_MAXTRANS

Basic Information

Item	Description
Name	DEFAULT_MAXTRANS
Summary	default maxtrans value of tables
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	1
MAX	32
Default value	32

Description

If a user does not explicitly set the MAXTRANS syntax when creating a table or an index, then it is set to DEFAULT_MAXTRANS property value.

10.87 DEFAULT_PCTFREE

Basic Information

Item	Description
Name	DEFAULT_PCTFREE
Summary	default pctfree value of tables (%)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	99
Default value	10

Description

If a user does not explicitly set the PCTFREE property when creating a table, it is set to DEFAULT_PCTFREE property value.

10.88 DEFAULT_PCTUSED

Basic Information

Item	Description
Name	DEFAULT_PCTUSED
Summary	default pctused value of tables (%)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	99
Default value	60

Description

If a user does not explicitly set the PCTUSED property when creating a table, it is set to DEFAULT_PCTUSED value.

10.89 DEFAULT_REMOVAL_BACKUP_FILE

Basic Information

Item	Description
Name	DEFAULT_REMOVAL_BACKUP_FILE
Summary	default removal flag of incremental backup files
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	NO

Description

It specifies whether to delete the backup file when deleting the backup list.

10.90 DEFAULT_REMOVAL_OBSOLETE_BACKUP_LIST

Basic Information

Item	Description
Name	DEFAULT_REMOVAL_OBSOLETE_BACKUP_LIST
Summary	default removal flag of obsolete incremental backup lists
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	NO

Description

It specifies whether to delete the previous obsoleted backup list when executing INCREMENTAL BACKUP.

10.91 DEFAULT_SHARDING

Basic Information

Item	Description
Name	DEFAULT_SHARDING
Summary	default sharding strategy (0: cloned, 1: hash sharding)
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	DEFERRED
MIN	0
MAX	1
Default value	NO

Description

It sets the default sharding strategy to be used if the sharding strategy is not determined when creating a table.

The following is an example of executing CREATE TABLE statement.

```
CREATE TABLE t1
(
    id   INTEGER,
    name VARCHAR(128)
);
```

If DEFAULT_SHARDING value is 0 (cloned), the table is created as a cloned table as follows.

```
CREATE TABLE t1
(
    id   INTEGER,
    name VARCHAR(128)
)
CLONED
```

```
AT CLUSTER WIDE
;
```

If DEFAULT_SHARDING value is 1 (hash sharding), the table is created as a hash-sharded table as follows.

```
CREATE TABLE t1
(
    id   INTEGER,
    name VARCHAR(128)
)
SHARDING BY HASH (id)
    SHARD COUNT 24
    AT CLUSTER WIDE
;
```

If DEFAULT_SHARDING is 1 (hash sharding) and <table sharding strategy> is not described, then the hash sharding key is determined based on the following order.

1. If PRIMARY KEY constraint is defined, primary key is used as a sharding key.

- The original message

```
CREATE TABLE t1 ( id INTEGER PRIMARY KEY, name VARCHAR(128) );
```

- Translation

```
CREATE TABLE t1 ( id INTEGER PRIMARY KEY, name VARCHAR(128) )
    SHARDING BY HASH(id)
    SHARD COUNT 24
    AT CLUSTER WIDE;
```

2. If UNIQUE constraint is defined, the firstly described UNIQUE constraint is used as the sharding key.

- The original message

```
CREATE TABLE t1 ( id INTEGER, name VARCHAR(128) UNIQUE );
```

- Translation

```
CREATE TABLE t1 ( id INTEGER, name VARCHAR(128) UNIQUE )
    SHARDING BY HASH(name)
    SHARD COUNT 24
    AT CLUSTER WIDE;
```

3. If key constraint is not defined, the first column except for the following excluded data type is used a s a sharding key.
    - The excluded data type: LONG VARCHAR, LONG VARBINARY, BOOLEAN

- The original message

```
CREATE TABLE t1 ( is_man BOOLEAN, id INTEGER, name VARCHAR(128) );
```

- Translation

```
CREATE TABLE t1 ( is_man BOOLEAN, id INTEGER, name VARCHAR(128) )
    SHARDING BY HASH(id)
    SHARD COUNT 24
    AT CLUSTER WIDE;
```

10.92 DISABLE_DDL

Basic Information

Item	Description
Name	DISABLE_DDL
Summary	disable All DDL
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	NO

Description

It prevents all DDL execution.

The SQL statements which are affected by DISABLE_DDL are queried as follows.

```
gSQL>
SELECT command
  FROM v$sql_command
 WHERE is_ddl = 'YES'
 ORDER BY 1
;
COMMAND
-------------------------------------------------------
ALTER AUDIT POLICY
ALTER CLUSTER GROUP .. ADD CLUSTER MEMBER
ALTER CLUSTER GROUP .. OFFLINE CLUSTER MEMBER
ALTER DATABASE ADD LOGFILE GROUP
ALTER DATABASE ADD LOGFILE MEMBER
ALTER DATABASE ARCHIVELOG
ALTER DATABASE CLEAR PASSWORD HISTORY
```

```
ALTER DATABASE DATAFILE AUTOEXTEND ..
ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS
ALTER DATABASE DROP LOGFILE GROUP
ALTER DATABASE DROP LOGFILE MEMBER
ALTER DATABASE NOARCHIVELOG
ALTER DATABASE OFFLINE INACTIVE CLUSTER MEMBERS
ALTER DATABASE RENAME GLOBAL TRANSACTION LOGFILE
ALTER DATABASE RENAME LOGFILE
ALTER DATABASE RESET LOCAL CLUSTER MEMBER
ALTER FUNCTION
ALTER INDEX .. REBUILD
ALTER INDEX .. RENAME
ALTER INDEX .. STORAGE
ALTER INDEX AGING
ALTER PACKAGE
ALTER PROCEDURE
ALTER PROFILE
ALTER SEQUENCE
ALTER SEQUENCE .. SYNCHRONIZE
ALTER SYSTEM SWITCH LOGFILE
ALTER TABLE .. ADD COLUMN
ALTER TABLE .. ADD CONSTRAINT
ALTER TABLE .. ADD GLOBAL SECONDARY INDEX
ALTER TABLE .. ADD SUPPLEMENTAL LOG
ALTER TABLE .. ALTER COLUMN .. AS IDENTITY
ALTER TABLE .. ALTER COLUMN .. DROP DEFAULT
ALTER TABLE .. ALTER COLUMN .. DROP IDENTITY
ALTER TABLE .. ALTER COLUMN .. DROP NOT NULL
ALTER TABLE .. ALTER COLUMN .. SET DATA TYPE
ALTER TABLE .. ALTER COLUMN .. SET DEFAULT
ALTER TABLE .. ALTER COLUMN .. SET NOT NULL
ALTER TABLE .. ALTER CONSTRAINT
ALTER TABLE .. ALTER GLOBAL SECONDARY INDEX
ALTER TABLE .. ALTER GLOBAL SECONDARY INDEX AGING
ALTER TABLE .. DROP CONSTRAINT
ALTER TABLE .. DROP GLOBAL SECONDARY INDEX
ALTER TABLE .. DROP OFFLINE SEGMENTS
ALTER TABLE .. DROP SUPPLEMENTAL LOG
ALTER TABLE .. MERGE SHARDS .. INTO ..
ALTER TABLE .. MOVE SHARD .. TO CLUSTER GROUP ..
ALTER TABLE .. READ ONLY
```

```
ALTER TABLE .. READ WRITE
ALTER TABLE .. REBALANCE ..
ALTER TABLE .. REBUILD GLOBAL SECONDARY INDEX
ALTER TABLE .. RENAME COLUMN
ALTER TABLE .. RENAME CONSTRAINT
ALTER TABLE .. RENAME SHARD .. TO ..
ALTER TABLE .. RENAME TO ..
ALTER TABLE .. SET UNUSED COLUMN
ALTER TABLE .. SPLIT SHARD .. INTO .. AT CLUSTER GROUP ..
ALTER TABLE .. STORAGE
ALTER TABLE .. SYNCHRONIZE ..
ALTER TABLE .. SYNCHRONIZE IDENTITY COLUMN
ALTER TABLESPACE .. ADD
ALTER TABLESPACE .. DROP
ALTER TABLESPACE .. OFFLINE
ALTER TABLESPACE .. ONLINE
ALTER TABLESPACE .. RENAME TO
ALTER TABLESPACE .. RENAME { DATAFILE ¦ MEMORY }
ALTER USER
ALTER USER .. IDENTIFIED BY
ALTER VIEW
ANALYZE SYSTEM COMPUTE STATISTICS
ANALYZE SYSTEM DELETE STATISTICS
ANALYZE TABLE .. DELETE STATISTICS
ANALYZE TABLE .. [COMPUTE¦ESTIMATE] STATISTICS
AUDIT POLICY
COMMENT ON .. IS
CREATE AUDIT POLICY
CREATE CLUSTER GROUP
CREATE FUNCTION
CREATE INDEX
CREATE PACKAGE
CREATE PACKAGE BODY
CREATE PROCEDURE
CREATE PROFILE
CREATE SCHEMA
CREATE SEQUENCE
CREATE SYNONYM
CREATE TABLE
CREATE TABLE ... AS SELECT
CREATE TABLESPACE
```

```
CREATE USER
CREATE VIEW
DROP AUDIT POLICY
DROP CLUSTER GROUP
DROP FUNCTION
DROP INDEX
DROP PACKAGE
DROP PROCEDURE
DROP PROFILE
DROP SCHEMA
DROP SEQUENCE
DROP SYNONYM
DROP TABLE
DROP TABLESPACE
DROP USER
DROP VIEW
FLASHBACK TABLE
GRANT .. ON DATABASE
GRANT .. ON PACKAGE
GRANT .. ON PROCEDURE
GRANT .. ON SCHEMA
GRANT .. ON TABLE
GRANT .. ON TABLESPACE
GRANT USAGE ON ..
NOAUDIT POLICY
PURGE CONSTRAINT
PURGE DBA_RECYCLEBIN
PURGE INDEX
PURGE RECYCLEBIN
PURGE TABLE
PURGE TABLESPACE
REVOKE .. ON DATABASE
REVOKE .. ON PACKAGE
REVOKE .. ON PROCEDURE
REVOKE .. ON SCHEMA
REVOKE .. ON TABLE
REVOKE .. ON TABLESPACE
REVOKE USAGE ON ..
TRUNCATE TABLE
128 rows selected.
```

10.93 DISABLE_DDL_CDC_GIVEUP

Basic Information

Item	Description
Name	DISABLE_DDL_CDC_GIVEUP
Summary	disable DDL which causing CDC give-up
Data type	BOOLEAN
Applicable phase	OPEN
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	TRUE
MIN	0
MAX	1
Default value	NO

Description

It prohibits DDL operation on the table of supplemental log, because it affects CDC's give up.
For more information, refer to **The occurrence of give up and whether to allow DDL statement according to DDL category**.

10.94 DISABLE_SERIAL_DDL

Basic Information

Item	Description
Name	DISABLE_SERIAL_DDL
Summary	disable Serial DDL
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	NO

Description

DDL is executed for all cluster members after sequentially acquiring locks in cluster environment as described in **Processing DDL in Cluster**.
The DDL which is performed in the serial lock method is called as the serial DDL.

DDL statements which are affected by DISABLE_SERIAL_DDL are queried as follows, and most of schema DDLs are affected.

```
gSQL>
SELECT command
     , is_ddl
     , cluster_lock_mode
  FROM v$sql_command
 WHERE is_ddl = 'YES'
   AND cluster_lock_mode = 'SERIAL'
 ORDER BY 1
;
SELECT command
     , is_ddl
     , cluster_lock_mode
```

```
  FROM v$sql_command
 WHERE is_ddl = 'YES'
   AND cluster_lock_mode = 'SERIAL'
 ORDER BY 1
;
COMMAND                                          IS_DDL CLUSTER_LOCK_MODE
------------------------------------------------ ------ -----------------
ALTER AUDIT POLICY                               YES    SERIAL
ALTER DATABASE CLEAR PASSWORD HISTORY            YES    SERIAL
ALTER DATABASE DATAFILE AUTOEXTEND ..            YES    SERIAL
ALTER FUNCTION                                   YES    SERIAL
ALTER INDEX .. RENAME                            YES    SERIAL
ALTER INDEX .. STORAGE                           YES    SERIAL
ALTER INDEX AGING                                YES    SERIAL
ALTER PACKAGE                                    YES    SERIAL
ALTER PROCEDURE                                  YES    SERIAL
ALTER PROFILE                                    YES    SERIAL
ALTER SEQUENCE                                   YES    SERIAL
ALTER TABLE .. ADD COLUMN                        YES    SERIAL
ALTER TABLE .. ADD CONSTRAINT                    YES    SERIAL
ALTER TABLE .. ADD GLOBAL SECONDARY INDEX        YES    SERIAL
ALTER TABLE .. ADD SUPPLEMENTAL LOG              YES    SERIAL
ALTER TABLE .. ALTER COLUMN .. AS IDENTITY       YES    SERIAL
ALTER TABLE .. ALTER COLUMN .. DROP DEFAULT      YES    SERIAL
ALTER TABLE .. ALTER COLUMN .. DROP IDENTITY     YES    SERIAL
ALTER TABLE .. ALTER COLUMN .. DROP NOT NULL     YES    SERIAL
ALTER TABLE .. ALTER COLUMN .. SET DATA TYPE     YES    SERIAL
ALTER TABLE .. ALTER COLUMN .. SET DEFAULT       YES    SERIAL
ALTER TABLE .. ALTER COLUMN .. SET NOT NULL      YES    SERIAL
ALTER TABLE .. ALTER CONSTRAINT                  YES    SERIAL
ALTER TABLE .. ALTER GLOBAL SECONDARY INDEX      YES    SERIAL
ALTER TABLE .. ALTER GLOBAL SECONDARY INDEX AGING YES   SERIAL
ALTER TABLE .. DROP CONSTRAINT                   YES    SERIAL
ALTER TABLE .. DROP GLOBAL SECONDARY INDEX       YES    SERIAL
ALTER TABLE .. DROP OFFLINE SEGMENTS             YES    SERIAL
ALTER TABLE .. DROP SUPPLEMENTAL LOG             YES    SERIAL
ALTER TABLE .. READ ONLY                         YES    SERIAL
ALTER TABLE .. READ WRITE                        YES    SERIAL
ALTER TABLE .. RENAME COLUMN                     YES    SERIAL
ALTER TABLE .. RENAME CONSTRAINT                 YES    SERIAL
ALTER TABLE .. RENAME SHARD .. TO ..             YES    SERIAL
```

```
ALTER TABLE .. RENAME TO ..                          YES    SERIAL
ALTER TABLE .. SET UNUSED COLUMN                     YES    SERIAL
ALTER TABLE .. STORAGE                               YES    SERIAL
ALTER TABLESPACE .. ADD                              YES    SERIAL
ALTER TABLESPACE .. DROP                             YES    SERIAL
ALTER TABLESPACE .. OFFLINE                          YES    SERIAL
ALTER TABLESPACE .. ONLINE                           YES    SERIAL
ALTER TABLESPACE .. RENAME TO                        YES    SERIAL
ALTER TABLESPACE .. RENAME { DATAFILE ¦ MEMORY }     YES    SERIAL
ALTER USER                                           YES    SERIAL
ALTER USER .. IDENTIFIED BY                          YES    SERIAL
ALTER VIEW                                           YES    SERIAL
ANALYZE SYSTEM COMPUTE STATISTICS                    YES    SERIAL
ANALYZE SYSTEM DELETE STATISTICS                     YES    SERIAL
ANALYZE TABLE .. DELETE STATISTICS                   YES    SERIAL
ANALYZE TABLE .. [COMPUTE¦ESTIMATE] STATISTICS       YES    SERIAL
AUDIT POLICY                                         YES    SERIAL
COMMENT ON .. IS                                     YES    SERIAL
CREATE AUDIT POLICY                                  YES    SERIAL
CREATE FUNCTION                                      YES    SERIAL
CREATE INDEX                                         YES    SERIAL
CREATE PACKAGE                                       YES    SERIAL
CREATE PACKAGE BODY                                  YES    SERIAL
CREATE PROCEDURE                                     YES    SERIAL
CREATE PROFILE                                       YES    SERIAL
CREATE SCHEMA                                        YES    SERIAL
CREATE SEQUENCE                                      YES    SERIAL
CREATE SYNONYM                                       YES    SERIAL
CREATE TABLE                                         YES    SERIAL
CREATE TABLE ... AS SELECT                           YES    SERIAL
CREATE TABLESPACE                                    YES    SERIAL
CREATE USER                                          YES    SERIAL
CREATE VIEW                                          YES    SERIAL
DROP AUDIT POLICY                                    YES    SERIAL
DROP FUNCTION                                        YES    SERIAL
DROP INDEX                                           YES    SERIAL
DROP PACKAGE                                         YES    SERIAL
DROP PROCEDURE                                       YES    SERIAL
DROP PROFILE                                         YES    SERIAL
DROP SCHEMA                                          YES    SERIAL
DROP SEQUENCE                                        YES    SERIAL
```

```
DROP SYNONYM                                YES    SERIAL
DROP TABLE                                  YES    SERIAL
DROP TABLESPACE                             YES    SERIAL
DROP USER                                   YES    SERIAL
DROP VIEW                                   YES    SERIAL
FLASHBACK TABLE                             YES    SERIAL
GRANT .. ON DATABASE                        YES    SERIAL
GRANT .. ON PACKAGE                         YES    SERIAL
GRANT .. ON PROCEDURE                       YES    SERIAL
GRANT .. ON SCHEMA                          YES    SERIAL
GRANT .. ON TABLE                           YES    SERIAL
GRANT .. ON TABLESPACE                      YES    SERIAL
GRANT USAGE ON ..                           YES    SERIAL
NOAUDIT POLICY                              YES    SERIAL
PURGE CONSTRAINT                            YES    SERIAL
PURGE DBA_RECYCLEBIN                        YES    SERIAL
PURGE INDEX                                 YES    SERIAL
PURGE RECYCLEBIN                            YES    SERIAL
PURGE TABLE                                 YES    SERIAL
PURGE TABLESPACE                            YES    SERIAL
REVOKE .. ON DATABASE                       YES    SERIAL
REVOKE .. ON PACKAGE                        YES    SERIAL
REVOKE .. ON PROCEDURE                      YES    SERIAL
REVOKE .. ON SCHEMA                         YES    SERIAL
REVOKE .. ON TABLE                          YES    SERIAL
REVOKE .. ON TABLESPACE                     YES    SERIAL
REVOKE USAGE ON ..                          YES    SERIAL
TRUNCATE TABLE                              YES    SERIAL
103 rows selected.
```

DDL statements which are not affected by DISABLE_SERIAL_DDL are queried as follows, and most of clus ter DDLs are affected.

```
gSQL>
SELECT command
     , is_ddl
     , cluster_lock_mode
  FROM v$sql_command
 WHERE is_ddl = 'YES'
   AND cluster_lock_mode <> 'SERIAL'
 ORDER BY 1
```

```
;
COMMAND                                                  IS_DDL CLUSTER_LOCK_MODE
-------------------------------------------------------- ------ ----------------
ALTER CLUSTER GROUP .. ADD CLUSTER MEMBER                YES    MANUAL
ALTER CLUSTER GROUP .. OFFLINE CLUSTER MEMBER            YES    MANUAL
ALTER DATABASE ADD LOGFILE GROUP                         YES    NONE
ALTER DATABASE ADD LOGFILE MEMBER                        YES    NONE
ALTER DATABASE ARCHIVELOG                                YES    NONE
ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS             YES    MANUAL
ALTER DATABASE DROP LOGFILE GROUP                        YES    NONE
ALTER DATABASE DROP LOGFILE MEMBER                       YES    NONE
ALTER DATABASE NOARCHIVELOG                              YES    NONE
ALTER DATABASE OFFLINE INACTIVE CLUSTER MEMBERS          YES    MANUAL
ALTER DATABASE RENAME GLOBAL TRANSACTION LOGFILE         YES    NONE
ALTER DATABASE RENAME LOGFILE                            YES    NONE
ALTER DATABASE RESET LOCAL CLUSTER MEMBER                YES    NONE
ALTER INDEX .. REBUILD                                   YES    MANUAL
ALTER SEQUENCE .. SYNCHRONIZE                            YES    MANUAL
ALTER SYSTEM SWITCH LOGFILE                              YES    NONE
ALTER TABLE .. MERGE SHARDS .. INTO ..                   YES    MANUAL
ALTER TABLE .. MOVE SHARD .. TO CLUSTER GROUP ..         YES    MANUAL
ALTER TABLE .. REBALANCE ..                              YES    MANUAL
ALTER TABLE .. REBUILD GLOBAL SECONDARY INDEX            YES    MANUAL
ALTER TABLE .. SPLIT SHARD .. INTO .. AT CLUSTER GROUP .. YES    MANUAL
ALTER TABLE .. SYNCHRONIZE ..                            YES    MANUAL
ALTER TABLE .. SYNCHRONIZE IDENTITY COLUMN               YES    MANUAL
CREATE CLUSTER GROUP                                     YES    MANUAL
DROP CLUSTER GROUP                                       YES    MANUAL
25 rows selected.
```

10.95 DISABLE_UPDATE_PK_CDC_GIVEUP

Basic Information

Item	Description
Name	DISABLE_UPDATE_PK_CDC_GIVEUP
Summary	disable UPDATE primary key which caused CDC give-up
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	NO

Description

It disables UPDATE primary key which caused CDC give up.

10.96 DISALLOWED_PROTOCOL_TARGETTYPE

Basic Information

Item	Description
Name	DISALLOWED_PROTOCOL_TARGETTYPE
Summary	disallowed TARGETTYPE protocol
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	DEFERRED
MIN	0
MAX	1
Default value	NO

Description

It disallows TARGETTYPE protocol.

10.97 DISALLOWED_PROTOCOL_TARGETTYPE_WITH_ALL

Basic Information

Item	Description
Name	DISALLOWED_PROTOCOL_TARGETTYPE_WITH_ALL
Summary	disallowed TARGETTYPE_WITH_ALL protocol
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	DEFERRED
MIN	0
MAX	1
Default value	NO

Description

It disallows TARGETTYPE_WITH_ALL protocol.

10.98 DISALLOWED_PROTOCOL_TARGETTYPE_WITH_NAME

Basic Information

Item	Description
Name	DISALLOWED_PROTOCOL_TARGETTYPE_WITH_NAME
Summary	disallowed TARGETTYPE_WITH_NAME protocol
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	DEFERRED
MIN	0
MAX	1
Default value	NO

Description

It disallows TARGETTYPE_WITH_NAME protocol.

10.99 DISPATCHER_CM_BUFFER_SIZE

Basic Information

Item	Description
Name	DISPATCHER_CM_BUFFER_SIZE
Summary	communication buffer size
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	10485760
MAX	34359738368
Default value	31457280

Description

It is the size of entire communication buffer used in shared mode. It is allocated to and used in Shared Sta tic Area (SSA).

10.100 DISPATCHER_CM_UNIT_SIZE

Basic Information

Item	Description
Name	DISPATCHER_CM_UNIT_SIZE
Summary	communication unit size
Data type	BIGINT
Applicable phase	MOUNT or below
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	1024
MAX	10485760
Default value	1024

Description

It is the unit size managed by dispatcher in shared mode. If the size is large, the memory is wasted. If it is small, the performance is degraded. It is set to the maximum communication packet size in the shared m ode.

10.101 DISPATCHER_CONNECTIONS

Basic Information

Item	Description
Name	DISPATCHER_CONNECTIONS
Summary	maximum number of connections for each dispatcher
Data type	BIGINT
Applicable phase	MOUNT or below
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	10
MAX	32768
Default value	950

Description

It is the maximum number of connection (client) which a dispatcher can manage in shared mode.
If the system- supported maximum value is smaller than the set value, it is internally set to the system max imum.

10.102 DISPATCHER_HOT_POLICY_INTERVAL

Basic Information

Item	Description
Name	DISPATCHER_HOT_POLICY_INTERVAL
Summary	dispatcher dequeue interval for busy waiting
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	86400000000: 1 day
Default value	100000: 0.1 second

Description

It is the dispatcher dequeue interval for busy waiting. (micro second)

10.103 DISPATCHER_LOAD_BALANCING

Basic Information

Item	Description
Name	DISPATCHER_LOAD_BALANCING
Summary	load balancing algorithm for shared mode (0: number of clients, 1: round robin)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	0

Description

It is an algorithm allocating a dispatcher when connecting to a client in the shared mode.

- 0: It is allocated to a dispatcher of which the number of currently attached clients are small.
- 1: It is sequentially allocated to a dispatcher.

10.104 DISPATCHER_NUMA_STREAM_MAP

Basic Information

Item	Description
Name	DISPATCHER_NUMA_STREAM_MAP
Summary	numa stream map for dispatcher
Data type	VARCHAR
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	N/A
MAX	N/A
Default value	'x' : no binding

Description

It determines NUMA node to whcih dispatchers are to be connected. This property is operated when NU MA property is set to on.
The following is an example of three dispatchers. It connects number 0 stream to number 0 NUMA node, connects number 1 stream to number 1 NUMA node, and number 2 stream to number 2 NUMA node.

```
DISPATCHER_NUMA_STREAM_MAP = '0:1:2'
```

10.105 DISPATCHER_QUEUE_SIZE

Basic Information

Item	Description
Name	DISPATCHER_QUEUE_SIZE
Summary	dispatcher queue size
Data type	BIGINT
Applicable phase	MOUNT or below
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	1024
MAX	32768
Default value	1024

Description

In shared mode, it sets the queue size for the communication between the dispatcher and the shared-ser ver.

10.106 DISPATCHER_REQUEST_MINI_QUEUE_COUNT

Basic Information

Item	Description
Name	DISPATCHER_REQUEST_MINI_QUEUE_COUNT
Summary	count of mini queue per request queue
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	1
MAX	16
Default value	4

Description

It is the count of mini queue per request queue.

10.107 DISPATCHER_RESPONSE_MINI_QUEUE_COUNT

Basic Information

Item	Description
Name	DISPATCHER_RESPONSE_MINI_QUEUE_COUNT
Summary	count of mini queue per response queue
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	1
MAX	16
Default value	4

Description

It is the count of mini queue per response queue.

10.108 DISPATCHERS

Basic Information

Item	Description
Name	DISPATCHERS
Summary	number of dispatcher processes
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	1
MAX	256
Default value	2

Description

It sets the number of dispatcher processes when using the shared mode.
It can not reduce the value by using alter system on open phase.

10.109 EXECUTE_INST_HASH_TABLE_USING_AVAILABLE_MEMORY

Basic Information

Item	Description
Name	EXECUTE_INST_HASH_TABLE_USING_AVAILABLE_MEMORY
Summary	execute instant hash table using available memory even if not enough
Data type	BOOLEAN
Applicable phase	NO_MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	YES

Description

If the memory is not sufficient to expand the hash bucket in the query using an instant hash table, then it determines whether to fail the query or to perform the query without expanding the hash bucket.

10.110 FETCH_FAILOVER

Basic Information

Item	Description
Name	FETCH_FAILOVER
Summary	enable fetch failover
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	FALSE
MIN	0
MAX	1
Default value	NO

Description

It enables the fetch failover.

10.111 FULL_TABLE_SCAN_CACHING_THRESHOLD

Basic Information

Item	Description
Name	FULL_TABLE_SCAN_CACHING_THRESHOLD
Summary	upper threshold of table size for buffer caching while full scan
Data type	BIGINT
Applicable phase	NO MOUNT above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1000
Default value	20

Description

The threshold of the table size determines whether to cache the tables created in the disk tablespace to buffer cache when performing a full scan. FULL_TABLE_SCAN_CACHING_THRESHOLD sets this threshold value. The default value is 20. In this case, it caches only the tables which are using the number of pages equal to or less than 2.0% of **BUFFER_CACHE_SIZE**. If this value is 1000 (100%), then all tables are cached to the buffer when performing the full scan.

For example, if BUFFER_CACHE_SIZE is 8192 and FULL_TABLE_SCAN_CACHING_THRESHOLD is 509 (50.9%), then only the tables which are using the number of pages equal to or less than 4169 (50.9% of 8192) are cached to the buffer when performing the full scan.

10.112 GLOBAL_CONNECTION_ALLOW_SESSION_DEPENDENCY

Basic Information

Item	Description
Name	GLOBAL_CONNECTION_ALLOW_SESSION_DEPENDENCY
Summary	allowed session dependent features in global connection
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	TRUE
MIN	0
MAX	1
Default value	NO

Description

It sets whether to support the query execution including the session dependent information in the global connection.

10.113 GLOBAL_JOURNAL_BUFFER_SIZE

Basic Information

Item	Description
Name	GLOBAL_JOURNAL_BUFFER_SIZE
Summary	global journal buffer size
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	1024
MAX	10 Giga
Default value	1 Mega

Description

It is the size of global journal buffer.

10.114 GLOBAL_JOURNAL_BUFFER_TOTAL_MAX_SIZE

Basic Information

Item	Description
Name	GLOBAL_JOURNAL_BUFFER_TOTAL_MAX_SIZE
Summary	global journal buffer total max size
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	1 Mega
MAX	100 Giga
Default value	64 Mega

Description

It is the total max size of global journal buffer.

10.115 GLOBAL_PROPERTY_LOCK_TIMEOUT

Basic Information

Item	Description
Name	GLOBAL_PROPERTY_LOCK_TIMEOUT
Summary	a time limit(second) for how long global property lock statements will wait
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0 (infinite)
MAX	100000000
Default value	0

Description

When changing the global property, it performs the lock to control the concurrency. In this case, the wait ing time to perform the lock is set.

10.116 GLOBAL_TRANSACTION_COMMIT_WRITE_MODE

Basic Information

Item	Description
Name	GLOBAL_TRANSACTION_COMMIT_WRITE_MODE
Summary	global transaction commit write mode (0: no_wait, 1: wait, 2: disable)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	2
Default value	2

Description

It is a property to change the commit write mode of the global transaction. TRANSACTION_COMMIT_ WRITE_MODE property is applied to all transactions, but GLOBAL_TRANSACTION_COMMIT_WRITE_MODE property is applied only to a global transaction. If the property is set to 2, then it follows the TRANSACTIO N_COMMIT_WRITE_MODE.

- 0: It does not wait.
- 1: It waits.
- 2: It follows the value of TRANSACTION_COMMIT_WRITE_MODE.

10.117 GLOBAL_TRANSACTION_ISOLATION_SCOPE

Basic Information

Item	Description
Name	GLOBAL_TRANSACTION_ISOLATION_SCOPE
Summary	isolation scope for global transaction(0: system, 1: group)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	0

Description

It determines whether to process the data with a global transaction or with multiple domain transactions when the transaction changed the data through two cluster groups.

- 0: It processes with a global transaction.
- 1: It processes with multiple domain transactions.

If this property is set to 1, it commits each cluster group with a separate transaction, so it does n ot guarantees the transaction atomicity.

10.118 GLOBAL_TRANSACTION_LOG_DIR

Basic Information

Item	Description
Name	GLOBAL_TRANSACTION_LOG_DIR
Summary	default global transaction log directory
Data type	VARCHAR
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	DEFERRED
MIN	N/A
MAX	N/A
Default value	$GOLDILOCKS_DATA/wal

Description

It is the default directory of global transaction log.

10.119 GLOBAL_TRANSACTION_LOG_FILE_SIZE

Basic Information

Item	Description
Name	GLOBAL_TRANSACTION_LOG_FILE_SIZE
Summary	global transaction log file size
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	20 Mega
MAX	10 Giga
Default value	100 Mega

Description

It is the file size of global transaction log.

10.120 GMASTER_NUMA_NODE

Basic Information

Item	Description
Name	GMASTER_NUMA_NODE
Summary	numa node for gmaster process
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	-1
MAX	63
Default value	-1

Description

It sets the ID of NUMA node to be used by gmaster daemon. This property is operated when NUMA property is set to on.

10.121 GMON_AUTOSTART

Basic Information

Item	Description
Name	GMON_AUTOSTART
Summary	Indicate whether gmon process automatically starts or not (0 | 1)
Data type	BOOLEAN
Applicable phase	MOUNT or below
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	0
MAX	1
Default value	1

Description

It sets whether to start gmon process automatically.

10.122 HINT_ERROR

Basic Information

Item	Description
Name	HINT_ERROR
Summary	enable hint error
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	DEFERRED
MIN	0
MAX	1
Default value	NO

Description

It sets whether to check syntax error and validation error for hint syntax.

10.123 IDLE_TIMEOUT

Basic Information

Item	Description
Name	IDLE_TIMEOUT
Summary	idle timeout(s)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	10000000
Default value	0

Description

It sets the maximum IDLE time possible to wait in C/S session. If it exceeds the specified idle time, TIMEO UT error occurs.

- 0: It means infinite waiting, and TIMEOUT error does not occur.

10.124 IN_DOUBT_DECISION

Basic Information

Item	Description
Name	IN_DOUBT_DECISION
Summary	decision for in-doubt transaction
Data type	BIGINT
Applicable phase	MOUNT or below
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	1
MAX	2
Default value	2

Description

It determines whether to commit or to rollback the in-doubt transaction of distributed transactions.

- 1: Commit
- 2: Rollback

10.125 IN_KEY_RANGE_ARRAY_COUNT

Basic Information

Item	Description
Name	IN_KEY_RANGE_ARRAY_COUNT
Summary	array count for in key range scan
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	DEFERRED
MIN	1
MAX	65536
Default value	20

Description

It is the maximum number of values which are targets of *in key range* performing the *in key range scan* b ased on array.

- IN_KEY_RANGE_ARRAY_COUNT should be 3 or bigger to perform the *in key range scan* based on arr ay for the statement below.

```
gSQL> SELECT * FROM T1 WHERE C1 IN ( 1, 2, 3 )
```

If the maximum number of in key range target values are bigger than IN_KEY_RANGE_ARRAY_COUNT v alue, then in key range in key range scan is performed based on instant table.

10.126 INCREMENTAL_BACKUP_SCAN_BUFFER_SIZE

Basic Information

Item	Description
Name	INCREMENTAL_BACKUP_SCAN_BUFFER_SIZE
Summary	number of pages read in one I/O operation during an incremental backup
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	DEFERRED
MIN	1
MAX	8192
Default value	32

Description

It sets the number of pages to read by one time disk IO for performing incremental backup of the disk tablespace.

10.127 INCREMENTAL_DATAFILE_HEADER_UPDATE_CRITERIA

Basic Information

Item	Description
Name	INCREMENTAL_DATAFILE_HEADER_UPDATE_CRITERIA
Summary	criteria for the number of flush page count to update datafile header
Data type	BIGINT
Applicable phase	NO_MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	134217728
Default value	134217728

Description

It sets the criteria of updating the data file header in the disk tablespace. It sets the LSN to start recovery in the datafile header when the pages updated as many as the set value is applied while IO slave applies the pages updated in the buffer cache to the disk. In this way, disk IO is decreased at restart recovery.

10.128 INDEX_BUILD_PARALLEL_FACTOR

Basic Information

Item	Description
Name	INDEX_BUILD_PARALLEL_FACTOR
Summary	index build parallel factor
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	64
Default value	0

Description

When creating an index, it specifies the number of parallel factor.

- 0: It is specified as the number of the core factor in the system.

10.129 INDEX_MERGE_RUN_COUNT

Basic Information

Item	Description
Name	INDEX_MERGE_RUN_COUNT
Summary	merge run count for memory index
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	2
MAX	64
Default value	32

Description

Memory B-tree index of bottom-up approach is created by extracting all keys from a table, sorting them i n certain block size (INDEX_SORT_RUN_SIZE) units, merging the sorted blocks, and generating the intern al node. INDEX_MERGE_RUN_COUNT sets the number of the sorted blocks to be merged at a time.

ALIAS

Item	Description
Original name	INDEX_MERGE_RUN_COUNT
ALIAS	MEMORY_MERGE_RUN_COUNT

10.130 INDEX_REBUILD_BLOCK_READ_COUNT

Basic Information

Item	Description
Name	INDEX_REBUILD_BLOCK_READ_COUNT
Summary	value count for a block read for index rebuild
Data type	BIGINT
Applicable phase	NO_MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	DEFERRED
MIN	1
MAX	65536
Default value	100

Description

When DML is performed while rebuilding the index on ONLINE mode, the journal data is stored. The index is rebuilt based on the data at the time of beginning of the rebuilding, then the updated data during the rebuilding is applied to the index through the journal data. INDEX_REBUILD_BLOCK_READ_COUNT sets how much journal data to be read and applied to the index during this process.

10.131 INDEX_SORT_RUN_SIZE

Basic Information

Item	Description
Name	INDEX_SORT_RUN_SIZE
Summary	sort run size for memory index (byte)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	8192
MAX	32768
Default value	8192

Description

Memory B-tree index of bottom-up approach is created by extracting all keys from a table, sorting them i n a certain block size (INDEX_SORT_RUN_SIZE) unit, merging the sorted blocks, and generating the inter nal node. INDEX_SORT_RUN_SIZE sets the size of a single block to be sorted.

ALIAS

Item	Description
Original name	INDEX_SORT_RUN_SIZE
ALIAS	MEMORY_SORT_RUN_SIZE

10.132 INDEX_TREE_MERGE_PARALLEL_FACTOR

Basic Information

Item	Description
Name	INDEX_TREE_MERGE_PARALLEL_FACTOR
Summary	parallel factor for merging sub-trees
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	64
Default value	0

Description

When creating an index, it specifies the number of parallel factor to merge the sub-tree.
If that value is bigger than INDEX_BUILD_PARALLEL_FACTOR, then INDEX_BUILD_PARALLEL_FACTOR is used.

- 0: It follows INDEX_BUILD_PARALLEL_FACTOR.

10.133 INST_ALLOCATOR_COUNT

Basic Information

Item	Description
Name	INST_ALLOCATOR_COUNT
Summary	memory allocator count for instant tables or indexes
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	3
MAX	128
Default value	3

Description

This property increases the parallel property of operation allocating or deleting an instant block.

10.134 INST_HASH_TABLE_BUCKET_MAX_COUNT

Basic Information

Item	Description
Name	INST_HASH_TABLE_BUCKET_MAX_COUNT
Summary	instant hash table bucket max count
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	DEFERRED
MIN	0
MAX	4294967295
Default value	0

Description

It sets the maximum expected bucket counts of the hash instant table.

10.135 INST_TABLE_BLOCK_SIZE

Basic Information

Item	Description
Name	INST_TABLE_BLOCK_SIZE
Summary	a block size of instant tables or indexes
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	8192
MAX	1048576
Default value	16384

Description

It determines the size of an instant block. If the anchor area of an instant record is bigger than an instant block, then the following error occurs.

```
gSQL> SELECT DISTINCT * FROM T1, T1, T1, T1, T1, T1, T1, T1, T1, T1;
ERR-HY000(14098): maximum record length(16360) exceeds
```

10.136 IPC_CHANNEL_COUNT

Basic Information

Item	Description
Name	IPC_CHANNEL_COUNT
Summary	IPC Channel Count
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	0
MAX	2048
Default value	0

Description

It sets the number of channels for IPC communication.

10.137 JOURNAL_TEMP_DIR

Basic Information

Item	Description
Name	JOURNAL_TEMP_DIR
Summary	journaling temporary directory
Data type	VARCHAR
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	DEFERRED
MIN	N/A
MAX	N/A
Default value	$GOLDILOCKS_DATA/journal

Description

It is the temporary directory of journaling.

10.138 KEEPALIVE_IDLE_TIME

Basic Information

Item	Description
Name	KEEPALIVE_IDLE_TIME
Summary	tcp keepalive idle time for checking dead client session (sec)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	DEFERRED
MIN	1
MAX	16383
Default value	300

Description

It means the idle duration between the server and client without tcp packet exchange before sending kee p alive packet. If there is not tcp packet exchange for seconds (KEEPALIVE_IDLE_TIME), keep alive mecha nism starts execution to detect the dead connection on the server side.

10.139 LOCAL_CLUSTER_MEMBER

Basic Information

Item	Description
Name	LOCAL_CLUSTER_MEMBER
Summary	local cluster member name
Data type	VARCHAR
Applicable phase	NONE
Updatable	FALSE
ALTER SESSION	FALSE
ALTER SYSTEM	NONE
MIN	N/A
MAX	N/A
Default value	'G1N1'

Description

It is the local cluster member name.

10.140 LOCAL_CLUSTER_MEMBER_HOST

Basic Information

Item	Description
Name	LOCAL_CLUSTER_MEMBER_HOST
Summary	host name of local cluster member
Data type	VARCHAR
Applicable phase	NONE
Updatable	FALSE
ALTER SESSION	FALSE
ALTER SYSTEM	NONE
MIN	N/A
MAX	N/A
Default value	'127.0.0.1'

Description

It is host name of local cluster member.

10.141 LOCAL_CLUSTER_MEMBER_PORT

Basic Information

Item	Description
Name	LOCAL_CLUSTER_MEMBER_PORT
Summary	listen port of local cluster member
Data type	BIGINT
Applicable phase	NONE
Updatable	FALSE
ALTER SESSION	FALSE
ALTER SYSTEM	NONE
MIN	1024
MAX	49151
Default value	10101

Description

It is listen port of local cluster member.

10.142 LOCAL_JOURNAL_BUFFER_SIZE

Basic Information

Item	Description
Name	LOCAL_JOURNAL_BUFFER_SIZE
Summary	local journal buffer size
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	1024
MAX	10737418240 (10 Giga)
Default value	65536

Description

It is the local journal buffer size.

10.143 LOCATION_FILE

Basic Information

Item	Description
Name	LOCATION_FILE
Summary	location file name
Data type	VARCHAR
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	N/A
MAX	N/A
Default value	$GOLDILOCKS_DATA/wal/location.ctl

Description

It is the location file name.

10.144 LOCATOR_QUERY_TIMEOUT

Basic Information

Item	Description
Name	LOCATOR_QUERY_TIMEOUT
Summary	timeout for waiting locator response (sec)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	10000000
Default value	3

Description

It sets the time (second) waiting for the response after the cluster system enquires of a locator about the solution of split-brain situation. This property is used only when CLUSTER_SPLIT_BRAIN_RESOLUTION_POLICY is set to 1 or more.

10.145 LOCK_HASH_TABLE_SIZE

Basic Information

Item	Description
Name	LOCK_HASH_TABLE_SIZE
Summary	lock manager hash table size (The number of buckets)
Data type	BIGINT
Applicable phase	MOUNT or below
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	2
MAX	1000000
Default value	65519

Description

It specifies the maximum hash table size managed by a lock manager.

10.146 LOCKABLE_DISPATCHER_CM_BUFFER_COUNT

Basic Information

Item	Description
Name	LOCKABLE_DISPATCHER_CM_BUFFER_COUNT
Summary	communication buffer count for lockable dispatcher
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	1
MAX	128
Default value	1

Description

It specifies the number of cluster lockable dispatcher's communication buffers.

10.147 LOCKLESS_DISPATCHER_CM_BUFFER_COUNT

Basic Information

Item	Description
Name	LOCKLESS_DISPATCHER_CM_BUFFER_COUNT
Summary	communication buffer count for lockless dispatcher
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	1
MAX	128
Default value	1

Description

It specifies the number of cluster lockless dispatcher's communication buffers.

10.148 LOG_BLOCK_SIZE

Basic Information

Item	Description
Name	LOG_BLOCK_SIZE
Summary	log block size (byte)
Data type	BIGINT
Applicable phase	NONE
Updatable	FALSE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	512
MAX	4096
Default value	512

Description

It means the minimum size of what log buffer is flushed to the log file of the disk. Its value should be set t
o one of 512, 1024, 2048, 4096.

10.149 LOG_BUFFER_SIZE

Basic Information

Item	Description
Name	LOG_BUFFER_SIZE
Summary	default log buffer size (byte)
Data type	BIGINT
Applicable phase	NO MOUNT or below
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	DEFERRED
MIN	1048576
MAX	10737418240
Default value	10485760

Description

A log buffer is the shared memory space in which the redo logs generated in database by the DML/DDL operations are stored. LOG_BUFFER_SIZE is referenced to set the memory size for the log buffer.

10.150 LOG_DIR

Basic Information

Item	Description
Name	LOG_DIR
Summary	default log direcotry
Data type	VARCHAR
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	DEFERRED
MIN	N/A
MAX	N/A
Default value	$GOLDILOCKS_DATA/wal

Description

The log recorded in the log buffer is flushed to the logfile which exists in a non-volatile storage device to ensure the database durability. LOG_DIR sets the path to the log file.

10.151 LOG_FILE_SIZE

Basic Information

Item	Description
Name	LOG_FILE_SIZE
Summary	log file size (byte)
Data type	BIGINT
Applicable phase	NONE
Updatable	FALSE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	20 Mbytes
MAX	120 Gbytes
Default value	100 Mbytes

Description

It sets the size of the logfile used in database. It is referenced only when creating the database, then log file size can not be updated after then.

10.152 LOG_GROUP_COUNT

Basic Information

Item	Description
Name	LOG_GROUP_COUNT
Summary	initial count of log group
Data type	BIGINT
Applicable phase	NONE
Updatable	FALSE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	4
MAX	254
Default value	4

Description

It sets the number of log group used in database. It is referenced only when creating the database, but af ter that, it does not affect any operations. After creating database, the operation to add or remove a log group is supported by a separate syntax.

10.153 LOG_MIRROR_MODE

Basic Information

Item	Description
Name	LOG_MIRROR_MODE
Summary	LogMirror Mode (1:Enable, 0:Disable)
Data type	BOOLEAN
Applicable phase	MOUNT or below
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	NO

Description

It is the property to configure the required shared memory when operating LogMirror, the redo log replic ation tool, at database startup.
It should be enabled to execute the LogMirror.
The size of the shared Memory can be changed using LOG_MIRROR_SHARED_MEMORY_STATIC_SIZE.

10.154 LOG_MIRROR_SHARED_MEMORY_STATIC_SIZE

Basic Information

Item	Description
Name	LOG_MIRROR_SHARED_MEMORY_STATIC_SIZE
Summary	shared memory size for LogMirror (byte)
Data type	BIGINT
Applicable phase	MOUNT or below
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	10485760 (10 M)
MAX	1073741824 (1 G)
Default value	104857600 (100 M)

Description

It sets the size of the shared memory used in LogMirror, the redo log replication tool.
It is applied in the state which LOG_MIRROR_MODE is enabled.

10.155 LOG_MIRROR_TIMEOUT

Basic Information

Item	Description
Name	LOG_MIRROR_TIMEOUT
Summary	logmirror retry timeout (sec)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	100000000
Default value	0

Description

It is the response waiting time of the LogMirror.
If its value is o, it waits indefinitely. Otherwise, it waits as long as the value set, then TIMEOUT occurs, an d it stops LogMirror service. Later, the server is operated normally.
It is applied in the state which LOG_MIRROR_MODE is enabled.

10.156 LOG_SYNC_INTERVAL

Basic Information

Item	Description
Name	LOG_SYNC_INTERVAL
Summary	interval for synchronize log (s)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	1
MAX	10000
Default value	3

Description

Log flusher of GOLDILOCKS is a system thread which flushes the log buffer contents to disk logfile. When log flusher wakes up in the idle phase, it checks if log to flush exists. Then it flushes the log if any.
If the log flusher did not flush within the time set in LOG_SYNC_INTERVAL, it synchronizes the log buffer and the log file by performing a flush until the last block of the current log buffer.

10.157 LOG_SYNC_INTERVAL_MSEC

Basic Information

Item	Description
Name	LOG_SYNC_INTERVAL_MSEC
Summary	milli-second interval for synchronize log
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	10000000
Default value	0

Description

It is the millisecond interval for synchronize log.

10.158 MAX_GROUP_COUNT

Basic Information

Item	Description
Name	MAX_GROUP_COUNT
Summary	maximum group count
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	0
MAX	8192
Default value	32

Description

It is the maximum group count in the cluster system.

10.159 MAX_JOURNAL_FILE_SIZE

Basic Information

Item	Description
Name	MAX_JOURNAL_FILE_SIZE
Summary	maximum journal file size
Data type	BIGINT
Applicable phase	MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1099511627776 (1 Terabytes)
Default value	0 (no limit)

Description

It sets the maximum size (quota) of the global journaling file which internally stores journaling data when a journaling occurs in cluster system.

10.160 MAX_NODE_COUNT

Basic Information

Item	Description
Name	MAX_NODE_COUNT
Summary	maximum node count
Data type	BIGINT
Applicable phase	NONE
Updatable	FALSE
ALTER SESSION	FALSE
ALTER SYSTEM	NONE
MIN	1
MAX	8192
Default value	64

Description

It is the maximum node (instance) count which can join the cluster system.

10.161 MAXIMUM_CONCURRENT_ACTIVITIES

Basic Information

Item	Description
Name	MAXIMUM_CONCURRENT_ACTIVITIES
Summary	maximum number of active statements that the driver can support for a connection
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	DEFERRED
MIN	1
MAX	65535
Default value	1024

Description

It sets the number of statements which can be executed simultaneously.

10.162 MAXIMUM_FILE_CACHE_SIZE

Basic Information

Item	Description
Name	MAXIMUM_FILE_CACHE_SIZE
Summary	the limit of file descriptor cache
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	16
MAX	32768
Default value	128

Description

It sets the maximum number of file caches which is being used in the session.

10.163 MAXIMUM_FLUSH_BUFFER_PAGE_COUNT

Basic Information

Item	Description
Name	MAXIMUM_FLUSH_BUFFER_PAGE_COUNT
Summary	maximum number of buffer page count to be flushing
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	1
MAX	8192
Default value	64

Description

It sets the maximum number of pages which can be recorded per disk writing operation. If the pages in the disk tablespace are updated in the buffer, then IO thread records them on the disk. If recording nearby pages together when performing disk writing operation, then it increases the efficiency of the system resource by decreasing the number of disk recording.

10.164 MAXIMUM_FLUSH_LOG_BLOCK_COUNT

Basic Information

Item	Description
Name	MAXIMUM_FLUSH_LOG_BLOCK_COUNT
Summary	maximum number of log block count to be flushing
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	1000
MAX	2000000
Default value	100000

Description

When flushing the contents of the log buffer to disk log file, it sets the maximum number of log blocks to be flushed with a single writing operation.

10.165 MAXIMUM_FLUSH_PAGE_COUNT

Basic Information

Item	Description
Name	MAXIMUM_FLUSH_PAGE_COUNT
Summary	maximum number of page count to be flushing
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	1
MAX	8192
Default value	1024

Description

GOLDILOCKS datafiles are flushed to the disk by the checkpoint and certain DDL statements. For flushing datafiles, it sets the maximum number of data pages to be flushed with a single writing operation.

10.166 MAXIMUM_INDEX_REBUILD_JOURNAL_REPLAY_COUNT

Basic Information

Item	Description
Name	MAXIMUM_INDEX_REBUILD_JOURNAL_REPLAY_COUNT
Summary	maximum number of replaying journals for rebuild index
Data type	BIGINT
Applicable phase	NO_MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	2
MAX	1024
Default value	2

Description

When rebuilding the index on ONLINE mode, it can be performed together with DML, and DML records the updates on the journal log. The index is rebuilt based on the data at the time of beginning of the rebuilding, then the updated data during the rebuilding is applied to the index through the journal log. The journal logs are initially applied, then journal logs which were accumulated while applying the journal logs are applied. This property sets how may times the journal logs are applied in this way.

10.167 MAXIMUM_JOURNAL_REPLAY_COUNT

Basic Information

Item	Description
Name	MAXIMUM_JOURNAL_REPLAY_COUNT
Summary	maximum number of replaying journals for rebalance table
Data type	BIGINT
Applicable phase	NO_MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	2
MAX	1024
Default value	2

Description

The table rebalancing online can be performed together with DML in the cluster environment, and DML r ecords the updates on the journal log at that moment. The table rebalancing initially applies the journal l ogs which occurred during synchronizing tables, then applies journal logs which were accumulated while applying the journal logs. This property sets how may times the journal logs are applied in this way.

10.168 MAXIMUM_NAMED_CURSOR_COUNT

Basic Information

Item	Description
Name	MAXIMUM_NAMED_CURSOR_COUNT
Summary	maximum number of named cursor that the driver can support for a connection
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	DEFERRED
MIN	1
MAX	100000
Default value	128

Description

It is the maximum number of named cursor which can be used within a single session.
A named curor is created in the following cases.

- When a named cursor is declared using functions like SQLSetCursorName(), SQLGetCursorName().

```
{
    ...
    SQLSetCursorName( stmt,
                      "my_cursor",
                      SQL_NTS );
    ...
}
```

- When the DECLARE cursor syntax is used by a function such as SQLExecDirect (), SQLPrepare ().

```
{
    ...
    SQLExecDirect( stmt,
                   "DECLARE my_cursor CURSOR FOR SELECT col_name FROM tab_name",
```

```
                    SQL_NTS );
    ...
}
```

- When DECLARE cursor FOR UPDATE syntax is used in an embedded SQL.

```
{
    ...
    EXEC SQL DECLARE my_cursor CURSOR FOR SELECT col_name FROM tab_name FOR UPDATE;
    ...
    EXEC SQL OPEN my_cursor;
    ...
    EXEC SQL FETCH my_cursor INTO :data;
    EXEC SQL DELETE FROM tab_name WHERE CURRENT OF my_cursor;
    ...
    EXEC SQL CLOSE my_cursor;
}
```

In an embedded SQL, DECLARE CURSOR syntax without FOR UPDATE as follows does not creat e a named cursor on the session.

```
{
    ...
    EXEC SQL DECLARE my_cursor CURSOR FOR SELECT col_name FROM tab_name;
    ...
}
```

10.169 MAXIMUM_PACKAGE_INSTANCE_COUNT

Basic Information

Item	Description
Name	MAXIMUM_PACKAGE_INSTANCE_COUNT
Summary	maximum number of package instance that the driver can support for a connection
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	DEFERRED
MIN	1
MAX	100000
Default value	128

Description

It is the maximum number of package instances which are available in a single session.
A package instance is created when using the stateful package in the session.

10.170 MAXIMUM_SESSION_CM_BUFFER_SIZE

Basic Information

Item	Description
Name	MAXIMUM_SESSION_CM_BUFFER_SIZE
Summary	maximum communication bytes per shared mode session
Data type	BIGINT
Applicable phase	MOUNT or below
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	1048576
MAX	1073741824
Default value	20971520

Description

It sets the maximum buffer size available in a single session which is connected to shared mode.
For more information, refer to **DISPATCHER_CM_BUFFER_SIZE**.

10.171 MEASURE_CLUSTER_LATENCY

Basic Information

Item	Description
Name	MEASURE_CLUSTER_LATENCY
Summary	measure cluster latency
Data type	BOOL
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	NO

Description

It is the measure cluster latency.

10.172 MIN_SAMPLE_ROW_COUNT

Basic Information

Item	Description
Name	MIN_SAMPLE_ROW_COUNT
Summary	minimum sampling row count for analyze table
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	DEFERRED
MIN	0
MAX	9223372036854775807 (INT64_MAX)
Default value	100000

Description

It is the minimum number of sampling rows when executing **ANALYZE TABLE** by using the sampling.

10.173 MINIMUM_UNDO_PAGE_COUNT

Basic Information

Item	Description
Name	MINIMUM_UNDO_PAGE_COUNT
Summary	minimum undo page count
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	16
MAX	1048576
Default value	16

Description

DML uses the undo page to store the previous image. Undo page is consumed by using a single undo seg ment per DML. If all allocated pages of undo segments are consumed, the page of another undo segmen t can be used. MINIMUM UNDO PAGE_COUNT is the minimum number of undo page to specify the und o segment to import page when undo pages are insufficient. If the undo pages are insufficient, the pages can be imported only from the undo segment having more pages than MINIMUM UNDO PAGE_COUNT.

10.174 NET_BUFFER_SIZE

Basic Information

Item	Description
Name	NET_BUFFER_SIZE
Summary	TCP network buffer size (byte)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	DEFERRED
MIN	1024
MAX	1073741824
Default value	32768

Description

It sets the TCP communications buffer size.
In the dedicated mode, it is set to the maximum communication packet size.
In the shared mode, it is set to **DISPATCHER_CM_UNIT_SIZE**.

10.175 NLS_DATE_FORMAT

Basic Information

Item	Description
Name	NLS_DATE_FORMAT
Summary	nls date format
Data type	VARCHAR
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	FALSE
MIN	N/A
MAX	N/A
Default value	YYYY-MM-DD

Description

NLS_DATE_FORMAT specifies the default date format of TO_CHAR and TO_DATE functions.

10.176 NLS_TIME_FORMAT

Basic Information

Item	Description
Name	NLS_TIME_FORMAT
Summary	nls time format
Data type	VARCHAR
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	FALSE
MIN	N/A
MAX	N/A
Default value	HH24:MI:SS.FF6

Description

NLS_DATE_FORMAT specifies the default time format of TO_CHAR and TO_DATE functions.

10.177 NLS_TIME_WITH_TIME_ZONE_FORMAT

Basic Information

Item	Description
Name	NLS_TIME_WITH_TIME_ZONE_FORMAT
Summary	nls time with time zone format
Data type	VARCHAR
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	FALSE
MIN	N/A
MAX	N/A
Default value	HH24:MI:SS.FF6 TZH:TZM

Description

NLS_TIME_WITH_TIME_ZONE FORMAT specifies the default time with time zone format of TO_CHAR an d TO_TIME_WITH_TIME_ZONE functions.

10.178 NLS_TIMESTAMP_FORMAT

Basic Information

Item	Description
Name	NLS_TIMESTAMP_FORMAT
Summary	nls timestamp format
Data type	VARCHAR
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	FALSE
MIN	N/A
MAX	N/A
Default value	YYYY-MM-DD HH24:MI:SS.FF6

Description

NLS_TIMESTAMP_FORMAT specifies the default timestamp format of TO_CHAR and TO_TIMESTAMP fun
ctions.

10.179 NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT

Basic Information

Item	Description
Name	NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT
Summary	nls timestamp with time zone format
Data type	VARCHAR
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	FALSE
MIN	N/A
MAX	N/A
Default value	YYYY-MM-DD HH24:MI:SS.FF6 TZH:TZM

Description

NLS_TIMESTAMP_WITH_TIME_ZONE FORMAT specifies the default timestamp with time zone format of TO_CHAR and TO_TIMESTAMP WITH TIMEZONE functions.

10.180 NUMA

Basic Information

Item	Description
Name	NUMA
Summary	enable numa
Data type	BOOLEAN
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	0
MAX	1
Default value	NO

Description

It enables NUMA.

To use the NUMA property in AIX, the user account should be modified. Execute the following command as a root user.

```
# chuser "capabilities=CAP_NUMA_ATTACH,CAP_PROPAGATE" <username>
```

<username> is not a root byt it is a user account of AIX.
Logout then login again to apply the modifications.

10.181 NUMA_MAP

Basic Information

Item	Description
Name	NUMA_MAP
Summary	numa node map
Data type	VARCHAR
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	N/A
MAX	N/A
Default value	'x' : no binding

Description

It sets the map to connect cores of the system to NUMA node. This property is operated when NUMA pro perty is set to on.

The following is an example of the system having four cores.

- Connect core 0 and 1 to number 0 NUMA node, and core 2 and 3 to number 1 NUMA node.

```
NUMA_MAP = '0:0:1:1' # core
```

- Connect core 0 and 1 to number 0 NUMA node, and core 2 and 3 to number 1 NUMA node, and cor e 1 and 3 to number 2 NUMA node.

```
NUMA_MAP = '0:0,2:1:1,2' # core
```

10.182 OFFLINE_MEMBER_AFTER_FAILOVER

Basic Information

Item	Description
Name	OFFLINE_MEMBER_AFTER_FAILOVER
Summary	Automatically offline member after failover
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	YES

Description

The background process automatically takes the errored member offline after completing the failover cau sed by the node error.
If it is not possible to take the errored member offline because it is set to *NO*, then execute the following syntax before the errored member joins the system again.

```
gSQL> ALTER DATABASE OFFLINE INACTIVE CLUSTER MEMBERS;
Database altered.
```

10.183 ONLINE_INDEX_REBUILD_JOURNAL_REPLAY_THRESHOLD

Basic Information

Item	Description
Name	ONLINE_INDEX_REBUILD_JOURNAL_REPLAY_THRESHOLD
Summary	threshold bytes for replaying journals without table lock
Data type	BIGINT
Applicable phase	NO_MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	10737418240
Default value	1048576

Description

DML performed during rebuilding the index on ONLINE mode records the journal log. The journals are applied to the index multiple times when finishing rebuilding the index. **MAXIMUM_INDEX_REBUILD_JOURNAL_REPLAY_COUNT** sets how many times to apply the journal logs. However, if the amount of journal logs to be applied are small, then it is not repeated as many as it is set to be, but instantly set the table the EXCLUSIVE lock, and uses it as the threshold value to apply the last journal log.

10.184 ONLINE_JOURNAL_REPLAY_THRESHOLD

Basic Information

Item	Description
Name	ONLINE_JOURNAL_REPLAY_THRESHOLD
Summary	threshold bytes for replaying journals without table lock
Data type	BIGINT
Applicable phase	NO_MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	10737418240 (10G)
Default value	1048576 (1M)

Description

The table rebalancing online applies journal logs several times which were recorded by dml occurred duri ng the performance in the cluster environment. MAXIMUM_JOURNAL_REPLAY_COUNT sets how many ti mes to apply the journal logs. However, if the amount of journal logs to be applied are small, then it is no t repeated as many as it is set to be, but instantly set the table the EXCLUSIVE lock, and uses it as the thre shold value to apply the last journal log.

10.185 OS_GROUP_ACCESS

Basic Information

Item	Description
Name	OS_GROUP_ACCESS
Summary	enable access database with OS group permission
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	DEFERRED
MIN	0
MAX	1
Default value	NO

Description

To connect to DA with another user of the same group, this property should be set to *YES*. Also, the uma sk of the system should be modified to *0002*.

10.186 PACKET_COMPRESSION_THRESHOLD

Basic Information

Item	Description
Name	PACKET_COMPRESSION_THRESHOLD
Summary	The size limit at which packets are compressed(bytes)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	32
MAX	2113929216
Default value	2113929216

Description

If the data size to be sent to the client is bigger than PACKET_COMPRESSION_THRESHOLD, it compresses the communication data.

10.187 PAGE_CHECKSUM_TYPE

Basic Information

Item	Description
Name	PAGE_CHECKSUM_TYPE
Summary	page checksum type (0:LSN, 1:CRC)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	0

Description

A checksum is used to guarantee the physical consistency for each page of the datafile. GOLDILOCKS sup ports a page checksum of LSN, CRC scheme.

- 0: LSN
- 1: CRC

10.188 PARALLEL_IO_FACTOR

Basic Information

Item	Information
Name	PARALLEL_IO_FACTOR
Summary	parallel load factor
Data type	BIGINT
Applicable phase	NO MOUNT or below
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	DEFERRED
MIN	1
MAX	16
Default value	1

Description

It sets the number of threads for the parallel loading of data file when starting database and the number of threads for parallel recording of data file at checkpoint.

10.189 PARALLEL_IO_GROUP_1 ~ PARALLEL_IO_GROUP_16

Basic Information

Item	Description
Name	PARALLEL_IO_GROUP_1
Summary	parallel load group 1
Data type	VARCHAR
Applicable phase	NO MOUNT or below
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	DEFERRED
MIN	N/A
MAX	N/A
Default value	$GOLDILOCKS_DATA/db

Description

It sets the group directory for parallel I/O of data file. It sets the number of group as many as PARALLEL_IO_FACTOR, then parallel I/O is performed in data file unit which belongs to each group.

10.190 PARALLEL_LOAD_FACTOR

Basic Information

Item	Description
Name	PARALLEL_LOAD_FACTOR
Summary	parallel load factor
Data type	BIGINT
Applicable phase	NO MOUNT or below
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	1
MAX	64
Default value	1

Description

When starting database, it sets the number of threads for parallel operation after loading the memory of a data file.

10.191 PENDING_LOG_BUFFER_COUNT

Basic Information

Item	Description
Name	PENDING_LOG_BUFFER_COUNT
Summary	default pending log buffer count
Data type	BIGINT
Applicable phase	NO MOUNT or below
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	DEFERRED
MIN	0
MAX	32
Default value	4

Description

When multiple transactions are simultaneously running, the pending log buffer is used to reduce the com petition for the log buffer. PENDING LOG_BUFFER COUNT sets the number of pending log buffer which can be used simultaneously.

10.192 PLAN_CACHE

Basic Information

Item	Description
Name	PLAN_CACHE
Summary	caching sql plan
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	DEFERRED
MIN	0
MAX	1
Default value	YES

Description

It determines whether to use the plan cache.

10.193 PLAN_CACHE_SIZE

Basic Information

Item	Description
Name	PLAN_CACHE_SIZE
Summary	sql plan cache size (byte)
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	20971520
MAX	1099511627776
Default value	104857600

Description

It sets the memory size to be used for the plan cache.

10.194 PLAN_HISTORY

Basic Information

Item	Description
Name	PLAN_HISTORY
Summary	plan history for SQLs
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	NO

Description

It sets whether to use the plan history.

10.195 PLAN_HISTORY_SIZE

Basic Information

Item	Description
Name	PLAN_HISTORY_SIZE
Summary	plan history size
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	DEFERRED
MIN	0
MAX	100000
Default value	0

Description

It sets the number of plans to be stored in the plan history.

10.196 PRIVATE_STATIC_AREA_INIT_SIZE

Basic Information

Item	Description
Name	PRIVATE_STATIC_AREA_INIT_SIZE
Summary	Initial size of Private Static Area (byte)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	1048576
MAX	34359738368
Default value	10485760

Description

It sets the initial size of the heap memory to use in the session. Even when there are memories which are not used in the session, the memories are not returned to operating system.

10.197 PRIVATE_STATIC_AREA_NEXT_SIZE

Basic Information

Item	Description
Name	PRIVATE_STATIC_AREA_NEXT_SIZE
Summary	Next size of Private Static Area (byte)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	1024
MAX	34359738368
Default value	10485760

Description

It sets the size of memory to extend when the session allocates additional heap memory.

10.198 PRIVATE_STATIC_AREA_SHRINK_THRESHOLD

Basic Information

Item	Description
Name	PRIVATE_STATIC_AREA_SHRINK_THRESHOLD
Summary	Threshold bytes to attempt to shrink private static area(byte)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	1048576
MAX	34359738368
Default value	10485760

Description

The memory size is preserved as big as this property even when there are unused heap memories in the session, and those memories are not returned to the system but are reused in the session.

Even when it is set to smaller than PRIVATE_STATIC_AREA_INIT_SIZE, it is not decreased to smaller than the size.

10.199 PRIVATE_STATIC_AREA_SIZE

Basic Information

Item	Description
Name	PRIVATE_STATIC_AREA_SIZE
Summary	Shared Static Area Size (byte)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	104857600
MAX	34359738368
Default value	104857600

Description

It specifies the maximum heap memory size to be allocated by the session.

10.200 PROCESS_MAX_COUNT

Basic Information

Item	Description
Name	PROCESS_MAX_COUNT
Summary	Process Max Count
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	12
MAX	65535
Default value	128

Description

It specifies the maximum number of processes (threads) available on the system.

Creating system process

- The process is created each time of connection to D/A mode or C/S dedicated mode.
- In C/S shared mode, processes are basic balancer, dispatcher and shared-server. A process is not created when connecting from client.

10.201 QUERY_TIMEOUT

Basic Information

Item	Description
Name	QUERY_TIMEOUT
Summary	query timeout (s)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	10000000
Default value	0

Description

It specifies the maximum time which a command received from the session can be executed. If the execution time exceeds, the TIMEOUT error occurs.

- 0: It means infinite waiting, and TIMEOUT error does not occur.

10.202 READABLE_ARCHIVELOG_DIR_COUNT

Basic Information

Item	Description
Name	READABLE_ARCHIVELOG_DIR_COUNT
Summary	readable archive log directory count
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	DEFERRED
MIN	1
MAX	10
Default value	1

Description

It sets the number of directories in which archive redo logs exist when executing media recovery.

10.203 READABLE_BACKUP_DIR_COUNT

Basic Information

Item	Description
Name	READABLE_BACKUP_DIR_COUNT
Summary	readable backup directory count
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	DEFERRED
MIN	1
MAX	10
Default value	1

Description

It sets the number of directories in which incremental backups exist when restoring files using incrementa l backups.

10.204 REBALANCE_BLOCK_READ_COUNT

Basic Information

Item	Description
Name	REBALANCE_BLOCK_READ_COUNT
Summary	block read count for rebalance
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	DEFERRED
MIN	1
MAX	65536
Default value	1

Description

It is the block read count for rebalance.

10.205 REBALANCE_SHARD_DIVISOR

Basic Information

Item	Description
Name	REBALANCE_SHARD_DIVISOR
Summary	partition factor of shard upon rebalance
Data type	BIGINT
Applicable phase	NO_MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	1
MAX	1000
Default value	1

Description

It sets the number of shards to divide which are used to rebalance and synchronize the table.
For more information, refer to **ALTER TABLE name REBALANCE**.

10.206 RECOMPILE_CHECK_MINIMUM_PAGE_COUNT


It is not supported after 3.1.

Basic Information

Item	Description
Name	RECOMPILE_CHECK_MINIMUM_PAGE_COUNT
Summary	minimum page count for recompile check
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	1
MAX	10000
Default value	64

Description

It sets the minimum page count to check if the plan is recompiled due to the page count modification.

10.207 RECOMPILE_PAGE_PERCENT


It is not supported after 3.1.

Basic Information

Item	Description
Name	RECOMPILE_PAGE_PERCENT
Summary	recompile page percent
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1000
Default value	30

Description

It sets the page percent when recompiles the plan due to the page count modification. If its value is 0, it does not recompile due to the page count modification.

10.208 RECOVERY_LOG_BUFFER_SIZE

Basic Information

Item	Description
Name	RECOVERY_LOG_BUFFER_SIZE
Summary	default log buffer size for recovery
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	786432
MAX	32 Mega
Default value	10 Mega

Description

It is the default log buffer size for recovery

10.209 RECYCLEBIN

Basic Information

Item	Description
Name	RECYCLEBIN
Summary	enable or disable recyclebin feature
Data type	BOOLEAN
Applicable phase	NO_MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	DEFERRED
MIN	0
MAX	1
Default value	NO

Description

It sets whether to activate the recyclebin feature.

10.210 REDO_LOG_COMPRESSION_THRESHOLD

Basic Information

Item	Description
Name	REDO_LOG_COMPRESSION_THRESHOLD
Summary	The size limit at which redo log are compressed(bytes)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	32
MAX	2113929216
Default value	256

Description

If the size of the created REDO LOG is bigger than REDO_LOG_COMPRESSION_THRESHOLD value, it compresses REDO LOG.

10.211 REFINE_RELATION

Basic Information

Item	Description
Name	REFINE_RELATION
Summary	refine aged relations
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	YES

Description

If this property is set to NO, then REFINE RELATION process is not performed when restarting the server.

This property can be used when an error occurs during the REFINE RELATION process. However, segments of RELATIONs (tables or indexes) which were dropped but not REFINEd can not be reused. When resolving the error then setting this property to YES and restarting, it tries to REFINE Relations which were not dropped.

10.212 SESSION_FATAL_BEHAVIOR

Basic Information

Item	Description
Name	SESSION_FATAL_BEHAVIOR
Summary	session fatal behavior
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	0

Description

When session fatal occurs, it determines whether to terminate only the thread which caused the fatal or to terminate the process.

- 0: It terminates only the thread which caused fatal.
- 1: It terminates the process.
  If multiple sessions are simultaneously performed in the process, the process is terminated after all sessions finish using database.

10.213 SESSION_MEMORY_INIT_SIZE

Basic Information

Item	Description
Name	SESSION_MEMORY_INIT_SIZE
Summary	initial memory size for session
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	131072 (128K)
MAX	1073741824 (1G)
Default value	131072 (128K)

Description

It sets the shared memory size to be allocated in advance so that it can be used in the session.

10.214 SESSION_MEMORY_SHRINK_THRESHOLD

Basic Information

Item	Description
Name	SESSION_MEMORY_SHRINK_THRESHOLD
Summary	threshold bytes to attempt to shrink session memory allocator (byte)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	131072 (128K)
MAX	1073741824 (1G)
Default value	131072 (128K)

Description

It sets the threshold value to determine whether to return the dynamic shared memory which is not used by the session to the system when releasing the dynamic shared memory used in the session. In other wo rds, if the memory chunk which is bigger than the set value among unused memory exists, then it is retur ned to the system.

10.215 SESSION_POOL_INIT_SIZE

Basic Information

Item	Description
Name	SESSION_POOL_INIT_SIZE
Summary	initial memory size for session pool
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	0
MAX	1099511627776 (1T)
Default value	0

Description

It sets the initial memory size of the session pool.

If each session needs a memory, then the space is allocated from the session pool. If the space in the sessi on pool is insufficient, then the space is allocated from SSA.
The session pool is used to prevent the session from frequently accessing to SSA, and if it is set to "0", the n the session pool feature is inactivated.

10.216 SESSION_POOL_NEXT_SIZE

Basic Information

Item	Description
Name	SESSION_POOL_NEXT_SIZE
Summary	memory size to be expanded in session pool
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	131072 (128K)
MAX	1073741824 (1G)
Default value	1048576 (1M)

Description

It sets how much to extend the memory size in the session pool when expanding the session pool space.
It is valid only when SESSION_POOL_INIT_SIZE is bigger than 0.

10.217 SHARED_MEMORY_ADDRESS

Basic Information

Item	Description
Name	SHARED_MEMORY_ADDRESS
Summary	shared memory address
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	N/A
MAX	N/A
Default value	1610612736

Description

It specifies the address of Shared Static Area (SSA).

10.218 SHARED_MEMORY_STATIC_KEY

Basic Information

Item	Description
Name	SHARED_MEMORY_STATIC_KEY
Summary	Shared Memory Static KEY
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	N/A
MAX	N/A
Default value	542353

Description

When running server, it specifies the shared memory key values which are used to allocate Static Shared Area (SSA) space.

10.219 SHARED_MEMORY_STATIC_NAME

Basic Information

Item	Description
Name	SHARED_MEMORY_STATIC_NAME
Summary	Shared Memory Static Name
Data type	VARCHAR
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	N/A
MAX	N/A
Default value	_STATIC

Description

When running server, it specifies the shared memory name which is used to allocate Static Shared Area (SA) space.

10.220 SHARED_MEMORY_STATIC_SIZE

Basic Information

Item	Description
Name	SHARED_MEMORY_STATIC_SIZE
Summary	Shared Memory Static Size (byte)
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	104857600
MAX	1099511627776
Default value	629145600

Description

It specifies the size of the Shared Static Area (SSA).

10.221 SHARED_REQUEST_QUEUE_COUNT

Basic Information

Item	Description
Name	SHARED_REQUEST_QUEUE_COUNT
Summary	count of global request queue
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	1
MAX	16
Default value	1

Description

In shared mode, it sets the number of queues of which the dispatcher requests to the shared-server. A queue is used when multiple dispatchers allocate user's requests to the shared-server. Generally, a single queue is used for the load-balance.
However, SHARED_REQUEST_QUEUE_COUNT value is increased because if the number of dispatcher and shared-server increase, then a conflict to the queue causes performance degradation.
If the the value becomes bigger, the load-balance can be inefficient and the possibility of deadlock increases.

10.222 SHARED_SERVERS

Basic Information

Item	Description
Name	SHARED_SERVERS
Summary	number of shared-server processes
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	1
MAX	2048
Default value	10

Description

It sets the number of shared-server processes on shared mode.
At open phase, the value can not be decreased by using alter system.

10.223 SHARED_SESSION

Basic Information

Item	Description
Name	SHARED_SESSION
Summary	to enable shared session
Data type	BOOL
Applicable phase	MOUNT or below
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	YES

Description

It sets whether to activate shared mode. If the value is set to *NO*, load-balancer (gbalancer), dispatcher (g dispatcher), shared-server (gserver) are not executed.

10.224 SNAPSHOT_STATEMENT_TIMEOUT

Basic Information

Item	Description
Name	SNAPSHOT_STATEMENT_TIMEOUT
Summary	snapshot statement timeout (s)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	1
MAX	22118400 (1 year)
Default value	22118400 (1 year)

Description

It sets the maximum holding time of the statement required for the snapshot read. TIMEOUT error occurs for a snapshot statement which exceeds the time.

10.225 SQL_HISTORY_SIZE

Basic Information

Item	Description
Name	SQL_HISTORY_SIZE
Summary	history size for SQLs
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	DEFERRED
MIN	0
MAX	100000
Default value	0

Description

It is the history size for SQLs.

10.226 SQL_HISTORY_TYPE

Basic Information

Item	Description
Name	SQL_HISTORY_TYPE
Summary	history type for SQLs
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	2
Default value	0

Description

It is the history type for SQLs.

- 0: Direct-execute
- 1: Prepare-execute
- 2: All

10.227 SUPPLEMENTAL_LOG_DATA_PRIMARY_KEY

Basic Information

Item	Description
Name	SUPPLEMENTAL_LOG_DATA_PRIMARY_KEY
Summary	supplemental log data of primary key columns be logged in redo log files
Data type	BOOLEAN
Applicable phase	MOUNT or below
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	NO

Description

It records supplemental log for all changes in the database.

10.228 SYNC_DISPATCHER_CM_BUFFER_COUNT

Basic Information

Item	Description
Name	SYNC_DISPATCHER_CM_BUFFER_COUNT
Summary	communication buffer count for synchronization dispatcher
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	1
MAX	128
Default value	1

Description

It specifies the number of cluster synchronization dispatcher's communication buffers.

10.229 SYSTEM_DISK_DATA_TABLESPACE_SIZE

Basic Information

Item	Description
Name	SYSTEM_DISK_DATA_TABLESPACE_SIZE
Summary	default system disk data tablespace size(byte)
Data type	BIGINT
Applicable phase	NONE
Updatable	FALSE
ALTER SESSION	FALSE
ALTER SYSTEM	NONE
MIN	32 Mega
MAX	30 Giga
Default value	200 Mega

Description

It sets the initial DISK_DATA_TBS tablespace size when creating the database.

10.230 SYSTEM_FILE_IO

Basic Information

Item	Description
Name	SYSTEM_FILE_IO
Summary	i/o type for system file (0: direct io, 1: buffered io)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	0

Description

It sets IO type of when using the database file except for the data file and the log file.

10.231 SYSTEM_MEMORY_AUX_TABLESPACE_SIZE

Basic Information

Item	Description
Name	SYSTEM_MEMORY_AUX_TABLESPACE_SIZE
Summary	default system memory auxiliary tablespace size (byte)
Data type	BIGINT
Applicable phase	NONE
Updatable	FALSE
ALTER SESSION	FALSE
ALTER SYSTEM	NONE
MIN	32 Mega
MAX	30 Giga
Default value	200 Mega

Description

It determines the size of initial MEM_AUX_TBS tablespace when creating the database.

10.232 SYSTEM_MEMORY_DATA_TABLESPACE_SIZE

Basic Information

Item	Description
Name	SYSTEM_MEMORY_DATA_TABLESPACE_SIZE
Summary	default system memory data tablespace size (byte)
Data type	BIGINT
Applicable phase	NONE
Updatable	FALSE
ALTER SESSION	FALSE
ALTER SYSTEM	NONE
MIN	32 Mega
MAX	30 Giga
Default value	200 Mega

Description

It determines the initial tablespace size of MEM_DATA_TBS when creating database.

10.233 SYSTEM_MEMORY_DICT_TABLESPACE_SIZE

Basic Information

Item	Description
Name	SYSTEM_MEMORY_DICT_TABLESPACE_SIZE
Summary	default dictionary tablespace size (byte)
Data type	BIGINT
Applicable phase	NONE
Updatable	FALSE
ALTER SESSION	FALSE
ALTER SYSTEM	NONE
MIN	256 Mega
MAX	30 Giga
Default value	256 Mega

Description

It determines the initial tablespace size of DICTIONARY_TBS when creating database.

10.234 SYSTEM_MEMORY_TEMP_TABLESPACE_SIZE

Basic Information

Item	Description
Name	SYSTEM_MEMORY_TEMP_TABLESPACE_SIZE
Summary	default system memory temporary tablespace size (byte)
Data type	BIGINT
Applicable phase	NONE
Updatable	FALSE
ALTER SESSION	FALSE
ALTER SYSTEM	NONE
MIN	32 Mega
MAX	30 Giga
Default value	200 Mega

Description

It determines the initial tablespace size of MEM_TEMP_TBS when creating database.

10.235 SYSTEM_MEMORY_UNDO_TABLESPACE_SIZE

Basic Information

Item	Description
Name	SYSTEM_MEMORY_UNDO_TABLESPACE_SIZE
Summary	default system memory undo tablespace size (byte)
Data type	BIGINT
Applicable phase	NONE
Updatable	FALSE
ALTER SESSION	FALSE
ALTER SYSTEM	NONE
MIN	32 Mega
MAX	30 Giga
Default value	32 Mega

Description

It determines the initial tablespace size of MEM_UNDO_TBS when creating database.

10.236 SYSTEM_TABLESPACE_DIR

Basic Information

Item	Description
Name	SYSTEM_TABLESPACE_DIR
Summary	system tablespace directory
Data type	VARCHAR
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	DEFERRED
MIN	N/A
MAX	N/A
Default value	$GOLDILOCKS_DATA/db

Description

It sets the path to which the initial system tablespaces are stored when creating database.

10.237 SYSTEM_UDS_DIR

Basic Information

Item	Description
Name	SYSTEM_UDS_DIR
Summary	system unix domain socket directory
Data type	VARCHAR
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	N/A
MAX	N/A
Default value	'/tmp'

Description

It sets a directory on which the unix domain socket file is created.
Setting the directory for the unix domain socket except for DB system, such asglsnr, is managed by a sepa rate configuration file.
The maximum setting value is 60 bytes. (The maximum size of the absolute path (directory + file name) f or the unix domain socket file varies according to OS, but generally it is around 100 bytes.)

10.238 TCP_CLIENT_NUMA_NODE

Basic Information

Item	Description
Name	TCP_CLIENT_NUMA_NODE
Summary	numa node for TCP clients
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	-1
MAX	63
Default value	-1

Description

It sets the NUMA node ID to which the client server session is bound. This property is operated when NU MA property is set to on.

10.239 TCP_NODELAY

Basic Information

Item	Description
Name	TCP_NODELAY
Summary	no delays in buffer flushing within the TCP/IP protocol stack
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	DEFERRED
MIN	0
MAX	1
Default value	YES

Description

It sets TCP_NODELAY option of the socket when transferring the data to a client in C/S method (TCP sock et).
Set it to *NO* when fast latency is not required and reducing the network load is needed.

10.240 TEMP_SEGMENT_CACHE_SIZE

Basic Information

Item	Description
Name	TEMP_SEGMENT_CACHE_SIZE
Summary	the number of segments to be cached for global temporary tables and indexes in each session
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	DEFERRED
MIN	0
MAX	4294967295
Default value	3

Description

It sets the number of segments to catch in a session instead of returning to a tablespace when dropping a global temporary table or a global temporary index segment. Segments in a segment cach is reused later in a global temporary tableor a global temporary index.

- 0: It does not use a segment cache of a global temporary table or of a global temporary index in a session.
- 1 ~ 4294967295: It keeps the specific number of segment caches of a global temporary table or of a global temporary index in a session.

10.241 TEMP_UNDO_ENABLED

Basic Information

Item	Description
Name	TEMP_UNDO_ENABLED
Summary	enables writing undo records of global temporary tables to the temp tablespace
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	0

Description

It defines the location of logging undo records for a global temporary table.

- 0 (FALSE): It records the undo records in the default undo tablespace of database.
- 1 (TRUE): It records the undo records in the default temporary tablespace of database.

10.242 TIMED_STATISTICS

Basic Information

Item	Description
Name	TIMED_STATISTICS
Summary	timed statistics
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	2
Default value	0

Description

It is whether to check the wait event.
To record the statistics related to wait event on v$system_event, v$session_event and v$session_wait tabl e, set this property.

- 0: It does not record the statistics.
- 1: It records the statistics.
- 2: It records the statistics by using the high precision timer.

10.243 TIMER_INTERVAL

Basic Information

Item	Description
Name	TIMER_INTERVAL
Summary	timer interval time(us)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	10
MAX	100000
Default value	10000

Description

It sets the time interval which is required when the timer thread sets the system time.

10.244 TIMEZONE

Basic Information

Item	Description
Name	TIMEZONE
Summary	timezone
Data type	VARCHAR
Applicable phase	NONE
Updatable	FALSE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	N/A
MAX	N/A
Default value	+09:00

Description

It is a time zone value of database.
It is applied when creating database, and it uses the value of the range from '-14:00' to '+14:00'.

10.245 TRACE_ALTER_SYSTEM

Basic Information

Item	Description
Name	TRACE_ALTER_SYSTEM
Summary	write trace messages forALTER SYSTEM
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	YES

Description

It records the SQL statements in trace file ($GOLDILOCKS_DATA/trc/system.trc) when executing ALTER SYSTEM syntax.

Set TRACE_ALTER_SYSTEM property to *ON* to record system changes.

SELECT inquiry, and execution of INSERT, UPDATE, DELETE syntax have nothing to do with TRACE_ALTER_SYSTEM property, so they do not affect the performance of TRACE_ALTER_SYSTEM.

10.246 TRACE_DDL

Basic Information

Item	Description
Name	TRACE_DDL
Summary	write trace messages for DDL
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	YES

Description

When executing DDL, it records the executed SQL statements in trace file ($GOLDILOCKS_DATA/trc/system.trc).

Set *TRACE_ALTER_SYSTEM* property to *ON* to record SQL statements execution such as CREATE/DROP/ALTER table.

TRACE_DDL property affects only to DDL statements. However, it has nothing to do with SELECT inquiry, and execution of INSERT, UPDATE, DELETE syntax. Therefore, it does not affect the performance.

10.247 TRACE_LOG_ID

Basic Information

Item	Description
Name	TRACE_LOG_ID
Summary	trace log ID
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	100000000
Default value	0

Description

The execution plan for the query, and other related information are recorded in the trace file (opt_p[process ID_s [session ID].trc) under the trace directory ($GOLDILOCKS_DATA/trc/) when processing queries. To record SQL statement for the query, the execution plan and the execution time, refer to the following flag information.

Table 10-1 Flag information for TRACE_LOG_ID

Information	Flag(on)	Flag(off)
Whether to output the PSM call flow (procedure/function)	1000000	0
Whether to output the successful SQL query	100000	0
Whether to output the failed SQL query	10000	0
Whether to output the execution plan	1000	0
Whether to output the execution type (direct/prepare)	100	0
Whether to output the bind value	10	0
Whether to output the execution time per section	1	0

To set it in a form of "output the successful SQL query" + "output the execution plan" + "output the bind value", set the TRACE_LOG_ID value to 101010.

10.248 TRACE_LOG_MSGBUF_SIZE

Basic Information

Item	Description
Name	TRACE_LOG_MSGBUF_SIZE
Summary	memory buffer size for trace log message
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	8192
MAX	10485760
Default value	24576

Description

It sets the size of the heap memory buffer which is used to configure the log message to be recorded in t he trace logfile.

10.249 TRACE_LOG_TIME_DETAIL

Basic Information

Item	Description
Name	TRACE_LOG_TIME_DETAIL
Summary	detail trace log time
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	NO

Description

It sets whether to increase the time accuracy when recording trace log.
If the value is ON, it has an accuracy of 1 us.
If the value is OFF, it has an accuracy of 10 ms.

10.250 TRACE_LOGGER

Basic Information

Item	Description
Name	TRACE_LOGGER
Summary	trace log type (1:file, 2:file & remote)
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	1
MAX	2
Default value	1

Description

It sets the target on which the trace log is written.
If it is 1, then it is recorded in a file, and if it is 2, then it is remotely recorded in a file.
When it is remotely written, then it remotely collects trace logs from gtrclogger and records in a file.

10.251 TRACE_LOGGER_REMOTE_HOST

Basic Information

Item	Description
Name	TRACE_LOGGER_REMOTE_HOST
Summary	remote host for trace logger
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	255255255255
Default value	127000000001

Description

It sets the host to which the trace log is remotely transferred by setting TRACE_LOGGER to 2.

10.252 TRACE_LOGGER_REMOTE_PORT

Basic Information

Item	Description
Name	TRACE_LOGGER_REMOTE_PORT
Summary	remote port for trace logger
Data type	BIGINT
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	1024
MAX	49151
Default value	21470

Description

It sets the port to which the trace log is remotely transferred by setting TRACE_LOGGER to 2.

10.253 TRACE_LOGIN

Basic Information

Item	Description
Name	TRACE_LOGIN
Summary	write login trace messages for user
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	NO

Description

It records the related access information in trace file ($GOLDILOCKS_DATA/trc/login.trc) on login.
Set TRACE_LOGIN property to *ON* to record the related information on login.

10.254 TRACE_LONG_RUN_CURSOR

Basic Information

Item	Description
Name	TRACE_LONG_RUN_CURSOR
Summary	write trace SQL for cursor life-time over specific time (mili-sec. 0 ~ 10000000)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	10000000
Default value	0

Description

When cursor life-time is longer than the specified property time, then it records the SQL statement of the cursor in the trace file ($GOLDILOCKS_DATA/trc/system.trc).

- Description of value
  - Unit: millisecond
  - Value 0: It does not record information.
  - Recommended value: 20 (millisecond) or longer
  - The value of 20 or longer is recommended because the execution time is measured using the tim e tic in 10 ms period.
  - Use **TRACE_LONG_RUN_TIMER** property to increase the precision.

- The following is an example of recording the SQL statement whose cursor life-time is longer than 1 s econd.

```
gSQL> ALTER SYSTEM SET TRACE_LONG_RUN_CURSOR = 1000;
```

- The following is an example of restoring to the default value.

```
gSQL> ALTER SYSTEM SET TRACE_LONG_RUN_CURSOR TO DEFAULT;
```

It is used to trace the user program maintaining the cursor for a long time as follows.

```
int main()
{
   ...
   EXEC SQL DECLARE cur1 CURSOR FOR SELECT name FROM t1 WHERE pk = :s_id;
   EXEC SQL OPEN cur1
   EXEC SQL FETCH cur1 INTO :s_name;
   ...
   long_run_user_logic( s_name ); ❶ Due to the user logic, ager fails to clean up resources
for a long time.
   ...
   EXEC SQL CLOSE cur1;
   ...
}
```

10.255 TRACE_LONG_RUN_SQL

Basic Information

Item	Description
Name	TRACE_LONG_RUN_SQL
Summary	write trace for long-run SQL over specific execution time (mili-sec. 0 ~ 10000000)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	10000000
Default value	0

Description

It records the SQL statement whose execution time is longer than the specified property time in the trace file ($GOLDILOCKS_DATA/trc/system.trc).

- Description of value
  - Unit: millisecond
  - Value 0: It does not record information.
  - Recommended value: 20 (millisecond) or longer
  - The value of 20 or longer is recommended because the execution time is measured using the tim e tic in 10 ms period.
  - Use **TRACE_LONG_RUN_TIMER** property to increase the precision.

The following is an example of recording the SQL statement whose execution time is longer than 1 secon d.

```
gSQL> ALTER SYSTEM SET TRACE_LONG_RUN_SQL = 1000;
```

The following is an example of restoring to the default value.

```
gSQL> ALTER SYSTEM SET TRACE_LONG_RUN_SQL TO DEFAULT;
```

10.256 TRACE_LONG_RUN_TIMER

Basic Information

Item	Description
Name	TRACE_LONG_RUN_TIMER
Summary	trace long-run timer resolution (0: timer thread(10 ms interval), 1: gettimeofday())
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	0

Description

It controls the measurement precision when measuring the execution time of the SQL statement by using the following properties.

- **TRACE_LONG_RUN_CURSOR**
- **TRACE_LONG_RUN_SQL**

- Value
  - 0: It uses the timer thread whose interval is 10 milliseconds.
  - 1: It measures the time by using gettimeofday() function. In this case, the precision is higher but t he system call causes the work load.

10.257 TRACE_SYSTEM_DIR

Basic Information

Item	Description
Name	TRACE_SYSTEM_DIR
Summary	system logger directory
Data type	VARCHAR
Applicable phase	NONE
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	FALSE
MIN	N/A
MAX	N/A
Default value	$GOLDILOCKS_DATA/trc

Description

It sets the disk path where the trace log message is recorded.

ALIAS

Item	Description
Original name	TRACE_SYSTEM_DIR
ALIAS	SYSTEM_LOGGER_DIR

10.258 TRACE_XA

Basic Information

Item	Description
Name	TRACE_XA
Summary	logging trace log for xa interfaces
Data type	BOOLEAN
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	NO

Description

It specifies whether to output trace messages when using XA interface. Message is output to the 'SYSTEM_LOGGER_DIR / xa.trc'.

10.259 TRANSACTION_ALLOCATION_TIMEOUT

Basic Information

Item	Description
Name	TRANSACTION_ALLOCATION_TIMEOUT
Summary	a time limit (sec) for allocating transaction slot
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	100000000
Default value	3

Description

It is the maximum waiting time when allocating transaction slots.

The following error occurs when the waiting time exceeds TRANSACTION_ALLOCATION_TIMEOUT.

```
gSQL> insert into t1 values(1);
ERR-HYT00(14129): transaction allocation time exceeded
```

10.260 TRANSACTION_COMMIT_WRITE_MODE

Basic Information

Item	Description
Name	TRANSACTION_COMMIT_WRITE_MODE
Summary	transaction commit write mode (0: no_wait, 1: wait)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	1
Default value	0

Description

TRANSACTION_COMMIT_WRITE_MODE specifies whether a log generated by the transaction is flushed to the disk log file, when the transaction is committed. If TRANSACTION_COMMIT_WRITE_MODE is '1', the log should be flushed to the disk log file at the time of the transaction commit. Otherwise the transaction is committed regardless of log flush.

If the system is operated when TRANSACTION_COMMIT_WRITE_MODE is set to '0', the latest data will be lost when GOLDILOCKS is abnormally terminated without log flush after COMMIT transaction. It is because the logs are not recorded in this case.

Therefore, if all committed transactions should be remained (stored) in database, the system should be operated after setting TRANSACTION_COMMIT_WRITE_MODE to '1'. Or, 'ALTER SYSTEM FLUSH LOGS' statement should be explicitly performed at the time of transaction commit in order to flush log after TRANSACTION_COMMIT_WRITE_MODE is set to '0'.

- 0: no wait
- 1: wait

10.261 TRANSACTION_MAXIMUM_UNDO_PAGE_COUNT

Basic Information

Item	Description
Name	TRANSACTION_MAXIMUM_UNDO_PAGE_COUNT
Summary	The maximum number of undo pages that a transaction can write.
Data type	BIGINT
Applicable phase	NO MOUNT or below
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	1
MAX	13107200
Default value	13107200

Description

It means the maximum number of undo pages which the transaction can record. The minimum value is 1 (8 Kbytes) and the maximum value is 13107200 (100 Gbytes).

10.262 TRANSACTION_TABLE_SIZE

Basic Information

Item	Description
Name	TRANSACTION_TABLE_SIZE
Summary	transaction table size
Data type	BIGINT
Applicable phase	NO MOUNT or below
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	64
MAX	10240
Default value	1024

Description

It sets the maximum number of transaction tables which are executed in database. The database should r estart to modify the number of transaction tables, and the number can be modified as long as the numbe r is bigger than the previously specified number. However, if it is modified to the smaller number, then th e restart fails when it is same or smaller than the maximum value of the transaction slot identifier used by the transactions prepared after the restart recovery.

For example, if the value set as 1,024 is modified to 512 and the maximum value of the transaction slot i dentifier used by the transactions prepared at the restart is also 512, then the restart fails as follows. In th is case, modify it to the number bigger than 512, then the restart succeeds.

```
gSQL> ALTER SYSTEM SET TRANSACTION_TABLE_SIZE = 512 SCOPE = FILE;
System altered.
gSQL> \CONNECT sys gliese as sysdba
gSQL> \SHUTDOWN
Shutdown success
```

- The restart fails.

```
gSQL> \STARTUP
ERR-HY000(14118): TRANSACTION_TABLE_SIZE property value must be equal to or greater than '513'
gSQL> ALTER SYSTEM SET TRANSACTION_TABLE_SIZE = 513 SCOPE = FILE;
System altered.
gSQL> \SHUTDOWN
Shutdown success
gSQL> \STARTUP
Startup success
```

TRANSACTION_TABLE_SIZE of all cluster members should be same in cluster environment, so it is required to restart all cluster members to modify TRANSACTION_TABLE_SIZE.

10.263 TRANSACTION_TIMEOUT

Basic Information

Item	Description
Name	TRANSACTION_TIMEOUT
Summary	transaction timeout (s)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	10000000
Default value	0

Description

It sets the duration of when the transaction is activated. It is used to prevent the side effects of when the transaction is activated for a long time. If a transaction exceeds the specified time, then gmaster daemon automatically terminates the session owned by that transaction.

10.264 UNDO_RELATION_ALLOCATION_TIMEOUT

Basic Information

Item	Description
Name	UNDO_RELATION_ALLOCATION_TIMEOUT
Summary	a time limit (sec) for allocating undo relation
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	100000000
Default value	3

Description

It is the maximum waiting time when allocating undo relations.

The following error occurs when the waiting time exceeds UNDO_RELATION_ALLOCATION_TIMEOUT.

```
gSQL> insert into t1 values(1);
ERR-HYT00(14130): undo relation allocation time exceeded
```

10.265 UNDO_RELATION_COUNT

Basic Information

Item	Description
Name	UNDO_RELATION_COUNT
Summary	undo relation count
Data type	BIGINT
Applicable phase	NO MOUNT or below
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	8
MAX	10240
Default value	128

Description

It sets the number of undo relation to be used in database. Undo relation is allocated for the purpose tha t the transaction executing DML uses undo segment. The database should restart to modify the number of undo relations, and the number can be modified only to the number bigger than the previously specifi ed number.

If it is modified to a smaller number, then the restart fails. For example, if the value set as 128 is modified to 64, then the restart fails as follows. In this case, modify it to 128 or bigger, then the restart succeeds.

```
gSQL> ALTER SYSTEM SET UNDO_RELATION_COUNT = 64 SCOPE = FILE;
System altered.
gSQL> \CONNECT sys gliese as sysdba
gSQL> \SHUTDOWN
Shutdown success
```

- The restart fails.

```
gSQL> \STARTUP
ERR-HY000(14119): UNDO_RELATION_COUNT property value must be equal to or greater than '128'
gSQL> ALTER SYSTEM SET UNDO_RELATION_COUNT = 128 SCOPE = FILE;
```

```
System altered.
gSQL> \SHUTDOWN
Shutdown success
gSQL> \STARTUP
Startup success
```

10.266 UNDO_SHRINK_THRESHOLD

Basic Information

Item	Description
Name	UNDO_SHRINK_THRESHOLD
Summary	threshold bytes to attempt to shrink undo segment (byte)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	1048576
MAX	107374182400
Default value	10485760

Description

Ager thread periodically (10 seconds) checks the undo segment space. If it occupies more space than this property value, then the reusable space is returned to the tablespace. The attempt to return is made until the undo segment space remains as big as this property (byte), and the return is finished when the amou nt of the remaining undo page becomes smaller than MINIMUM_UNDO_PAGE_COUNT.

10.267 USE_LARGE_PAGES

Basic Information

Item	Description
Name	USE_LARGE_PAGES
Summary	use large pages
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0
MAX	2
Default value	0

Description

It uses HugePage. HugePage should first be set in the device to use USE_LARGE_PAGES property.

- 0: It does not use the large page.
- 1: It uses the large page. When it fails to allocate the shared memory, then an error occurs.
- 2: It tries to allocate the shared memory by using the large page. When it fails to allocate the shared memory, then it allocates the memory by using the regular page.

It can be used in Linux kernel 2.6.32-573 or higher.

10.268 USER_DATA_TABLESPACE_MEDIA_TYPE

Basic Information

Item	Description
Name	USER_DATA_TABLESPACE_MEDIA_TYPE
Summary	default media type of user data tablespace
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	0 (Memory)
MAX	1 (Disk)
Default value	0 (Memory)

Description

It sets the default media type if the media type of the tablespace is omitted when creating the user data t ablespace. 0 is memory and 1 is the disk.

10.269 USER_DATA_TABLESPACE_SIZE

Basic Information

Item	Description
Name	USER_DATA_TABLESPACE_SIZE
Summary	default user data tablespace size(byte)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	32 Mega
MAX	30 Giga
Default value	32 Mega

Description

It sets the default size if the data file size is omitted when creating the user data tablespace or adding the data file.

10.270 USER_DISK_DATA_TABLESPACE_NEXTSIZE

Basic Information

Item	Description
Name	USER_DISK_DATA_TABLESPACE_NEXTSIZE
Summary	default next size of user data tablespace
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	3554432 Byte
MAX	30 Giga
Default value	10 Mega

Description

It sets the default size if the size to be extended is not set when it is required to extend the data file of th e user disk data tablespace.

10.271 USER_TEMP_TABLESPACE_SIZE

Basic Information

Item	Description
Name	USER_TEMP_TABLESPACE_SIZE
Summary	default user temp tablespace size(byte)
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	TRUE
ALTER SYSTEM	IMMEDIATE
MIN	32 Mega
MAX	30 Giga
Default value	32 Mega

Description

It sets the default size if the data file size is omitted when creating the user temp tablespace or adding th e data file.

10.272 XA_TRANSACTION_IDLE_TIMEOUT

Basic Information

Item	Description
Name	XA_TRANSACTION_IDLE_TIMEOUT
Summary	idle timeout for xa transaction
Data type	BIGINT
Applicable phase	NO MOUNT or above
Updatable	TRUE
ALTER SESSION	FALSE
ALTER SYSTEM	IMMEDIATE
MIN	0 (no limit)
MAX	10000000
Default value	60

Description

It is the maximum waiting time of xa transaction in idle (The duration between the beginning of XA and t he next transaction). If it remains in Idle exceeding this time, then xa transaction is rolled back.

If it is set to 0, then XA infinitely waits even in idle.

Part III.

SQL Manual

11.

SQL Elements

11.1 Syntax Elements

Identifiers

Identifier is divided into an ordinary identifier and a delimited identifier.
An ordinary identifier consists of a letter or letters and numbers and it is used by internally substituting all characters to uppercase letters. Therefore, an ordinary identifier is not case-sensitive.

The following is an example of an ordinary identifier.

```
GOLDILOCKS
GoldiLocks
```

A delimited identifier consists of a letter or letters and numbers and they are enclosed with double quotes ("). All the characters are used literally as internally described. Therefore, a delimited identifier is case-sens itive.

The following is an example of a delimited identifier.

```
"GOLDILOCKS"
"GoldiLocks"
```

Literals

Literals mean representation of non-null value.

Text Literals

Text literals mean representation of strings and binary strings.

Use single quote (') at the beginning and end of a string to write string of text literals.
Not only the double quotes (") string but also all strings except for the single quote (') string can be writt en within a single quote (').
Use a single quote twice without white spaces in between to write a single quote (') in a string.
A maximum of 4000 characters can be written in a string.

The followings are examples of text literals for a string.

```
'GOLDILOCKS'
'Sunje''s DBMS'
```

A binary string of text literals is a string of hexadecimal numbers which starts with x'(X') and ends with '. Only the characters corresponding to 0 ~ 9, A (a) ~ F (f) can be written in each position of a hexadecimal string. The length of a hexadecimal string should always be a even number because its two digits mean o ne byte. A maximum of 4,000 characters can be written in a binary string.

The followings are examples of text literals for a binary string.

```
x'001f'
X'FF0A'
x'aF37BBc013'
```

Numeric Literals

Numeric literals mean literals of numeric type, and integers or the number with decimal point can be writ ten. The syntax for numeric literals is as follows.

```
[ + ¦ - ] <digits> [ . <digits> ] [ E ¦ e [ + ¦ - ] <digits> ] [ f ¦ F ¦ d ¦ D ]
```

- The first +/- means a positive or negative value of the entire number, and it can be omitted. If omitte d, the default value is a positive value.
- In <digits>, numbers from 0 to 9 can be listed without white space, the number with decimal point ca n be written by using decimal point (.).
- An exponent form can be written after the first <digits>, and E or e is written, and it stands for expon ent. Then +/- is written according to exponent sign, and <digits> is written. +/- of exponent can be o mitted. If omitted, the default value is a positive value.
- Lastly, character such as f, F, d, D can be written after numbers. It means that it is a number of BINAR Y_FLOAT, BINARY_DOUBLE. If the corresponding character is omitted, the number is considered as N UMBER type.

The followings are examples of numeric literals.

```
20
+123.45
0.03
+1.23E-02
-1.5
10f
+123.45F
1.2E-3F
-22d
```

```
123.45D
-1.23E+05D
```

Datetime Literals

Datetime literals are representation of date/time type.
Datetime value is specified using string literal, or by converting character or numeric value to datetime val ue using TO_*function (TO_DATE, etc).

Datetime data types are DATE, TIME, TIME WITH TIME ZONE, TIMESTAMP, TIMESTAMP WITH TIME ZON E.

Date Literals

Date literals are written in a form of DATE'string literal' or TO_DATE(string_literal [, format]).

- DATE'string literal'
  - The format of date type is ' SYYYY-MM-DD'.
  - DATE'2002-07-15'
- TO_DATE(string_literal [, format])
  - If the format is not specified, the format of date type is NLS_DATE_FORMAT by default.
  - If the format is specified, the specified format is applied.

- The date type includes year, month, day, hour, minute, second (except for fractional seconds).
- If the date is omitted, the default value is the first day of the current month.
- If the hour, minute, second are omitted, the default value is midnight.
  - HH24 format: '00:00:00'
  - HH12 format: '12:00:00'
- To set the hour, minute, second to the default value (midnight) when the date value includes hour, minute, second, then use the TRUNC(date) function.
  - For example, in TRUNC(SYSDATE), SYSDATE includes the values of the year, month, day, hour, minute, second.
- To compare only the values of the year, month, day among date values, then set the values of the ho ur, minute, second are set to midnight using the TRUNC function.

For more information, refer to **TO_DATE**, **Datetime Format String**, **NLS_DATE_FORMAT**.

- The following is an example of date literals.

```
DATE'2002-07-15'
```

- The following is an example of when the format is not specified, so NLS_DATE_FORMAT is 'YYYY-M M-DD'.

```
TO_DATE( '2002-07-15' )
```

- The followings are examples of when the format is specified.

```
TO_DATE( '15-JUL-02', 'DD-MON-RR' )
TO_DATE( '2002-07-15 00:00:00', 'YYYY-MM-DD HH24:MI:SS' )
TO_DATE( '2002-07-15 13:25:30', 'YYYY-MM-DD HH24:MI:SS' )
```

- The following is an example of when the date is omitted. (It is set to the first day of the current mont h).

```
gSQL> SELECT TO_DATE( '2000-07', 'YYYY-MM' ) FROM DUAL;
TO_DATE( '2000-07', 'YYYY-MM' )
------------------------------
2000-07-01
```

- The following is an example of when the hour, minute, second are omitted. (It is set to the midnight).

```
gSQL> SELECT
      TO_CHAR( DATE'2002-07-15', 'YYYY-MM-DD HH24:MI:SS' ) AS RESULT
      FROM DUAL;
RESULT
-------------------
2002-07-15 00:00:00
```

- The following is an example of setting the hour, minute, second of DATE value (SYSDATE) to the def ault value (midnight).

```
gSQL> SELECT
      TO_CHAR( SYSDATE,
               'YYYY-MM-DD HH24:MI:SS' ) AS RESULT_SYSDATE,
      TO_CHAR( TRUNC( SYSDATE ),
               'YYYY-MM-DD HH24:MI:SS' ) AS RESULT_TRUNC_SYSDATE
      FROM DUAL;
RESULT_SYSDATE      RESULT_TRUNC_SYSDATE
  ------------------- --------------------
  2014-08-19 10:06:49 2014-08-19 00:00:00
```

- The following is an example of comparing only the values of the year, month, day among date values.

```
gSQL> SELECT
      TO_DATE( '2002-08-12' ) =
```

```
      TRUNC( TO_DATE( '2002-08-12 23:59:59', 'YYYY-MM-DD HH24:MI:SS' ) )
      AS RESULT FROM DUAL;
RESULT
------
TRUE
```

Time Literals

Time literals are written in a form of TIME'string literal' or TO_TIME(string_literal [, format]).

- TIME'string literal'
  - The format of time type is 'HH24:MI:SS[.[FF6]]'.
  - TIME'15:30:59.999999'
- TO_TIME(string_literal [, format])
  - If the format is not specified, the format of time type is NLS_TIME_FORMAT by default.
  - If the format is specified, the specified format is applied.

The time type includes hour, minute, second (fractional seconds).
Fractional seconds can be specified to maximum six digits numbers format.

For more information, refer to **TO_TIME**, **Datetime Format String**, **NLS_TIME_FORMAT**.

- The following is an example of time literals.

```
TIME'15:30:59.999999'
```

- The following is an example of when the format is not specified, so NLS_TIME_FORMAT is 'HH24:MI:SS.FF6'.

```
TO_TIME( '15:30:59.999999' )
```

- The following is an example of when the format is specified.

```
TO_TIME( '09.45.03.546873 AM', 'HH12.MI.SS.FF6 AM' )
TO_TIME( '09:45:03', 'HH12:MI:SS' )
```

Time with Time Zone Literals

Time with time zone literals is written in a form of TIME'string literal', TIME WITH TIME ZONE'string literal' , TO_TIME_WITH_TIME_ZONE(string_literal [, format]), or TO_TIME_TZ(string_literal [, format]).

- TIME'string literal' or TIME WITH TIME ZONE'string literal'
  - The format of time with time zone type is 'HH24:MI:SS[.[FF6]] TZH:TZM'.

    - TIME'15:30:59.999999 +09:00'
    - TIME WITH TIME ZONE'15:30:59.999999 +09:00'
- TO_TIME_WITH_TIME_ZONE(string_literal [, format])
    - If the format is not specified, the format of time with time zone type is NLS_TIME_WITH_TIME_ZONE_FORMAT by default.
    - If the format is specified, the specified format is applied.

Time with time zone type includes hour, minute, second (fractional seconds), and time zone offset (time zone hour, time zone minute).
Fractional seconds can be specified to maximum six digits numbers format.

For more information, refer to **TO_TIME_WITH_TIME_ZONE**, **Datetime Format String**, **NLS_TIME_WITH_TIME_ZONE_FORMAT**.

- The followings are examples of time with time zone literals.

```
TIME'15:30:59.999999 +09:00'
TIME WITH TIME ZONE'15:30:59.999999 +09:00'
```

- The following is an example of when the format is not specified, so NLS_TIME_WITH_TIME_ZONE_FORMAT is 'HH24:MI:SS.FF6 TZH:TZM'.

```
TO_TIME_WITH_TIME_ZONE( '15:30:59.999999 +09:00' )
TO_TIME_TZ( '15:30:59.999999 +09:00' )
```

- The following is an example of when the format is specified.

```
TO_TIME_WITH_TIME_ZONE( '09.45.03.546873 +09:00 AM',
                        'HH12.MI.SS.FF6 TZH:TZM AM' )
```

Timestamp Literals

Timestamp literals are written in a form of TIMESTAMP'string literal' or TO_TIMESTAMP(string_literal [, format]).

- TIMESTAMP'string literal'
    - The format of timestamp type is 'SYYYY-MM-DD HH24:MI:SS[.[FF6]]'.
    - TIMESTAMP'2002-07-15 15:39:59.999999'
- TO_TIMESTAMP(string_literal [, format])
    - If the format is not specified, the format of timestamp type is NLS_TIMESTAMP_FORMAT by default.
    - If the format is specified, the specified format is applied.

Timestamp type includes year, month, day, hour, minute, second (fractional seconds).
Fractional seconds can be specified to maximum six digits numbers format.

For more information, refer to **TO_TIMESTAMP**, **Datetime Format String**, **NLS_TIMESTAMP_FORMAT**.

- The following is an example of timestamp literals.

```
TIMESTAMP'2002-07-15 15:39:59.999999'
```

- The following is an example of when the format is not specified, so NLS_TIMESTAMP_FORMAT is 'YYYY-MM-DD HH24:MI:SS.FF6'.

```
TO_TIMESTAMP( '2002-07-15 15:39:59.999999' )
```

- The following is an example of when the format is specified.

```
TO_TIMESTAMP( '15-JUL-02 11.06.30.123456 AM',
              'DD-MON-RR HH12.MI.SS.FF6 AM' )
```

Timestamp with Time Zone Literals

Timestamp with time zone literals is written in a form of TIMESTAMP'string literal', TIMESTAMP WITH TIME ZONE'string literal', TO_TIMESTAMP_WITH_TIME_ZONE(string_literal [, formt]), or TO_TIMESTAMP_TZ(string_literal [, format]).

- TIMESTAMP'string literal' or TIMESTAMP WITH TIME ZONE'string literal'
  - The format of timestamp with time zone type is 'SYYYY-MM-DD HH24:MI:SS[.[FF6]] TZH:TZM'.
  - TIMESTAMP'2002-07-15 15:39:59.999999 +09:00'
  - TIMESTAMP WITH TIME ZONE'2002-07-15 15:39:59.999999 +09:00'
- TO_TIMESTAMP_WITH_TIME_ZONE(string_literal [, formt])
  - If the format is not specified, the format of timestamp with time zone type is NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT by default.
  - If the format is specified, the specified format is applied.

Timestamp with time zone type includes year, month, day, hour, minute, second (fractional seconds), time zone offset (time zone hour, time zone minute).
Fractional seconds can be specified to maximum six digits numbers format.

For more information, refer to **TO_TIMESTAMP_WITH_TIME_ZONE**, **Datetime Format String** , **NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT**.

- The following is an example of timestamp with time zone literals.

```
TIMESTAMP'2002-07-15 15:39:59.999999 +09:00'
TIMESTAMP WITH TIME ZONE'2002-07-15 15:39:59.999999 +09:00'
```

- The following is an example of when the format is not specified, so NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT is 'YYYY-MM-DD HH24:MI:SS.FF6 TZH:TZM'.

```
TO_TIMESTAMP_WITH_TIME_ZONE( '2002-07-15 15:39:59.999999 +09:00' )
TO_TIMESTAMP_TZ( '2002-07-15 15:39:59.999999 +09:00' )
```

- The following is an example of when the format is specified.

```
TO_TIMESTAMP_WITH_TIME_ZONE( '15-JUL-02 11.06.30.123456 +09:00 AM',
                             'DD-MON-RR HH12.MI.SS.FF6 TZH:TZM AM' )
TO_TIMESTAMP_TZ( '15-JUL-02 11.06.30.123456 +09:00 AM',
                 'DD-MON-RR HH12.MI.SS.FF6 TZH:TZM AM' )
```

Interval Literals

The interval literals specify the time interval.

Intervals are classified and expressed as follows.

- Year-month INTERVAL values
  - It includes YEAR and MONTH.
  - Display string expression: 'year-month'
  - It can be written in a form as *INTERVAL 'string literal' YEAR[leading precision] TO MONTH* or *NUMTOYMINTERVAL(num, interval_indicator)*
- Day-time INTERVAL values
  - It includes DAY, HOUR, MINUTE, SECOND (fractional seconds).
  - Display string expression: 'day hour:minute:second.fractional_seconds'
  - It can be written in a form as *INTERVAL 'string literal' DAY[leading precision] TO SECOND[fractional seconds precision]* or *NUMTODSINTERVAL(num, interval_indicator)*.
- The interval value's sign is specified only once at the beginning of the string representation.
  e.g. INTERVAL'+3 11:22:33.999999'DAY TO SECOND (O)
  INTERVAL'-3 +11:22:33.999999'DAY TO SECOND (X)

The followings are the list of interval types.

- INTERVAL YEAR (leading precision)
- INTERVAL MONTH (leading precision)
- INTERVAL YEAR (leading precision) TO MONTH
- INTERVAL DAY (leading precision)

- INTERVAL HOUR (leading precision)
- INTERVAL MINUTE (leading precision)
- INTERVAL SECOND (leading precision[, fractional seconds precision])
- INTERVAL DAY (leading precision) TO HOUR
- INTERVAL DAY (leading precision) TO MINUTE
- INTERVAL DAY (leading precision) TO SECOND (fractional seconds precision)
- INTERVAL HOUR (leading precision) TO MINUTE
- INTERVAL HOUR (leading precision) TO SECOND (fractional seconds precision)
- INTERVAL MINUTE (leading precision) TO SECOND (fractional seconds precision)

Leading precision
- It is the digit number of the field, it can be specified from 2 to 6. If it is not specified, the default value is set to 2.
- If the leading field value exceeds the leading precision, then an error is returned.

Fractional seconds precision
- It is the digit number of fractional seconds, and it can be specified from 0 to 6. If it is not specified, the default value is set to 6.
- If the fractional second field value exceeds the fractional seconds precision, then it is rounded off.

For more information, refer to **INTERVAL**, **Precisions and value range of the second or later field in INTERVAL * TO * , NUMTODSINTERVAL**, **NUMTOYMINTERVAL**.

Examples of Using Interval Literals.

The followings are examples of using interval literals.

Interval YEAR

The followings are examples of using interval YEAR literals.

Example	Description	Display string
INTERVAL'1'YEAR INTERVAL'01-00'YEAR	1 year	+01-00
INTERVAL'100'YEAR	It exceeds the leading precision 2, so it returns the error.	-
INTERVAL'100'YEAR(3)	100 year	+100-00
INTERVAL'+999999'YEAR(6)	999999 year	+999999-00
INTERVAL'-999999'YEAR(6)	-(999999 year)	-999999-00

Interval MONTH

The followings are examples of using interval MONTH literals.

Example	Description	Display string
INTERVAL'1'MONTH INTERVAL'00-01'MONTH	1 month	+00-01
INTERVAL'100'MONTH	It exceeds the leading precision 2, so it returns the error.	-
INTERVAL'100'MONTH(3)	8 year 4 month	+008-04
INTERVAL'+999999'MONTH(6)	83333 year 3 month	+083333-03
INTERVAL'-999999'MONTH(6)	-(83333 year 3 month)	-083333-03

Interval YEAR TO MONTH

The followings are examples of using interval YEAR TO MONTH literals.

Example	Description	Display string
INTERVAL'1-06'YEAR TO MONTH	1 year 6 month	+01-06
INTERVAL'1-12'YEAR TO MONTH	The month value exceeded 11, so it returns the error.	-
INTERVAL'100-11'YEAR TO MONTH	It exceeds the leading precision 2, so it returns the er ror.	-
INTERVAL'100-11'YEAR(3) TO MONTH	100 year 11 month	+100-11
INTERVAL'+999999-11'YEAR(6) TO MO NTH	999999 year 11 month	+999999-11
INTERVAL'-999999-11'YEAR(6) TO MO NTH	-(999999 year 11 month)	-999999-11

Interval DAY

The followings are examples of using interval DAY literals.

Example	Description	Display string
INTERVAL'1'DAY INTERVAL'01 00:00:00'DAY	1 day	+01 00:00:00
INTERVAL'100'DAY	It exceeds the leading precision 2, so it returns the error.	-
INTERVAL'100'DAY(3)	100 day	+100 00:00:00
INTERVAL'+999999'DAY(6)	999999 day	+999999 00:00:00
INTERVAL'-999999'DAY(6)	-(999999 day)	-999999 00:00:00

Interval HOUR

The followings are examples of using interval HOUR literals.

Example	Description	Display string
INTERVAL'1'HOUR INTERVAL'00 01:00:00'HOUR	1 hour	+00 01:00:00
INTERVAL'1000'HOUR(3)	It exceeds the leading precision 3, so it returns the error	-

Example	Description	Display string
INTERVAL'1000'HOUR(4)	41 day 16 hour	+0041 16:00:00
INTERVAL'+999999'HOUR(6)	41666 day 15 hour	+041666 15:00:00
INTERVAL'-999999'HOUR(6)	-(41666 day 15 hour)	-041666 15:00:00

Interval MINUTE

The following are examples of using interval MINUTE literals.

Example	Description	Display string
INTERVAL'1'MINUTE INTERVAL'00 00:01:00'MINUTE	1 minute	+00 00:01:00
INTERVAL'12345'MINUTE(4)	It exceeds the leading precision 4, so it returns the error	-
INTERVAL'12345'MINUTE(5)	8 day 13 hour 45 minute	+00008 13:45:00
INTERVAL'+999999'MINUTE(6)	694 day 10 hour 39 minute	+000694 10:39:00
INTERVAL'-999999'MINUTE(6)	-(694 day 10 hour 39 minute)	-000694 10:39:00

Interval SECOND

The followings are examples of using interval SECOND literals.

Example	Description	Display string
INTERVAL'1'SECOND INTERVAL'00 00:00:01.000000'SECOND	1 second	+00 00:00:01.000000
INTERVAL'100'SECOND	It exceeds the leading precision 2, so it returns the error.	-
INTERVAL'99.9999999'SECOND INTERVAL'99.9999999'SECOND(2,6)	The fractional seconds are rounded off to become 100 second, then it exceeds the leading precision 2, so it returns the error.	-
INTERVAL'99.9999999'SECOND(3)	1 minute 40 second	+000 00:01:40.000000
INTERVAL'29.506167'SECOND(2, 2)	29.51 second	+00 00:00:29.51
INTERVAL'999999.999999'SECOND(6,6)	11day 13 hour 46 minute 39.999999 second	+000011 13:46:39.999999
INTERVAL'-999999.999999'SECOND(6,6)	-(11day 13 hour 46 minute 39.999999 second)	-000011 13:46:39.999999

Interval DAY TO HOUR

The followings are examples of using interval DAY TO HOUR literals.

Example	Description	Display string
INTERVAL'1 23'DAY TO HOUR		

Example	Description	Display string
INTERVAL'01 23:00:00'DAY TO HOUR	1 day 23 hour	+01 23:00:00
INTERVAL'1 24'DAY TO HOUR	The hour value exceeds 23 (in valid), so it returns the error.	-
INTERVAL'100 23'DAY TO HOUR	It exceeds the leading precisio n 2, so it returns the error.	-
INTERVAL'100 23'DAY(3) TO HOUR	100 day 23 hour	+100 23:00:00
INTERVAL'+999999 23'DAY(6) TO HOUR	999999 day 23 hour	+999999 23:00:00
INTERVAL'-999999 23'DAY(6) TO HOUR	-(999999 day 23 hour)	-999999 23:00:00
INTERVAL'-999999 +23'DAY(6) TO HOUR	Invalid sign error	-

Interval DAY TO MINUTE

The followings are examples of using interval DAY TO MINUTE literals.

Example	Description	Display string
INTERVAL'1 23:59'DAY TO MINUTE INTERVAL'01 23:59:00'DAY TO MINUTE	1 day 23 hour 59 second	+01 23:59:00
INTERVAL'1 24:59'DAY TO MINUTE	The hour value exceeds 23 (in valid), so it returns the error.	-
INTERVAL'1 23:60'DAY TO MINUTE	The minute value exceeds 59 (invalid), so it returns the erro r.	-
INTERVAL'100 23:59'DAY TO MINUTE	It exceeds the leading precisio n 2, so it returns the error.	-
INTERVAL'100 23:59'DAY(3) TO MINUTE	100 day 23 hour 59 minute	+100 23:59:00
INTERVAL'+999999 23:59'DAY(6) TO MINUTE	999999 day 23 hour 59 minut e	+999999 23:59:00
INTERVAL'-999999 23:59'DAY(6) TO MINUTE	-(999999 day 23 hour 59 min ute)	-999999 23:59:00

Interval DAY TO SECOND

The followings are examples of using interval DAY TO SECOND literals.

Example	Description	display string
INTERVAL '1 23:59:59.999999'DAY TO SECOND	1 day 23 hour 59 minute 59.9 99999 second	+01 23:59:59.999999
INTERVAL '1 24:59:59.999999'DAY TO SECOND	The hour value exceeds 23, so it returns the error.	-
INTERVAL '1 23:60:59.999999'DAY TO SECOND	The minute value exceeds 59, so it returns the error.	-
	The second value exceeds 60,	

Example	Description	display string
INTERVAL '1 23:59:60.999999'DAY TO SECOND	so it returns the error.	-
INTERVAL '99 23:59:59.9999999'DAY TO SECOND	The fractional seconds are rounded off to become 100 day, then it exceeds the leading precision 2, so it returns the error.	-
INTERVAL '99 23:59:59.9999999'DAY(3) TO SECOND	100 day	+100 00:00:00.000000
INTERVAL '1 11:22:33.567890'DAY(2) TO SECOND(2)	1 day 11 hour 22 minute 33.57 second	+01 11:22:33.57
INTERVAL '+999999 23:59:59.999999'DAY(6) TO SECOND(6)	999999 day 23 hour 59 minute 59.999999 hour	+999999 23:59:59.999999
INTERVAL '-999999 23:59:59.999999'DAY(6) TO SECOND(6)	-(999999 day 23 hour 59 minute 59.999999 hour)	-999999 23:59:59.999999

Interval HOUR TO MINUTE

The followings are examples of using interval HOUR TO MINUTE literals.

Example	Description	Display string
INTERVAL'23:59'HOUR TO MINUTE INTERVAL'00 23:59:00'HOUR TO MINUTE	23 hour 59 minute	+00 23:59:00
INTERVAL'23:60'HOUR TO MINUTE	The minute value exceeds 59, so it returns the error.	-
INTERVAL'100:59'HOUR TO MINUTE	It exceeds the leading precision 2, so it returns the error.	-
INTERVAL'100:59'HOUR(3) TO MINUTE	4 day 4 hour 59 minute	+004 04:59:00
INTERVAL'+999999:59'HOUR(6) TO MINUTE	41666 day 15 hour 59 minute	+041666 15:59:00
INTERVAL'-999999:59'HOUR(6) TO MINUTE	-(41666 day 15 hour 59 minute)	-041666 15:59:00

Interval HOUR TO SECOND

The followings are examples of using interval HOUR TO SECOND literals.

Example	Description	Display string
INTERVAL '23:59:59.999999'HOUR TO SECOND INTERVAL '00 23:59:59.999999'HOUR TO SECOND	23 hour 59 minute 59.999999 second	+00 23:59:59.999999
INTERVAL '23:60:59.999999'HOUR TO SECOND	The minute value exceeds 59, so it returns the error.	-
INTERVAL '23:59:60.999999'HOUR TO SECOND	The second value exceeds 59,	-

Example	Description	Display string
	so it returns the error.	
INTERVAL '99:59:59.9999999'HOUR TO SECOND	The fractional seconds are rounded off to become 100 hour, then it exceeds the leading precision 2, so it returns the error.	-
INTERVAL '99:59:59.9999999'HOUR(3) TO SECOND	4 day 4 hour	+004 04:00:00.000000
INTERVAL '11:22:29.569'HOUR(3) TO SECOND(1)	11 hour 22 minute 29.6 second	+000 11:22:29.6
INTERVAL '+999999:59:59.999999'HOUR(6) TO SECOND(6)	41666 day 15 hour 59 minute 59.999999 second	+041666 15:59:59.999999
INTERVAL '-999999:59:59.999999'HOUR(6) TO SECOND(6)	-(41666 day 15 hour 59 minute 59.999999 second)	-041666 15:59:59.999999

Interval MINUTE TO SECOND

The followings are examples of using interval MINUTE TO SECOND literals.

Example	Description	Display string
INTERVAL '15:23.123456'MINUTE TO SECOND INTERVAL '00 00:15:23.123456'MINUTE TO SECOND	15 minute 23.123456 second	+00 00:15:23.123456
INTERVAL '15:60.123456'MINUTE TO SECOND	The second value exceeds 59, so it returns the error.	-
INTERVAL '99:59.999999'MINUTE TO SECOND(2)	The fractional seconds are rounded off to become 100 minute, then it exceeds the leading precision 2, so it returns the error.	-
INTERVAL '99:59.999999'MINUTE(3) TO SECOND(2)	1 hour 40 minute	+000 01:40:00.00
INTERVAL '+999999:59.999999'MINUTE(6) TO SECOND(6)	694 day 10 hour 39 minute 59.999999 second	+000694 10:39:59.999999
INTERVAL '-999999:59.999999'MINUTE(6) TO SECOND(6)	-(694 day 10 hour 39 minute 59.999999 second)	-000694 10:39:59.999999

Null Value

Null value is an unknown value or an undefined value. NULL value can be a value of any data type. The u nknown value of the boolean type is represented as null value.
Null value is defined as a keyword and it is not case-sensitive.

The following is an example of null value representation.

```
NULL
Null
```

Comments

Single Line Comment

A single line comment is a comment which starts with -- or //. The single line comment processes a com ment from the behind of the comment's symbol to the end of the line.

The following is an example of using a single line comment.

```
gSQL> SELECT I1, -- I2, I3,
2 I4, I5
3 FROM T1;
I1        I4        I5
--------- --------- ---------
column i1 column i4 column i5
1 row selected.
gSQL> SELECT I1, // I2, I3,
2 I4, I5
3 FROM T1;
I1        I4        I5
--------- --------- ---------
column i1 column i4 column i5
1 row selected.
```

Multiple Line Comment

A multiple line comment is a comment which starts with /* and ends with */. Multiple line comments spe cify a comment from /* to */, and it can use multiple lines to represent comments.

The following is an example of using multiple line comment.

```
gSQL> SELECT I1, I2, I3, I4, I5
2  /* Output
3   all columns of TABLE T1 */
4 FROM T1;
I1        I2        I3        I4        I5
--------- --------- --------- --------- ---------
column i1 column i2 column i3 column i4 column i5
1 row selected.
```

Hint Comment

A hint comment is a comment which starts with /*+ and ends with */. Hint comment is similar to multipl e line comment, but the difference is that the hint comment has + at the beginning.
Do not use a space between * and +. If so, it will be treated as multiple line comment.

Unlike other comments, a hint comment is specified to be used only at the location which is right after th e SELECT keyword. The processing method which a user specified to GOLDILOCKS optimizer is described in the hint comment. For more information, refer to **SQL Hint**.

The following is an example of using hint comment.

```
gSQL> SELECT /*+ FULL(T1) */ * FROM T1;
I1        I2        I3        I4        I5
--------- --------- --------- --------- ---------
column i1 column i2 column i3 column i4 column i5
1 row selected.
```

SQL Reserved Words and Keywords

SQL Reserved Words

GOLDILOCKS supports reserved words which are specified as SQL reserved words. The SQL reserved word s can not be used other than specified location.

SQL reserved words can be used as identifiers by using double quotes ("), but it is not recommended to u se it as follows, because readability decreases.

```
gSQL> CREATE TABLE "SELECT" ( "FROM" INTEGER );
Table created.
gSQL> INSERT INTO "SELECT" ( "FROM" ) VALUES ( 1 );
1 row created.
gSQL> SELECT "FROM" FROM "SELECT";
FROM
----
   1
1 row selected.
```

The followings are SQL reserved words of GOLDILOCKS. * marked SQL reserved words are supported by t he SQL standard.
For more information about the list, refer to **V$RESERVED_WORDS**.

ABSOLUTE
ACCESS
ALL *
ALLOCATE *
ALTER *
AND *
ANY *
ARE *
AS *
ASYMMETRIC *
AT *
AUTHORIZATION *
BEGIN *
BETWEEN *
BOTH *
BY *
CALL *
CASE *
CHECK *
CLOSE *
COLUMN *
COMMENT
COMMIT *
CONNECT *
CONSTRAINT *
CREATE *
CROSS *

CURRENT *
CURRENT_CATALOG *
CURRENT_DATE *
CURRENT_DEFAULT_TRANSFORM_GROUP *
CURRENT_PATH *
CURRENT_ROLE *
CURRENT_ROW *
CURRENT_SCHEMA *
CURRENT_TIME *
CURRENT_TIMESTAMP *
CURRENT_TRANSFORM_GROUP_FOR_TYPE *
CURRENT_USER *
DATABASE
DEALLOCATE *
DECLARE *
DEFAULT *
DELETE *
DEREF *
DESCRIBE *
DETERMINISTIC *
DISCONNECT *
DISTINCT *
DROP *
ELSE *
END *
END_EXEC *
ESCAPE *
EXCEPT *
EXEC *
EXECUTE *
EXISTS *
FALSE *
FETCH *
FILTER *
FIRST
FOR *
FOREIGN *
FREE *
FROM *
FULL *
FUNCTION *

GET *
GLOBAL *
GRANT *
GROUP *
HAVING *
HOLD *
IDENTIFIED
IF
IMMEDIATE
IN *
INDICATOR *
INNER *
INOUT *
INSERT *
INTERSECT *
INTO *
IS *
JOIN *
LAST
LEADING *
LEFT *
LIKE *
LIMIT
LOCAL *
LOCALTIME *
LOCALTIMESTAMP *
MATCH *
MEMBER *
MERGE *
MINUS
NATURAL *
NEW *
NEXT
NOT *
NULL *
OF *
OFFSET *
OLD *
ON *
OPEN *
OR *

ORDER *
OUT *
PREPARE *
PRIMARY *
PRIOR
PROCEDURE *
PROFILE
REF *
REFERENCES *
RELATIVE
RELEASE *
RENAME
RETURN *
RETURNING
RETURNS *
REVOKE *
RIGHT *
ROLLBACK *
ROW *
ROWID
ROWS *
ROW_NUMBER *
SAVEPOINT *
SELECT *
SESSION_USER *
SET *
SOME *
SQL *
SQLEXCEPTION *
SQLSTATE *
SQLWARNING *
START *
SYMMETRIC *
SYNONYM
SYSDATE
SYSTEM *
SYSTEM_USER *
SYSTIME
SYSTIMESTAMP
TABLE *
THEN *

TO *
TRAILING *
TRIGGER *
TRUE *
TRUNCATE *
UNION *
UNIQUE *
UNKNOWN *
UPDATE *
UPPER *
USER *
USING *
VALUES *
VIEW
WHEN *
WHENEVER *
WHERE *
WINDOW *
WITH *
WITHOUT *

SQL Keywords

GOLDILOCKS SQL keywords are not reserved words. However, they are keywords which are internally use d by GOLDILOCKS. Therefore, it is not recommended to use GOLDILOCKS SQL keywords because it can d ecrease the readability of the results.

GOLDILOCKS SQL keywords list can be viewed through **V$KEYWORDS**.

Compatibility for Syntax Elements

The SQL standard compatibility for syntax element is as follows.

Table 11-1 SQL standard compatibility for syntax element

Feature ID	Description	Availability
E021-03	Character literals	O
E131	Null value support (nulls in lieu of values)	O
E161	SQL comments using leading double minus	O
F051-01	DATE data type (including support of DATE literal)	O
	TIME data type (including support of TIME literal) with fractional seconds pr	

Feature ID	Description	Availability
F051-02	ecision of at least 0	O
F051-03	TIMESTAMP data type (including support of TIMESTAMP literal) with fractio nal seconds precision of at least 0 and 6	O
F271	Compound character literals	X
F383	Set column not null clause	O
F391	Long identifiers	X
F392	Unicode escapes in identifiers	X
F393	Unicode escapes in literals	X
T023	Compound binary literals	X
T024	Spaces in binary literals	X
T101	Enhanced nullability determination	X
T351	Bracketed comments	X
T591	UNIQUE constraints of possibly null columns	O
X041	Basic table mapping: null absent	X
X042	Basic table mapping: null as nil	X
X051	Advanced table mapping: null absent	X
X052	Advanced table mapping: null as nil	X
X170	XML null handling options	X
X400	Name and identifier mapping	X

11.2 Data Type

Numeric Type

Numeric data types are classified according to the storage method and the fractional part representation.

- Classification by the storage method
  - Decimal numeric type
    - It stores decimal numbers in 100 digits basis.
    - Types: NUMBER, NUMERIC, FLOAT
  - Binary numeric type
    - It stores numbers of C language as it is.
    - Types: NATIVE_INTEGER, NATIVE_DOUBLE

- Classification by fractional part representation
  - Fixed point data type (exact numeric)
    - It is a numeric type whose scale is fixed.
    - Types: NUMERIC(precision, scale), NATIVE_INTEGER
  - Floating point data type (approximate numeric)
    - It is a numeric type whose scale is not fixed.
    - Types: FLOAT(precision), NATIVE_DOUBLE

Decimal Numeric Type

This type's precision and scale are based on decimal number. The precision indicates accuracy of the valid digits, and the scale indicates the range of fraction.

Decimal Fixed Point Number Type

The decimal fixed point number type is defined in SQL.

Table 11-2 Decimal fixed point number type

Type	Decimal precision	Decimal scale	Refer to
NUMBER(p)	p	0	**NUMBER**
NUMBER(p, s)	p	s	**NUMBER**
NUMERIC(p)	p	0	**NUMERIC**
NUMERIC(p, s)	p	s	**NUMERIC**
DECIMAL(p)	p	0	**NUMERIC** type alias
DECIMAL(p, s)	p	s	**NUMERIC** type alias

Type	Decimal precision	Decimal scale	Refer to
DEC(p)	p	0	**NUMERIC** type alias
DEC(p, s)	p	s	**NUMERIC** type alias
SMALLINT	5	0	**NUMBER** type alias
INTEGER	10	0	**NUMBER** type alias
BIGINT	19	0	**NUMBER** type alias
INT2	5	0	**NUMBER** type alias
INT4	10	0	**NUMBER** type alias
INT8	19	0	**NUMBER** type alias

Decimal Floating Point Number Type

The decimal floating point number type is defined in SQL.

Table 11-3 Decimal floating point number type

Type	Decimal precision	Decimal scale	Refer to
NUMBER	38	N/A	**NUMBER**
FLOAT(p)	ceil($\log_{10} 2^{p}$)	N/A	**FLOAT**
REAL	ceil($\log_{10} 2^{24}$) = 8	N/A	**FLOAT** type alias
DOUBLE	ceil($\log_{10} 2^{53}$) = 16	N/A	**FLOAT** type alias
FLOAT4	ceil($\log_{10} 2^{24}$) = 8	N/A	**FLOAT** type alias
FLOAT8	ceil($\log_{10} 2^{53}$) = 16	N/A	**FLOAT** type alias

Binary Number Type

This type's precision and scale are based on binary number. The precision indicates accuracy of the valid d igits, and the scale indicates the range of fraction.

Binary Fixed Point Number Type

The binary fixed point number type refers to the signed integer data type of C language.
1 bit is used to represent the sign bit, and other bits are used to represent the precision, but not any bit is used to represent the scale.

Table 11-4 Binary fixed point number type

Type	Binary precision	Binary scale	Refer to
NATIVE_SMALLINT	15	0	**NATIVE_SMALLINT**
NATIVE_INTEGER	31	0	**NATIVE_INTEGER**
NATIVE_BIGINT	63	0	**NATIVE_BIGINT**

Binary Floating Point Number Type

The binary floating point type refers to the float and double data type in C language.
1 bit is used to represent the sign bit, and other bits are used to represent the precision and scale.

Table 11-5 Binary floating point number type

Type	Binary precision	Binary scale	Refer to
NATIVE_REAL	23	8	**NATIVE_REAL**
NATIVE_DOUBLE	52	11	**NATIVE_DOUBLE**

The precision and scale of binary floating point type is subject to change depending on the influence of the compiler and OS.

CHARACTER STRING Type

CHARACTER STRING data types are classified according to whether it is a variable length string and the maximum length of string.

- Classification by whether it is a variable length string
  - Fixed length string
    - Refer to **CHARACTER**.
  - Variable length string
    - Refer to **CHARACTER VARYING**, **CHARACTER LONG VARYING**.
- Classification by the maximum length of a string
  - 2000 [characters or bytes]
    - Refer to **CHARACTER**.
  - 4000 [characters or bytes]
    - Refer to **CHARACTER VARYING**.
  - 100 megabytes
    - Refer to **CHARACTER LONG VARYING**.

BINARY STRING Type

BINARY STRING data types are classified according to whether it is a variable length binary string and the maximum length of binary string.

- Classification by whether it is a variable length binary string

    - Fixed length binary string
        - Refer to **BINARY**.
    - Variable length binary string
        - Refer to **BINARY VARYING**, **BINARY LONG VARYING**.
- Classification by the maximum length of binary string
    - 2000
        - Refer to **BINARY**.
    - 4000
        - Refer to **BINARY VARYING**.
    - 100 Mega Bytes
        - Refer to **BINARY LONG VARYING**.

Date/ Time Type

Date/ time data type specifies the year, month, day, hour, minute, second, time zone offset in accordanc e with their representation method.
Date/ time type has **DATE**, **TIME**, **TIMESTAMP** types.

INTERVAL Type

INTERVAL data type specifies the time interval.
It specifies the time interval of the year, month, day, hour, minute, second in accordance with their repres entation method.

INTERVAL data types are classified to the YEAR TO MONTH family type and the DAY TO SECOND family t ype, according to the range of value representation.

BOOLEAN Type

The BOOLEAN data type stores truth value of TRUE, FALSE, UNKNOWN. UNKNOWN value is represented as a null value. All expressions used as condition returns the BOOLEAN value and the column or the value defined as a BOOLEAN data type can be used as a condition.

The following literals can be stored in the boolean data type.

- TRUE
    - Keyword: TRUE

    - Literal: 't', 'true' , 'y', 'yes' , 'on' ,'1'
- FALSE
    - Keyword: FALSE
    - Literal: 'f', 'false', 'n', 'no', 'off', '0'
- UNKNOWN
    - Keyword: UNKNOWN, NULL

For more information, refer to **BOOLEAN**.

ROWID Type

All the records stored in the database has a unique location information. The record identifier (ROWID) is used to distinguish each record.

ROWID data type is used to store and manage the record identifier (ROWID).
Record identifier (ROWID) is obtained by the query using the ROWID pseudo column.

For more information, refer to **ROWID**.

Type Comparison

Comparing two types is executed on the basis of one representative type. If the comparison target type is different from the representative type, then the comparison can go through a type conversion.
The representative types for type comparison defines the representative type for comparing two types.

The following table describes target type conversion for comparison per each representative type.

- **Type conversion for the VC comparison**
- **Type conversion for the LC comparison**
- **Type conversion for the VB comparison**
- **Type conversion for the LB comparison**
- **Type conversion for the NB comparison**
- **Type conversion for the ND comparison**
- **Type conversion for the NU comparison**
- **Type conversion for the DA comparison**
- **Type conversion for the TI comparison**
- **Type conversion for the TZ comparison**
- **Type conversion for the TS comparison**
- **Type conversion for the SZ comparison**

- **Type conversion for the YM comparison**
- **Type conversion for the DS comparison**
- **Type conversion for the BO comparison**
- **Type conversion for the RI comparison**

The followings are abbreviations which are used for the type comparison.

- "VC": CHARACTER VARYING
- "LC": CHARACTER LONG VARYING
- "VB": BINARY VARYING
- "LB": BINARY LONG VARYING
- "NB": NATIVE_BIGINT
- "ND": NATIVE_DOUBLE
- "NU": NUMBER
- "DA": DATE
- "TI": TIME
- "TZ": TIME WITH TIMEZONE
- "TS": TIMESTAMP
- "SZ": TIMESTAMP WITH TIMEZONE
- "YM": INTERVAL YEAR TO MONTH
- "DS": INTERVAL DAY TO SECOND
- "BO": BOOLEAN
- "RI": ROWID

In the type comparison table, the built-in data types are represented by an abbreviated word en closed in double quotes ("").

- "CHAR": CHARACTER
- "VARCHAR": CHARACTER VARYING
- "LONG VARCHAR": CHARACTER LONG VARYING
- "VARBINARY": BINARY VARYING
- "LONG VARBINARY": BINARY LONG VARYING
- "TIME_TZ": TIME WITH TIMEZONE
- "TIMESTAMP_TZ": TIMESTAMP WITH TIMEZONE
- "INTERVAL_YM": INTERVAL YEAR TO MONTH
- "INTERVAL_DS": INTERVAL DAY TO SECOND

Table 11-6 The representative types for type comparison

Data type	CHAR	VARCHAR	LONG VARCHAR	BINARY	VARBINARY	LONG VARBINARY	NATIVE SMALLINT	NATIVE INTEGER	NATIVE BIGINT	NATIVE REAL	NATIVE DOUBLE	NUMBER	NUMERIC	FLOAT	DATE	TIME	TIME TZ	TIMESTAMP	TIMESTAMP TZ	INTERVAL YM	INTERVAL DS	BOOLEAN	ROWID
CHAR	VC	VC	LC				NU	NU	NU	NU	ND	NU	NU	NU	DA	TI	TZ	TS	SZ	YM	DS	BO	RI
VARCHAR	VC	VC	LC				NU	NU	NU	NU	ND	NU	NU	NU	DA	TI	TZ	TS	SZ	YM	DS	BO	RI
LONG VARCHAR	LC	LC	LC				NU	NU	NU	NU	ND	NU	NU	NU	DA	TI	TZ	TS	SZ	YM	DS	BO	RI
BINARY				VB	VB	LB																	
VARBINARY				VB	VB	LB																	
LONG VARBINARY				LB	LB	LB																	
NATIVE_SMALLINT	NU	NU	NU				NB	NB	NB	ND	ND	NU	NU	NU						YM	DS		
NATIVE_INTEGER	NU	NU	NU				NB	NB	NB	ND	ND	NU	NU	NU						YM	DS		
NATIVE_BIGINT	NU	NU	NU				NB	NB	NB	ND	ND	NU	NU	NU						YM	DS		
NATIVE_REAL	NU	NU	NU				ND	ND	ND	ND	ND	NU	NU	NU									
NATIVE_DOUBLE	ND	ND	ND				ND	ND	ND	ND	ND	ND	ND	ND									

Data type	CHAR	VARCHAR	LONG VARCHAR	BINARY	VARBINARY	LONG VARBINARY	NATIVE SMALLINT	NATIVE INTEGER	NATIVE BIGINT	NATIVE REAL	NATIVE DOUBLE	NUMBER	NUMERIC	FLOAT	DATE	TIME	TIME TZ	TIMESTAMP	TIMESTAMP TZ	INTERVALYM	INTERVALDS	BOOLEAN	ROWID
NUMBER	NU	NU	NU				NU	NU	NU	NU	ND	NU	NU	NU						YM	DS		
NUMERIC	NU	NU	NU				NU	NU	NU	NU	ND	NU	NU	NU						YM	DS		
FLOAT	NU	NU	NU				NU	NU	NU	NU	ND	NU	NU	NU						YM	DS		
DATE	DA	DA	DA												DA			TS	SZ				
TIME	TI	TI	TI													TI	TZ						
TIME_TZ	TZ	TZ	TZ													TZ	TZ						
TIMESTAMP	TS	TS	TS												TS			TS	SZ				
TIMESTAMP_TZ	SZ	SZ	SZ												SZ			SZ	SZ				
INTERVAL_YM	YM	YM	YM				YM	YM	YM			YM	YM	YM						YM			
INTERVAL_DS	DS	DS	DS				DS	DS	DS			DS	DS	DS							DS		
BOOLEAN	BO	BO	BO																			BO	
ROWID	RI	RI	RI																				RI

Table 11-7 Type conversion for the VC comparison

Source type	Converted type
CHAR	CHAR (no conversion)

Source type	Converted type
VARCHAR	VARCHAR (no conversion)

Table 11-8 Type conversion for the LC comparison

Source type	Converted type
CHAR	CHAR (no conversion)
VARCHAR	VARCHAR (no conversion)
LONG VARCHAR	LONG VARCHAR (no conversion)

Table 11-9 Type conversion for the VB comparison

Source type	Converted type
BINARY	BINARY (no conversion)
VARBINARY	VARBINARY (no conversion)

Table 11-10 Type conversion for the LB comparison

Source type	Converted type
BINARY	BINARY (no conversion)
VARBINARY	VARBINARY (no conversion)
LONG VARBINARY	LONG VARBINARY (no conversion)

Table 11-11 Type conversion for the NB comparison

Source type	Converted type
CHAR	NATIVE_BIGINT
VARCHAR	NATIVE_BIGINT
LONG VARCHAR	NATIVE_BIGINT
NATIVE_SMALLINT	NATIVE_SMALLINT (no conversion)
NATIVE_INTEGER	NATIVE_INTEGER (no conversion)
NATIVE_BIGINT	NATIVE_BIGINT (no conversion)

Table 11-12 Type conversion for the ND comparison

Source type	Converted type
CHAR	NATIVE_DOUBLE
VARCHAR	NATIVE_DOUBLE
LONG VARCHAR	NATIVE_DOUBLE
NATIVE_SMALLINT	NATIVE_SMALLINT (no conversion)
NATIVE_INTEGER	NATIVE_INTEGER (no conversion)
NATIVE_BIGINT	NATIVE_BIGINT (no conversion)
NATIVE_REAL	NATIVE_REAL (no conversion)
NATIVE_DOUBLE	NATIVE_DOUBLE (no conversion)

Source type	Converted type
NUMBER	NUMBER (no conversion)
NUMERIC	NUMERIC (no conversion)
FLOAT	FLOAT (no conversion)

Table 11-13 Type conversion for the NU comparison

Source type	Converted type
CHAR	NUMBER
VARCHAR	NUMBER
LONG VARCHAR	NUMBER
NATIVE_SMALLINT	NATIVE_SMALLINT (no conversion)
NATIVE_INTEGER	NATIVE_INTEGER (no conversion)
NATIVE_BIGINT	NATIVE_BIGINT (no conversion)
NATIVE_REAL	NATIVE_REAL (no conversion)
NATIVE_DOUBLE	NATIVE_DOUBLE (no conversion)
NUMBER	NUMBER (no conversion)
NUMERIC	NUMERIC (no conversion)
FLOAT	FLOAT (no conversion)

Table 11-14 Type conversion for the DA comparison

Source type	Converted type
CHAR	DATE
VARCHAR	DATE
LONG VARCHAR	DATE
DATE	DATE (no conversion)

Table 11-15 Type conversion for the TI comparison

Source type	Converted type
CHAR	TIME
VARCHAR	TIME
LONG VARCHAR	TIME
TIME	TIME (no conversion)

Table 11-16 Type conversion for the TZ comparison

Source type	Converted type
CHAR	TIME_TZ
VARCHAR	TIME_TZ
LONG VARCHAR	TIME_TZ
TIME	TIME_TZ
TIME_TZ	TIME_TZ (no conversion)

Table 11-17 Type conversion for the TS comparison

Source type	Converted type
CHAR	TIMESTAMP
VARCHAR	TIMESTAMP
LONG VARCHAR	TIMESTAMP
DATE	DATE (no conversion)
TIMESTAMP	TIMESTAMP (no conversion)

Table 11-18 Type conversion for the SZ comparison

Source type	Converted type
CHAR	TIMESTAMP_TZ
VARCHAR	TIMESTAMP_TZ
LONG VARCHAR	TIMESTAMP_TZ
DATE	TIMESTAMP_TZ
TIMESTAMP	TIMESTAMP_TZ
TIMESTAMP_TZ	TIMESTAMP_TZ (no conversion)

Table 11-19 Type conversion for the YM comparison

Source type	Converted type
CHAR	INTERVAL_YM
VARCHAR	INTERVAL_YM
LONG VARCHAR	INTERVAL_YM
NATIVE_SMALLINT	INTERVAL_YM
NATIVE_INTEGER	INTERVAL_YM
NATIVE_BIGINT	INTERVAL_YM
NUMBER	INTERVAL_YM
NUMERIC	INTERVAL_YM
FLOAT	INTERVAL_YM
INTERVAL_YM	INTERVAL_YM (no conversion)

Table 11-20 Type conversion for the DS comparison

Source type	Converted type
CHAR	INTERVAL_DS
VARCHAR	INTERVAL_DS
LONG VARCHAR	INTERVAL_DS
NATIVE_SMALLINT	INTERVAL_DS
NATIVE_INTEGER	INTERVAL_DS
NATIVE_BIGINT	INTERVAL_DS
NUMBER	INTERVAL_DS
NUMERIC	INTERVAL_DS

Source type	Converted type
FLOAT	INTERVAL_DS
INTERVAL_DS	INTERVAL_DS (no conversion)

Table 11-21 Type conversion for the BO comparison

Source type	Converted type
CHAR	BOOLEAN
VARCHAR	BOOLEAN
LONG VARCHAR	BOOLEAN
BOOLEAN	BOOLEAN (no conversion)

Table 11-22 Type conversion for the RI comparison

Source type	Converted type
CHAR	ROWID
VARCHAR	ROWID
LONG VARCHAR	ROWID
ROWID	ROWID (no conversion)

Type Conversion

Type conversions are classified into implicit type conversion and explicit type conversion.

- Implicit type conversion occurs in an expression, an operator, a function, a condition for select, insert, delete, update.
- Explicit type conversion occurs through the CAST operator.

The availability of type conversion describes the availability of data type conversion from a type to another type.

In the type conversion table, the built-in data types are represented by an abbreviated string enc losed in double quotes ("").

- "CHAR": CHARACTER
- "VARCHAR": CHARACTER VARYING
- "LONG VARCHAR": CHARACTER LONG VARYING
- "VARBINARY": BINARY VARYING
- "LONG VARBINARY": BINARY LONG VARYING
- "TIME_TZ": TIME WITH TIMEZONE
- "TIMESTAMP_TZ": TIMESTAMP WITH TIMEZONE

- "INTERVAL_YM": INTERVAL YEAR TO MONTH
- "INTERVAL_DS": INTERVAL DAY TO SECOND

Table 11-23 The availability of type conversion

Data type	CHAR	VARCHAR	LONG VARCHAR	BINARY	VARBINARY	LONG VARBINARY	NATIVE SMALLINT	NATIVE INTEGER	NATIVE BIGINT	NATIVE REAL	NATIVE DOUBLE	NUMBER	NUMERIC	FLOAT	DATE	TIME	TIME TZ	TIMESTAMP	TIMESTAMP TZ	INTERVAL YM	INTERVAL DS	BOOLEAN	ROWID
CHAR	O	O	O				O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
VARCHAR	O	O	O				O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
LONG VARCHAR	O	O	O				O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
BINARY				O	O	O																	
VARBINARY				O	O	O																	
LONG VARBINARY				O	O	O																	
NATIVE_SMALLINT	O	O	O				O	O	O	O	O	O	O	O						O	O		
NATIVE_INTEGER	O	O	O				O	O	O	O	O	O	O	O						O	O		
NATIVE_BIGINT	O	O	O				O	O	O	O	O	O	O	O						O	O		
NATIVE_REA	O	O	O				O	O	O	O	O	O	O	O									

Data type	CHAR	VARCHAR	LONG VARCHAR	BINARY	VARBINARY	LONG VARBINARY	NATIVE SMALLINT	NATIVE INTEGER	NATIVE BIGINT	NATIVE REAL	NATIVE DOUBLE	NUMBER	NUMERIC	FLOAT	DATE	TIME	TIME TZ	TIMESTAMP	TIMESTAMP TZ	INTERVAL YM	INTERVAL DS	BOOLEAN	ROWID
L																							
NATIVE_DOUBLE	O	O	O				O	O	O	O	O	O	O	O									
NUMBER	O	O	O				O	O	O	O	O	O	O	O						O	O		
NUMERIC	O	O	O				O	O	O	O	O	O	O	O						O	O		
FLOAT	O	O	O				O	O	O	O	O	O	O	O						O	O		
DATE	O	O	O												O			O	O				
TIME	O	O	O													O	O						
TIME_TZ	O	O	O													O	O						
TIMESTAMP	O	O	O												O	O		O	O				
TIMESTAMP_TZ	O	O	O												O	O	O	O	O				
INTERVAL_YM	O	O	O				O	O	O			O	O	O						O			
INTERVAL_DS	O	O	O				O	O	O			O	O	O							O		
BOOLEAN	O	O	O																			O	
ROWID	O	O	O																				O

- Conversion to CHARACTER type
  - When the source type is CHARACTER type:
    - If the source type precision is bigger than the CHARACTER type precision, then an error occu rs.
    - If the source type precision is equal to the CHARACTER type precision, then the string is not c hanged.
    - If the source type precision is smaller than the CHARACTER type precision, then space charac ters are added to the string as many as the precision difference.
  - When the source type is CHARACTER VARYING type or CHARACTER LONG VARYING type:
    - If the source type string length is bigger than the CHARACTER type precision, then an error o ccurs.
    - If the source type string length is equal to the CHARACTER type precision, then the string is n ot changed.
    - If the source type string length is smaller than the CHARACTER type precision, then space ch aracters are added to the string as many as precision difference.
  - When the source type is numeric type, date/time type, INTERVAL type, BOOLEAN type or ROWID type:
    - If the converted string length of the source type is bigger than the CHARACTER type precisio n, then an error occurs.
    - If the converted string length of the source type is equal to the CHARACTER type precision, t he string is not changed.
    - If the converted string length of the source type is smaller than the CHARACTER type precisio n, then space characters are added to the string as many as precision difference.

- Conversion to CHARACTER VARYING type
  - When the source type is CHARACTER type:
    - If the source type precision is bigger than the CHARACTER VARYING type precision, then an error occurs.
    - If the source type precision is equal or smaller than the CHARACTER VARYING type precision, then the string is not changed.
  - When the source type is CHARACTER VARYING type or CHARACTER LONG VARYING type:
    - If the source type string length is bigger than the CHARACTER VARYING type precision, then an error occurs.
    - If the source type string length is equal or smaller than the CHARACTER VARYING type precis ion, then the string is not changed.
  - When the source type is numeric type, date/time type, INTERVAL type, BOOLEAN type or ROWID type:
    - If the converted string length of the source type is bigger than the CHARACTER VARYING typ e precision, then an error occurs.
    - If the converted string length of the source type is equal or smaller than the CHARACTER VA RYING type precision, then the string is not changed.

- Conversion to CHARACTER LONG VARYING type
  - When the source type is CHARACTER STRING type:

        - The source type string is not changed.
    - When the source type is numeric type, date/time type, INTERVAL type, BOOLEAN type or ROWID type:
        - The converted string of the source type is not changed.

- Conversion to BINARY type
    - When the source type is BINARY type:
        - If the source type precision is bigger than the BINARY type precision, then an error occurs.
        - If the source type precision is equal to the BINARY type precision, then the binary string is not changed.
        - If the source type precision is smaller than the BINARY type precision, the X'00' characters ar e added to the binary string as many as the precision difference.
    - When the source type is BINARY VARYING type or BINARY LONG VARYING type:
        - If the source type binary string length is bigger than the BINARY type precision, then an error occurs.
        - If the source type binary string length is equal to the BINARY type precision, then the binary s tring is not changed.
        - If the source type binary string length is smaller than the BINARY type precision, the X'00' ch aracters are added to a binary string as many as precision difference.

- Conversion to BINARY VARYING type
    - When the source type is BINARY type:
        - If the source type precision is bigger than the BINARY VARYING type precision, then an error occurs.
        - If the source type precision is equal or smaller than the BINARY VARYING type precision, the n the binary string is not changed.
    - When the source type is BINARY VARYING type, BINARY LONG VARYING type:
        - If the source type binary string length is bigger than the BINARY VARYING type precision, the n an error occurs.
        - If the source type binary string length is equal or smaller than the BINARY VARYING type pre cision, the binary string is not changed.

- Conversion to BINARY LONG VARYING type
    - If the source type is BINARY STRING type, the source type binary string is not changed.

- Conversion to numeric type
    - When the source type is CHARACTER STRING type:
        - If the string does not comply with the numeric format, then an error occurs.
        - An overflow or rounding can occur due to the precision and scale which is defined in the con verted type.
    - When the source type is numeric type:
        - An overflow or rounding can occur due to the precision and scale which is defined in the con verted type.

  - When the source type is INTERVAL type:
    - It can be converted to the numeric type only when the source type is the single field (YEAR, MONTH, DAY, HOUR, MINUTE, SECOND).
    - An overflow or rounding can occur due to the precision and scale which is defined in the converted type.

- Conversion to DATE type
  - When the source type is CHARACTER STRING type:
    - If the string does not comply with the DATE type format, then an error occurs.
    - An overflow can occur due to a value range which is defined in the converted type.
  - When the source type is DATE type or TIMESTAMP type:
    - The error does not occur.
  - When the source type is TIMESTAMP WITH TIME ZONE type:
    - It is converted to the DATE type value upon consideration of the time zone offset.

- Conversion to TIME type
  - When the source type is CHARACTER STRING type:
    - If the string does not comply with the TIME type format, then an error occurs.
    - A rounding can occur due to a value range which is defined in the converted type.
  - When the source type is TIME type or TIMESTAMP type:
    - The error does not occur.
  - When the source type is TIME WITH TIME ZONE type or TIMESTAMP WITH TIME ZONE type:
    - It is converted to the value of TIME type upon consideration of the time zone offset.

- Conversion to TIME WITH TIME ZONE type
  - When the source type is CHARACTER STRING type:
    - If the string does not comply with the TIME WITH TIME ZONE type format, then an error occurs.
    - A rounding can occur due to a value range which is defined in the converted type.
  - When the source type is TIME type, TIME WITH TIME ZONE type or TIMESTAMP WITH TIME ZONE type:
    - It is converted to the value of TIME WITH TIME ZONE type upon consideration of the time zone offset.

- Conversion to TIMESTAMP type
  - When the source type is CHARACTER STRING type:
    - If the string does not comply with the TIMESTAMP type format, then an error occurs.
    - An overflow or rounding can occur due to a value range which is defined in the converted type.
  - When the source type is DATE type or TIMESTAMP type:
    - The error does not occur.
  - When the source type is TIMESTAMP WITH TIME ZONE type:
    - It is converted to the TIMESTAMP type value upon consideration of the time zone offset.

- Conversion to TIMESTAMP WITH TIME ZONE type
  - When the source type is CHARACTER STRING type:
    - If the string does not comply with the TIMESTAMP WITH TIME ZONE type format, then an error occurs.
    - An overflow or rounding can occur due to a value range which is defined in the converted type.
  - When the source type is DATE type, TIMESTAMP type or TIMESTAMP WITH TIME ZONE type:
    - It is converted to the value of TIMESTAMP WITH TIME ZONE type upon consideration of the time zone offset.

- Conversion to INTERVAL YEAR TO MONTH family type
  - When the source type is CHARACTER STRING type:
    - If the string does not comply with the year-month interval literal format, then an error occurs.
    - For more information, refer to **Interval Literals**.
    - An overflow can occur due to a precision which is defined in the converted type.
  - When the source type is NATIVE_SMALLINT, NATIVE_INTEGER, NATIVE_BIGINT, NUMBER, NUMERIC or FLOAT type:
    - The converted type should be a single field (YEAR, MONTH).
    - An overflow can occur due to a precision which is defined in the converted type.
  - When the source type is INTERVAL YEAR TO MONTH family type:
    - An overflow can occur due to a precision which is defined in the converted type.

- Conversion to INTERVAL DAY TO SECOND family type
  - When the source type is CHARACTER STRING type:
    - If the string does not comply with the day-time interval literal format, then an error occurs.
    - For more information, refer to **Interval Literals**.
    - An overflow or rounding can occur due to the precision which is defined in the converted type.
  - When the source type is NATIVE_SMALLINT, NATIVE_INTEGER, NATIVE_BIGINT, NUMBER, NUMERIC, or FLOAT type:
    - The converted type should be a single field (DAY, HOUR, MINUTE, SECOND)
    - An overflow or rounding can occur due to the precision which is defined in the converted type.
  - When the source type is INTERVAL DAY TO SECOND family type:
    - An overflow or rounding can occur due to the precision which is defined in the converted type.

- Conversion to BOOLEAN type
  - When the source type is CHARACTER STRING type:
    - The conversion is available when the string is "TRUE" or "FALSE", and it is case-insensitive. (It is convertible even when the string includes a space before and after it.)
  - When the source type is BOOLEAN type:
    - The error does not occur.

- Conversion to ROWID type
  - When the source type is CHARACTER STRING type:
    - If the string does not comply with the ROWID type format, then an error occurs.
  - When the source type is ROWID type:
    - The error does not occur.

Type Combination

When Type Combination Is Required

A CASE operator and a **set operator** have many expressions as a result of operation.
Each expression can have different types each other as follows. In this case, the result type should be dete rmined.

- The following describes an excution result of a CASE operator.

```
SELECT CASE expr WHEN expr THEN char(3)
                 WHEN expr THEN char(5)
                 ELSE char(1)
       END
  FROM t1;
```

- The following describes an excution result of a set operator.

```
SELECT float_column
  FROM t1
UNION ALL
SELECT number_precision_column
  FROM t2
UNION ALL
SELECT native_integer_column
  FROM t3;
```

A rule is applied to determine the result type according to the type combination. The following is an exa mple of applying the rule.

- Result type combination rule
  - **set operator**
  - CASE operator
    - **CASE Expression**

        - **COALESCE**
        - **NULLIF**

Result Type Combination Rule

Each expression's data type should be the same family type which is available to combine.

- Examples of applying result type combination rule
    - **set operator**
    - CASE operator
        - **CASE Expression**
        - **COALESCE**
        - **NULLIF**

The result types determined by result type combination rule are described in the following table.

The following abbreviations are used to describe the result type combination rule.

- "VC": CHARACTER VARYING
- "LC": CHARACTER LONG VARYING
- "VB": BINARY VARYING
- "LB": BINARY LONG VARYING
- "NS": NATIVE_SMALLINT
- "NI": NATIVE_INTEGER
- "NB": NATIVE_BIGINT
- "NR": NATIVE_REAL
- "ND": NATIVE_DOUBLE
- "FL": FLOAT
- "NU": NUMBER
- "DA": DATE
- "TI": TIME
- "TZ": TIME WITH TIMEZONE
- "TS": TIMESTAMP
- "SZ": TIMESTAMP WITH TIMEZONE
- "YM": INTERVAL YEAR TO MONTH
- "DS": INTERVAL DAY TO SECOND
- "BO": BOOLEAN
- "RI": ROWID

In result type combination table, the built-in data types are represented by an abbreviated word enclosed in double quotes ("").

- "CHAR": CHARACTER
- "VARCHAR": CHARACTER VARYING
- "LONG VARCHAR": CHARACTER LONG VARYING
- "BINARY": BINARY
- "VARBINARY": BINARY VARYING
- "LONG VARBINARY": BINARY LONG VARYING
- "TIME_TZ": TIME WITH TIMEZONE
- "TIMESTAMP_TZ": TIMESTAMP WITH TIMEZONE
- "INTERVAL_YM": INTERVAL YEAR TO MONTH
- "INTERVAL_DS": INTERVAL DAY TO SECOND

Table 11-24 The result type determined by result type combination rule

Data type	CHAR	VARCHAR	LONG VARCHAR	BINARY	VARBINARY	LONG VARBINARY	NATIVE SMALLINT	NATIVE INTEGER	NATIVE BIGINT	NATIVE REAL	NATIVE DOUBLE	NUMBER	NUMERIC	FLOAT	DATE	TIME	TIME TZ	TIMESTAMP	TIMESTAMP TZ	INTERVAL YM	INTERVAL DS	BOOLEAN	ROWID
CHAR	VC	VC																					
VARCHAR	VC	VC																					
LONG VARCHAR			LC																				
BINARY				VB	VB																		
VARBINARY				VB	VB																		
LONG VARBI						LB																	

Data type	CHAR	VARCHAR	LONG VARCHAR	BINARY	VARBINARY	LONG VARBINARY	NATIVE SMALLINT	NATIVE INTEGER	NATIVE BIGINT	NATIVE REAL	NATIVE DOUBLE	NUMBER	NUMERIC	FLOAT	DATE	TIME	TIME TZ	TIMESTAMP	TIMESTAMP TZ	INTERVALYM	INTERVALDS	BOOLEAN	ROWID
NARY																							
NATIVE_SMALLINT							NS	NI	NB	ND	ND	NU	NU	NU									
NATIVE_INTEGER							NI	NI	NB	ND	ND	NU	NU	NU									
NATIVE_BIGINT							NB	NB	NB	ND	ND	NU	NU	NU									
NATIVE_REAL							ND	ND	ND	NR	ND	ND	ND	ND									
NATIVE_DOUBLE							ND	ND	ND	ND	ND	ND	ND	ND									
NUMBER							NU	NU	NU	ND	ND	NU	NU	NU									
NUMERIC							NU	NU	NU	ND	ND	NU	NU	NU									
FLOAT							NU	NU	NU	ND	ND	NU	NU	FL									
DATE															DA			TS	SZ				
TIME																TI	TZ						
TIME_TZ																TZ	TZ						
TIMESTAMP															TS			TS	SZ				
TIMESTAMP_															SZ			SZ	SZ				

Data type	CHAR	VARCHAR	LONG VARCHAR	BINARY	VARBINARY	LONG VARBINARY	NATIVE SMALLINT	NATIVE INTEGER	NATIVE BIGINT	NATIVE REAL	NATIVE DOUBLE	NUMBER	NUMERIC	FLOAT	DATE	TIME	TIME TZ	TIMESTAMP	TIMESTAMP TZ	INTERVAL YM	INTERVAL DS	BOOLEAN	ROWID
TZ																							
INTERVAL_YM																				YM			
INTERVAL_DS																					DS		
BOOLEAN																						BO	
ROWID																							RI

- The result type of when all is CHAR type
  - If the lengths are different, then it is VARCHAR type.
  - If the lengths are equal, then it is CHAR type.
- The result type of when all is BINARY type
  - If the lengths are different, then it is VARBINARY type.
  - If the lengths are equal, then it is BINARY type.
- The result type of INTERVAL YEAR TO MONTH type
  - If YEAR type and MONTH type are mixed, then it is INTERVAL YEAR TO MONTH type.
  - If there is only YEAR type, then it is INTERVAL YEAR type.
  - If there is only MONTH type, then it is INTERVAL MONTH type.
- The result type of INTERVAL DAY TO SECOND type
  - The start field is the biggest range field of each target expression.
  - The end field is the smallest range field of each target expression.
  - e.g. {INTERVAL DAY, INTERVAL HOUR} => INTERVAL DAY TO HOUR
- The precision, scale of the result type
  - CHARACTER STRING type
    - The maximum character length of the target expression

    - BINARY STRING type
        - The maximum character length of the target expression
    - NUMERIC type
        - It specifies an acceptable maximum range of the type's value.
    - TIME/TIMESTAMP type
        - The maximum fractional seconds precision of the target expression
    - INTERVAL YEAR TO MONTH
        - The maximum leading precision of the target expression
    - INTERVAL DAY TO SECOND
        - Leading precision is the maximum leading precision of the start field
        - Fractional seconds precision is the maximum fractional seconds precision.
- For more information, refer to **Type Comparison**.

Compatibility for Data Type

The SQL standard compatibility for data type is as follows.

Table 11-25 SQL standard compatibility for data type

Feature ID	Description	Availability
B033	Untyped SQL-invoked function arguments	X
E011-01	INTEGER and SMALLINT data types	O
E011-02	REAL, DOUBLE PRECISION, and FLOAT data types	O
E011-03	DECIMAL and NUMERIC data types	X
E011-04	Arithmetic operators	O
E011-05	Numeric comparison	O
E011-06	Implicit casting among the numeric data types	O
E021-01	CHARACTER data type	O
E021-02	CHARACTER VARYING data type	O
E021-03	Character literals	O
E021-04	CHARACTER_LENGTH function	O
E021-05	OCTET_LENGTH function	O
E021-06	SUBSTRING function	O
E021-07	Character concatenation	O
E021-08	UPPER and LOWER functions	O
E021-09	TRIM function	O
E021-10	Implicit casting among the fixed-length and variable-length character string types	O
E021-11	POSITION function	O
E021-12	Character comparison	O
	Columns combined via table operators need not have exactly the same data	

Feature ID	Description	Availability
E071-05	type	O
F051-01	DATE data type (including support of DATE literal)	O
F051-02	TIME data type (including support of TIME literal) with fractional seconds precision of at least 0	O
F051-03	TIMESTAMP data type (including support of TIMESTAMP literal) with fractional seconds precision of at least 0 and 6	O
F051-04	Comparison predicate on DATE, TIME, and TIMESTAMP data types	X
F051-05	Explicit CAST between datetime types and character string types	O
F054	TIMESTAMP in DATE type precedence list	X
F382	Alter column data type	O
F611	Indicator data types	X
F741	Referential MATCH types	X
J521	JDBC data types	X
J622	external Java types	X
S011-01	USER_DEFINED_TYPES view	X
S023	Basic structured types	X
S024	Enhanced structured types	X
S025	Final structured types	X
S026	Self-referencing structured types	X
S041	Basic reference types	X
S043	Enhanced reference types	X
S051	Create table of type	X
S071	SQL paths in function and type name resolution	X
S091-01	Arrays of built-in data types	X
S091-02	Arrays of distinct types	X
S092	Arrays of user-defined types	X
S094	Arrays of reference types	X
S161	Subtype treatment	X
S162	Subtype treatment for references	X
S201-02	Array as result type of functions	X
S231	Structured type locators	X
S261	Specific type method	X
S272	Multisets of user-defined types	X
S274	Multisets of reference types	X
S281	Nested collection types	X
S401	Distinct types based on array types	X
S402	Distinct types based on distinct types	X
T021	BINARY and VARBINARY data types	O
T022	Advanced support for BINARY and VARBINARY data types	O
T031	BOOLEAN data type	O

Feature ID	Description	Availability
T041	Basic LOB data type support	X
T042	Extended LOB data type support	X
T051	Row types	X
T071	BIGINT data type	O
T201	Comparable data types for referential constraints	X
T322	Declared data type attributes	X
X010	XML type	X
X011	Arrays of XML type	X
X012	XMultisets of XML type	X
X013	Distinct types of XML type	X
X014	Attributes of XML type	X
X015	Fields of XML type	X
X181	XML(DOCUMENT(UNTYPED)) type	X
X182	XML(DOCUMENT(ANY)) type	X
X190	XML(SEQUENCE) type	X
X191	XML(DOCUMENT(XMLSCHEMA)) type	X
X192	XML(CONTENT(XMLSCHEMA)) type	X
X231	XML(CONTENT(UNTYPED)) type	X
X232	XML(CONTENT(ANY)) type	X
X251	Persistent XML values of XML(DOCUMENT(UNTYPED)) type	X
X252	Persistent XML values of XML(DOCUMENT(ANY)) type	X
X253	Persistent XML values of XML(CONTENT(UNTYPED)) type	X
X254	Persistent XML values of XML(CONTENT(ANY)) type	X
X255	Persistent XML values of XML(SEQUENCE) type	X
X256	Persistent XML values of XML(DOCUMENT(XMLSCHEMA)) type	X
X257	Persistent XML values of XML(CONTENT(XMLSCHEMA)) type	X
X260	XML type: ELEMENT clause	X
X261	XML type: NAMESPACE without ELEMENT clause	X
X263	XML type: NO NAMESPACE with ELEMENT clause	X
X264	XML type: schema location	X
X410	Alter column data type: XML type	X

11.3 Format String

Format string defines the format which is used when a numeric type or date/time type is converted to a c haracter string or when a character string is converted to a numeric types or date/time type.

- When a numeric type or date/time type is converted to a character string type, the representation for mat of the string is as follows.
  - Refer to **TO_CHAR(number)**, **TO_CHAR(datetime)**.
  - Numeric type: TO_CHAR(1234.56, 'S9,999.99') → '+1,234.56'
  - Date/time type: TO_CHAR(SYSDATE, 'YYYY-MM-DD') → '2012-07-15'
- When a character string is converted to the numeric type or date/time type, the representation forma t of the string is as follows.
  - Refer to **TO_NUMBER**, **TO_NATIVE_REAL**, **TO_NATIVE_DOUBLE**.
  - Refer to **TO_DATE**.
  - Refer to **TO_TIMESTAMP**, **TO_TIMESTAMP_WITH_TIME_ZONE**.
  - Refer to **TO_TIME**, **TO_TIME_WITH_TIME_ZONE** .
  - Numeric type: TO_NUMBER('+1,234.56', 'S9,999.99') → NUMBER TYPE
  - Date/time type: TO_DATE('2012-07-15', 'YYYY-MM-DD') → DATE TYPE

Format strings are classified according to the type.
- Numeric data type: Refer to **Number Format String**.
- Date/time type: Refer to **Datetime Format String**.

Number Format String

Number format string defines the format which is used when a numeric type is converted to a character s tring type, or when a character string type is converted to a numeric type.

Number format string is used as an argument of the functions such as **TO_CHAR(number)**, **TO_NATIVE_ SMALLINT**, **TO_NATIVE_INTEGER**, **TO_NATIVE_BIGINT**, **TO_NUMBER**, **TO_NATIVE_REAL**, **TO_NATIVE_DO UBLE**.

Number format string can specify multiple format elements according to the desired format.

All number format elements are rounded off to fit the format.
If the number of digits before the decimal point of the value to be converted is bigger than the number o f digits specified in the format string, then they are replaced with '#' character.
If the format element representing the sign of MI, S, PR is not specified, a negative number returns - sign and a positive number returns a white space to the front of the number.

Table 11-26 Number format elements

Format element	Example	Description
, (comma)	9,999	It returns a comma to the specified position. Multiple commas can be specified. Format string can not begin with a comma, and it can not come after the decimal point (.).
. (period)	99.99	It returns a decimal point (.) to the specified position. The decimal point in the format string can be specified only once.
$	$9999	It returns the $ sign to the front of the number.
0	0999 9990	It returns zero (0) to the front of or to the end of the number. If the number of digit of the value to be converted is smaller than the number of digit to the zero position of the format string, then the gap is filled with zero (0)s and is returned.
9	9999	It returns a white space and numbers according to the sign and the number of specified 9. If the number of digit of the value to be converted is smaller than the number of the specified 9, then the gap is filled with white spaces and is returned. For a positive number, a white space is returned to the front of the number. For a negative number, '-' symbol is returned to the front of the number. If the value before the format string's decimal point is 0, then 0 is returned as a white space. e.g. TO_CHAR(0.123, '9.999') → .123 e.g. TO_CHAR(0, '9') → 0
B	B9999	If the value is zero, it returns a white space.
EEEE	9.9EEEE	It returns in exponential notation. It can be at the end of format string or it can be in front of S, MI, PR. It can not be specified together with a comma (,).
MI	9999MI	For a positive number, a white space is returned to the end of the number. For a negative number, '-' symbol is returned to the end of the number. It can be specified only at the end of format string and it can not be specified together with S, PR.
PR	9999PR	For a positive number, white spaces are returned to the beginning and end of the number. For a negative number, it returns the number into the inside of angle brackets. <number> It can be specified only at the end of format string, and it can not be specified together with S, MI.
RN rn	RN rn	Roman numerals are converted to uppercase and returned. (RN) Roman numerals are converted to lowercase and returned. (rn) Only the numbers between 1 ~ 3999 are returned. It can be specified together only with FM format element, but it can not be specified with any other format elements. It can not be used in TO_NUMBER function.

Format element	Example	Description
S	S9999 9999S	For a positive number, '+' symbol is returned to the front of the number. For a negative number '-' symbol is returned to the front of the number. (S9999) For a positive number, '+' symbol is returned to the end of the number. For a negative number '-' symbol is returned to the end of the number. (9999S) It can be specified only at the beginning of format string or at the end of format string. It can not be specified together with MI, PR.
V	999V99	10^n(n: the digit number of 9 after V format element) multiplied by the value is returned. It can not specified together with the decimal point (.). It can not be used in TO_NUMBER function.
X	XXXX xxxx	It returns the white space and hexadecimal number according to the digit number of the specified X. It converts an integer value to the hexadecimal number, and returns it. (A non-integer value is rounded off to make it to an integer value.) XXX returns hexadecimal uppercase letters and xxxx returns hexadecimal lowercase letters. If the number of the converted hexadecimal digit is smaller than the number of the specified X, then the gap is filled with white spaces and is returned. Only 0 and positive integers are processed, and negative numbers are replaced with '#'. It can be specified together only with format element 0 and FM, but it can not be specified with any other format elements.
FM	FM	It removes the front and end white spaces, and returns left aligned effect. It removes the front and end white spaces of the number. It removes zero(0)s under the decimal point which are added by 9 format element.

Followings are examples of using number format string.

```
TO_CHAR( 12345, '99,999' )          : ' 12,345'
TO_CHAR( 123456789, '999,999,999' ) : ' 123,456,789'
TO_CHAR( 12.345, '99.999' )         : ' 12.345'
TO_CHAR( 1234.56, '$9,999.99' )     : ' $1,234.56'
TO_CHAR( 123, '099999' )            : ' 000123'
TO_CHAR( 0.2, '0.9' )               : ' 0.2'
TO_CHAR( 123.45, '999999.99' )      : '    123.45'
TO_CHAR( -123.45, '999999.99' )     : '   -123.45'
TO_CHAR( 123.45, 'FM999999.99' )    : '123.45'
TO_CHAR( -123.45, 'FM999999.99' )   : '-123.45'
```

```
TO_CHAR( 12345.67, '999.99' )        : '#######'
TO_CHAR( 123.100567, '999.999' )     : ' 123.101'
TO_CHAR( 0.2, '90.99' )              : '  0.20'
TO_CHAR( 0.2, '99.99' )              : '   .20'
TO_CHAR( 0, '90.99' )                : '  0.00'
TO_CHAR( 0, 'B90.99' )               : '      '
TO_CHAR( 123.45, '9.9EEEE' )         : '  1.2E+02'
TO_CHAR( 123.45, '999.99MI' )        : '123.45 '
TO_CHAR( -123.45, '999.99MI' )       : '123.45-'
TO_CHAR( 123.45, '999.99PR' )        : ' 123.45 '
TO_CHAR( -123.45, '999.99PR' )       : '<123.45>'
TO_CHAR( 123, 'RN' )                 : '         CXXIII'
TO_CHAR( 123, 'rn' )                 : '         cxxiii'
TO_CHAR( 123, 'FMRN' )               : 'CXXIII'
TO_CHAR( 4000, 'RN' )                : '###############'
TO_CHAR( 123.45, 'S999.99' )         : '+123.45'
TO_CHAR( -123.45, 'S999.99' )        : '-123.45'
TO_CHAR( 123.45, '999.99S' )         : '123.45+'
TO_CHAR( -123.45, '999.99S' )        : '123.45-'
TO_CHAR( 123.45, '999V999' )         : ' 123450'
TO_CHAR( 123, 'XX' )                 : ' 7B'
TO_CHAR( 123, 'xx' )                 : ' 7b'
TO_CHAR( 45678, 'XXXXXXX' )          : '    B26E'
TO_CHAR( 45678, 'FMXXXXXXX' )        : 'B26E'
TO_CHAR( 123.45, '99,999.999999' )   : '    123.450000'
TO_CHAR( 123.45, 'FM99,999.999999' ) : '123.45'
```

Datetime Format String

Datetime format string defines the format which is used when a date/time type is converted to a characte r string type, or when a character string type is converted to a date/time type.

Datetime format string is used as an argument of the functions such as **TO_CHAR(datetime)**, **TO_DATE**, **TO_TIMESTAMP**, **TO_TIMESTAMP_WITH_TIME_ZONE**, **TO_TIME**, **TO_TIME_WITH_TIME_ZONE**.

For datetime format string, if the format string is not specified, then the default value is used. The default value of each type is specified in the session property (NLS _ * _ FORMAT).

- DATE: Refer to **NLS_DATE_FORMAT**.
- TIMESTAMP: Refer to **NLS_TIMESTAMP_FORMAT**.

- TIMESTAMP WITH TIME ZONE: Refer to **NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT**.
- TIME: Refer to **NLS_TIME_FORMAT**.
- TIME WITH TIME ZONE: Refer to **NLS_TIME_WITH_TIME_ZONE_FORMAT**.

NLS * _FORMAT values can be changed by using **ALTER SESSION SET property_name**.

In datetime format string, multiple format elements can be specified upon the desired representation.

Table 11-27 Datetime format elements

Format element	Whether to use TO_* datetime	Description
- / , . ; : "text" Special characters	Y	It returns the format element character to the specified location.
AD A.D.	Y	AD with or without periods.
AM A.M.	Y	AM with or without periods.
BC B.C.	Y	BC with or without periods.
CC	N	Century If the last two digits of the four digits year is 01~ 99, the value which is added by one to the first two digits is returned. (e.g. If the year is 2005, 21 is returned.) If the last two digits of the four digits year is 00, The first two digits value is returned. (e.g. If the year is 2000, 20 is returned.)
D	Y	It returns the sequence of the day in a week. (1 ~ 7) Sunday is 1, saturday is 7, and so on.
DAY Day day	Y	It returns the day of the week. (e.g. SUNDAY) • DAY: The day which is all in uppercase is returned. • Day: The day whose first character is uppercase and others are lowercase is returned. • day: The day which is all in lowercase is returned.
DD	Y	It returns the sequence of the day in a month. (1 ~ 31)
DDD	Y	It returns the sequence of the day in a year. (1 ~ 366)
DY		It returns the abbreviated word for the day of the week. (e.g. SUN) • DY: The day which is all in uppercase is returned.

Format element	Whether to use TO_* datetime	Description
Dy dy	Y	• Dy: The day whose first character is uppercase and others are lowercase is returned. • dy: The day which is all in lowercase is returned.
FF[1..6]	Y	It returns fractional seconds as many as the number of the specified digits (1-6) after FF. If the number is not specified, the default value is 6. (FF is equal to FF6.) If the number of fractional seconds digit is bigger than the number specified after FF, then it is rounded down. If the number of fractional seconds digit is smaller than the number specified after FF, then zero(0) is added according to the specified number. It can not be used in DATE type.
HH HH12	Y	The hour (1 ~ 12)
HH24	Y	The hour (0 ~ 23)
IW	N	The week containing the first thursday of the year designated as the calendar week by ISO 8601 standards (1 ~ 52 week or 1 ~ 53 weeks) becomes the first week. • The calendar week starts from monday. • The first calendar week includes January 4th. • The first calendar week may includes December 29th, 30th, and 31st. • The last calendar week may include January 1st, 2nd, and 3rd.
IYYY	N	The 4 digits year embracing the calendar week defined by ISO 8601 standards
IYY IY I	N	The 3 digits year embracing the calendar week defined by ISO 8601 standards The 2 digits year embracing the calendar week defined by ISO 8601 standards The single digit year embracing the calendar week defined by ISO 8601 standards
J	Y	Julian day: The number of days since BC 4714-11-24
MI	Y	Minute (0 ~ 59)
MM	Y	Month (01 ~ 12), January(01)~December(12)
MON Mon mon	Y	The abbreviated word for the month. (e.g. JAN) • MON: The month which is all in uppercase is returned. • Mon: The month whose first character is uppercase and others are lowercase is returned. • mon: The month which is all in lowercase is returned.
MONTH Month	Y	The month name (e.g. JANUARY) • MONTH: All uppercase month name is returned. • Month: The month name that only the first letter is uppercase and oth

Format element	Whether to use TO_* datetime	Description
month		ers are lowercase is returned. • month: The month of which is all in lowercase is returned.
PM P.M.	Y	PM with or without periods.
Q	N	The quarter of the year (1 ~ 4) January to March is 1 and October to December is 4.
RM Rm rm	Y	It returns the roman numeral month. (e.g. I) • RM: The month which is all in uppercase is returned. • Rm: The month whose first character is uppercase and others are lowercase is returned. • rm: The month which is all in lowercase is returned.
RR	Y	Adjusted two digit year The two digit year represented by RR can be converted to four digit year as follows. • When the two digit year represented by RR is 00~49: ◦ If the last two digits of the current year is 00~50, ▪ the four digit year is represented using the first two digits of the current year and the two digits which is represented by RR. ◦ If the last two digits of the current year is 51~99, ▪ the four digit year is represented using "the first two digits of the current year+1" and the two digits which is represented by RR. • When the two digit year represented by RR is 50~99: ◦ If the last two digits of the current year is 00~50, ▪ the four digit year is represented using "the first two digit of the current year - 1" and the two digits which is represented by RR. ◦ If the last two digit of the current year is 51~99, ▪ the four digit year is expressed using the first two digit of the current year and the two digits which is represented by RR.
RRRR	Y	Adjusted four digit year Two digit or four digit can be input. Two digit input is processed in the same way as RR.
SS	Y	Second (0 ~ 59)
SSSSS	Y	Seconds since last midnight (0 ~ 86399)
TZH	Y	Time Zone Hour It can not be used in DATE, TIMESTAMP, TIME types. It is available in TIMESTAMP WITH TIME ZONE, TIME WITH TIME ZONE types.
TZM	Y	Time Zone Minute It can not be used in DATE, TIMESTAMP, TIME types. It can be used only in

Format element	Whether to use TO_* datetime	Description
		TIMESTAMP WITH TIME ZONE, TIME WITH TIME ZONE types.
WW	N	The sequence of the week in a year. (1~ 53) The first week 1 starts on the first day of the year and continues to the seven th day of the year.
W	N	The sequence of the week in a month. (1 ~ 5) The first week 1 starts on the first day of the month and ends on the seventh day.
Y,YYY	Y	It returns the year with comma in the Y,YYY form.
YYYY SYYYY	Y	Four digit year. SYYYY displays the sign of year. • If it is BC, it displays '-'. If it is AD, it displays ' '.
YYY YY Y	Y	• YYY: The last three digit year of the current year • YY: The last two digit year of the current year • Y: The last one digit year of the current year

The followings are examples of using datetime format string.

```
* - / , . ; : "text" Special character
   • TO_CHAR( TO_DATE( '2012-07-15 03:30:30', 'YYYY-MM-DD HH12:MI:SS' ),
              'YYYY/MM/DD HH12:MI:SS' )
     ==> '2012/07/15 03:30:30'
   • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ),
              'YYYY"year" DDD"th day"' )
     ==> '2012year 197th day'
```

```
* AD
   • TO_CHAR( TO_DATE( '2012-07-15 AD', 'YYYY-MM-DD AD' ), 'YYYY AD' )
     ==> '2012 AD'
   • TO_CHAR( TO_DATE( '0001-01-01 BC', 'YYYY-MM-DD AD'), 'YYYY AD' )
     ==> '0001 BC'
* BC
   • TO_CHAR( TO_DATE( '0001-01-01 BC', 'YYYY-MM-DD BC'), 'YYYY BC' )
     ==> '0001 BC'
   • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'YYYY BC' )
     ==> '2012 AD'
```

```
* AM
   • TO_CHAR( TO_DATE( '2012-07-15 03:30:30 AM',
                       'YYYY-MM-DD HH12:MI:SS AM' ),
```

```
            'HH12:MI:SS AM' )
    ==> '03:30:30 AM'
  • TO_CHAR( TO_DATE( '2012-07-15 21:30:30', 'YYYY-MM-DD HH24:MI:SS' ),
            'HH12:MI:SS AM' )
    ==> '09:30:30 PM'
* PM
  • TO_CHAR( TO_DATE( '2012-07-15 03:30:30',
                    'YYYY-MM-DD HH24:MI:SS' ),
            'HH12:MI:SS PM' )
    ==> '03:30:30 AM'
  • TO_CHAR( TO_DATE( '2012-07-15 09:30:30 PM',
                    'YYYY-MM-DD HH12:MI:SS PM' ),
            'HH12:MI:SS PM' )
    ==> '09:30:30 PM'
```

```
* CC
  • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'CC' )
    ==> '21'
```

```
* D
  • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'D' )
    ==> '1'
* DD
  • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'DD' )
    ==>  '15'
* DDD
  • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'DDD' )
    ==> '197'
```

```
* DAY
  • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'DAY' )
    ==> 'SUNDAY   '
  • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'Day' )
    ==> 'Sunday   '
  • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'day' )
    ==> 'sunday   '
* DY
  • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'DY' )
    ==> 'SUN'
  • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'Dy' )
    ==> 'Sun'
```

```
   • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'dy' )
     ==> 'sun'
```

```
* FF[1 ... 6]
   • TO_CHAR( TO_TIMESTAMP( '2012-07-15 03:30:45.123456',
                            'YYYY-MM-DD HH24:MI:SS.FF6' ),
              'FF' )
     ==> '123456'
   • TO_CHAR( TO_TIMESTAMP( '2012-07-15 03:30:45.123456',
                            'YYYY-MM-DD HH24:MI:SS.FF6' ),
              'FF5' )
     ==> '12345'
   • TO_CHAR( TO_TIMESTAMP( '2012-07-15 03:30:45.9',
                            'YYYY-MM-DD HH24:MI.SS.FF1' ) ,
              'FF6' )
     ==> '900000'
```

```
* HH HH12 HH24
   • TO_CHAR( TO_TIMESTAMP( '2012-07-15 03:30:45.123456',
                            'YYYY-MM-DD HH24:MI:SS.FF6' ),
              'HH12' )
     ==> '03'
   • TO_CHAR( TO_TIMESTAMP( '2012-07-15 23:30:45.123456',
                            'YYYY-MM-DD HH24:MI:SS.FF6' ),
              'HH12' )
     ==> '11'
   • TO_CHAR( TO_TIMESTAMP( '2012-07-15 23:30:45.123456',
                            'YYYY-MM-DD HH24:MI:SS.FF6' ),
              'HH24' )
     ==> '23'
```

```
* IW
   • TO_CHAR( DATE'2016-01-01', 'IW' )
     ==> 53
   • TO_CHAR( DATE'2014-12-30', 'IW' )
     ==> 01
```

```
* IYYY
   • TO_CHAR( DATE'2016-01-01', 'IYYY' )
     ==> 2015
   • TO_CHAR( DATE'2014-12-30', 'IYYY' )
```

```
    ==> 2015
* IYY
   • TO_CHAR( DATE'2016-01-01', 'IYY' )
    ==> 015
* IY
   • TO_CHAR( DATE'2016-01-01', 'IY' )
    ==> 15
* I
   • TO_CHAR( DATE'2016-01-01', 'I' )
    ==> 5
```

```
* J
   • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'J' )
    ==> '2456124'
   • TO_CHAR( TO_DATE( '2456124', 'J' ), 'YYYY-MM-DD' )
    ==> '2012-07-15'
```

```
* MI
   • TO_CHAR( TO_TIMESTAMP( '2012-07-15 23:30:45.123456',
             'YYYY-MM-DD HH24:MI:SS.FF6' ),
             'MI' )
    ==> '30'
```

```
* MM
   • TO_CHAR( TO_TIMESTAMP( '2012-07-15 23:30:45.123456',
                           'YYYY-MM-DD HH24:MI:SS.FF6' ),
             'MM' )
    ==> '07'
```

```
* MON
   • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'MON' )
    ==> 'JUL'
   • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'Mon' )
    ==> 'Jul'
   • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'mon' )
    ==> 'jul'
* MONTH
   • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'MONTH' )
    ==> 'JULY     '
   • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'Month' )
    ==> 'July     '
```

- TO_CHAR(TO_DATE('2012-07-15', 'YYYY-MM-DD'), 'month')
  ==> 'july   '

* Q
  - TO_CHAR(TO_DATE('2012-07-15', 'YYYY-MM-DD'), 'Q')
    ==> '3'

* RM
  - TO_CHAR(TO_DATE('2012-07-15', 'YYYY-MM-DD'), 'RM')
    ==> 'VII '
  - TO_CHAR(TO_DATE('2012-07-15', 'YYYY-MM-DD'), 'Rm')
    ==> 'Vii '
  - TO_CHAR(TO_DATE('2012-07-15', 'YYYY-MM-DD'), 'rm')
    ==> 'vii '

* RR, RRRR (The current year is 2014.)
  - TO_CHAR(TO_DATE('49-07-15', 'RR-MM-DD'), 'RRRR')
    ==> '2049'
  - TO_CHAR(TO_DATE('49-07-15', 'RR-MM-DD'), 'YYYY')
    ==> '2049'
  - TO_CHAR(TO_DATE('50-07-15', 'RR-MM-DD'), 'RRRR')
    ==> '1950'
  - TO_CHAR(TO_DATE('50-07-15', 'RR-MM-DD'), 'YYYY')
    ==> '1950'
  - TO_CHAR(TO_DATE('50-07-15', 'YY-MM-DD'), 'RRRR')
    ==> '2050'
  - TO_CHAR(TO_DATE('49-07-15', 'RRRR-MM-DD'), 'YYYY')
    ==> '2049'
  - TO_CHAR(TO_DATE('50-07-15', 'RRRR-MM-DD'), 'YYYY')
    ==> '1950'
* RR, RRRR (The current year is 2051.)
  - TO_CHAR(TO_DATE('49-07-15', 'RR-MM-DD'), 'RRRR')
    ==> '2149'
  - TO_CHAR(TO_DATE('49-07-15', 'RR-MM-DD'), 'YYYY')
    ==> '2149'
  - TO_CHAR(TO_DATE('50-07-15', 'RR-MM-DD'), 'RRRR')
    ==> '2050'
  - TO_CHAR(TO_DATE('50-07-15', 'RR-MM-DD'), 'YYYY')
    ==> '2050'
  - TO_CHAR(TO_DATE('50-07-15', 'YY-MM-DD'), 'RRRR')
    ==> '2050'

```
   • TO_CHAR( TO_DATE( '49-07-15', 'RRRR-MM-DD' ), 'YYYY' )
     ==> '2149'
   • TO_CHAR( TO_DATE( '50-07-15', 'RRRR-MM-DD' ), 'YYYY' )
     ==> '2050'
```

```
* SS
   • TO_CHAR( TO_TIMESTAMP( '2012-07-15 23:30:45.123456',
                            'YYYY-MM-DD HH24:MI:SS.FF6' ),
              'SS' )
     ==> '45'
* SSSSS
   • TO_CHAR( TO_TIMESTAMP( '2012-07-15 23:30:45.123456',
                            'YYYY-MM-DD HH24:MI:SS.FF6' ),
              'SSSSS' )
     ==> '84645'
```

```
* TZH
   • TO_CHAR( TO_TIMESTAMP_TZ( '2012-07-15 23:30:45.123456 +09:00',
                               'YYYY-MM-DD HH24:MI:SS.FF6 TZH:TZM' ),
              'TZH' )
     ==> '+09'
* TZM
   • TO_CHAR( TO_TIMESTAMP_TZ( '2012-07-15 23:30:45.123456 +09:00',
                               'YYYY-MM-DD HH24:MI:SS.FF6 TZH:TZM' ),
              'TZM' )
     ==> '00'
   • TO_CHAR( TO_TIMESTAMP_TZ( '2012-07-15 23:30:45.123456 +09:00',
                               'YYYY-MM-DD HH24:MI:SS.FF6 TZH:TZM' ),
              'TZH:TZM' )
     ==> '+09:00'
```

```
* WW
   • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'WW' )
     ==> '29'
* W
   • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'W' )
     ==> '3'
```

```
* Y,YYY
   • TO_CHAR( TO_DATE( '2,012-07-15', 'Y,YYY-MM-DD' ), 'Y,YYY' )
     ==> '2,012'
```

```
* YYYY
   • TO_CHAR( TO_DATE( '2012-07-15', 'YYYY-MM-DD' ), 'YYYY' )
     ==> '2012'
* SYYYY
   • TO_CHAR( TO_DATE( '-0001-01-01', 'SYYYY-MM-DD' ), 'SYYYY' )
     ==> '-0001'
   • TO_CHAR( TO_DATE( '2000-01-01', 'SYYYY-MM-DD' ), 'SYYYY' )
     ==> ' 2000'
* YYY
   • TO_CHAR( TO_DATE( '012-07-15', 'YYY-MM-DD' ), 'YYY' )
     ==> '012'
   • TO_CHAR( TO_DATE( '012-07-15', 'YYY-MM-DD' ), 'YYYY' )
     ==> '2012' ( (Current year / 1000) is 2 )
* YY
   • TO_CHAR( TO_DATE( '12-07-15', 'YY-MM-DD' ), 'YY' )
     ==> '12'
   • TO_CHAR( TO_DATE( '12-07-15', 'YY-MM-DD' ), 'YYYY' )
     ==> '2012' ( (Current year / 100) is 20 )
     ==> '2112' ( (Current year / 100) is 21 )
* Y
   • TO_CHAR( TO_DATE( '2-07-15', 'Y-MM-DD' ), 'Y' )
     ==> '2'
   • TO_CHAR( TO_DATE( '12-07-15', 'YY-MM-DD' ), 'YYYY' )
     ==> '2012' ( (Current year / 10 years) is 201 )
     ==> '2052' ( (Current year / 10 years) is 205 )
```

11.4 Expressions

Expression is a combination of value, operator and function for getting data values.

The following is the position of the SQL commands in which expression can be used.
- Target clause in SELECT
- GROUP BY clause in SELECT
- ORDER BY clause in SELECT
- WHERE clause and HAVING clause in SELECT
- INSERT VALUES clause
- UPDATE SET clause
- RETURN clause in INSERT, DELETE, UPDATE

Expression types are various as follows.
- Simple expression
- Compound expression
- Boolean value expression
- Case expression
- Datetime expression
- Scalar subquery expression
- Sequence manipulation expression

Simple expressions are column, pseudo columns, literals, and null value.
Compound expressions are combination of multiple expressions.

For more information, refer to the followings.
- **Null Value**
- **Literals**
- **Pseudo Columns**
- **Operators**
- **Functions**

Boolean Value Expression

Syntax

```
<boolean value expression> ::=
        <boolean term>
      | <boolean value expression> OR <boolean term>
```

```
<boolean term> ::=
        <boolean factor>
      ¦ <boolean term> AND <boolean factor>
<boolean factor> ::=
        [ NOT ] <boolean test>
<boolean test> ::=
        <boolean primary> [ IS [ NOT ] <truth value> ]
<truth value> ::=
        TRUE
      ¦ FALSE
      ¦ UNKNOWN
<boolean primary> ::=
        <column>
        <condition>
      ¦ <boolean predicand>
<boolean predicand> ::=
        <parenthesized boolean value expression>
      ¦ <nonparenthesized value expression primary>
<parenthesized boolean value expression> ::=
        <left paren> <boolean value expression> <right paren>
```

Description

<boolean value expression> describes a boolean value. <boolean primary> with boolean value are <column>, <condition>, and <boolean predicand>. <column> should be declared as BOOLEAN type, and it is allowed to return a boolean value using CAST.

<boolean value expression> can use logical operators such as AND, OR, NOT, and the dedicated operators of boolean value such as IS, IS NOT are also supported.

IS operator and IS NOT operator which are described in <boolean test> determine whether the boolean value described in <boolean primary> matches with one of the <truth value> (TRUE, FALSE, UNKNOWN).

For more information, refer to **Conditions**.

Example

```
gSQL> SELECT * FROM T1 WHERE CAST('TRUE' AS BOOLEAN);
I1
-----
TRUE
```

```
FALSE
null
3 rows selected.
gSQL> SELECT * FROM T1 WHERE I1;
I1
----
TRUE
1 row selected.
gSQL> SELECT * FROM T1 WHERE I1 IS TRUE;
I1
----
TRUE
1 row selected.
gSQL> SELECT * FROM T1 WHERE I1 IS NOT FALSE;
I1
----
TRUE
null
2 rows selected.
gSQL> SELECT * FROM T1 WHERE I1 IS UNKNOWN;
I1
----
null
1 row selected.
```

CASE Expression

Syntax

```
<case expression> ::=
        <simple case>
      ¦ <searched case>
<simple case> ::=
        CASE expr WHEN comparison_expr THEN result
                 [ WHEN comparison_expr THEN result ... ]
                 [ ELSE result ]
        END
<searched case> ::=
        CASE WHEN condition THEN result
```

```
            [ WHEN condition THEN result ... ]
            [ ELSE result ]
        END
```

Description

WHEN ... THEN clause is evaluated in the order of which is described in the CASE statement.
If a comparison result is FALSE, the subsequent WHEN ... THEN clauses are evaluated until TRUE comes up.
If a comparison result is TRUE, the result is returned, and the evaluation is not executed any more.

- Simple case
  The comparison_expr of CASE expr and WHEN ... THEN clause is evaluated as the equal operation. (expr = comparison_expr).
- Searched case
  The condition of WHEN ... THEN clause is evaluated.

If all evaluation results of the WHEN clause are FALSE, then result of ELSE clause is returned.
If ELSE clause is omitted, NULL is returned as a result.

If there are multiple types of THEN or ELSE clause results, the result type is determined by **Result Type Combination Rule**.

For more information, refer to the followings.
- **COALESCE**
- **NULLIF**

Example

- Simple case

```
gSQL> SELECT I1,
            CASE I1 WHEN 1 THEN 'ONE'
                    WHEN 2 THEN 'TWO'
                    ELSE 'NUMBER'
            END AS CASE_RESULT1,
            CASE I1 WHEN 1 THEN 'ONE'
                    WHEN 2 THEN 'TWO'
            END AS CASE_RESULT2
        FROM T1;
I1 CASE_RESULT1 CASE_RESULT2
```

```
-- ------------ -----------
 1 ONE          ONE
 2 TWO          TWO
 3 NUMBER       null
3 rows selected.
```

- Searched case

```
gSQL> SELECT I1,
             CASE WHEN I1 = 1 THEN 'ONE'
                  WHEN I1 = 2 THEN 'TWO'
                  ELSE 'NUMBER'
             END AS CASE_RESULT1,
             CASE WHEN I1 = 1 THEN 'ONE'
                  WHEN I1 = 2 THEN 'TWO'
             END AS CASE_RESULT2
        FROM T1;
I1 CASE_RESULT1 CASE_RESULT2
-- ------------ ------------
 1 ONE          ONE
 2 TWO          TWO
 3 NUMBER       null
3 rows selected.
```

CAST Specification

Syntax

```
CAST( expression AS data_type )
```

Description

CAST converts the expression data type to the data type of the specified data_type.

Example

```
gSQL> SELECT CAST( '1-2' AS INTERVAL YEAR TO MONTH ) AS RESULT FROM DUAL;
RESULT
------
```

```
+01-02
1 row selected.
```

Scalar Subquery Expression

Scalar subquery expression is a subquery which returns a single row with one column as a result. The scal ar subquery expression result is the values described in select list of the subquery.

If the subquery does not return any row, then the result value is NULL, and if it returns two or more rows, then an error occurs.

Scalar subquery expression can be described on most position which describes expression. The subquery s hould be enclosed in parentheses. Even when scalar subquery expression is used as a function argument and the scalar subquery expression is enclosed in parentheses, other parentheses for the subquery is requ ired regardless of the function parentheses. Otherwise, an error occurs.

The following is an example of using scalar subquery expression.

```
gSQL> select * from dual where dummy = (select * from dual);
DUMMY
-----
X
1 row selected.
gSQL> select sum(select 1 from dual) from dual;
ERR-42000(40000): syntax error
select sum(select 1 from dual) from dual
...........^    ^
Error at line 1
gSQL> select sum((select 1 from dual)) from dual;
SUM((SELECT 1 FROM DUAL))
-------------------------
                        1
1 row selected.
```

Compatibility

The SQL standard compatibility for expression is as follows.

Table 11-28 SQL standard compatibility for expression

Feature ID	Description	Availability
E121-03	Value expressions in ORDER BY clause	O
F051-05	Basic date and time Explicit CAST between datetime types and character string types	O
F201	CAST function	O
F261-01	Simple CASE	O
F261-02	Searched CASE	O
F261-03	NULLIF	O
F261-04	COALESCE	O
F263	Comma-separated predicates in simple CASE expression	X
F301	CORRESPONDING in query expressions	X
F385	Drop column generation expression clause	X
F561	Full value expressions	X
F846	Octet support in regular expression operators	X
F847	Nonconstant regular expressions	X
F850	Top-level <order by clause> in <query expression>	O
F855	Nested <order by clause> in <query expression>	O
F856	Nested <fetch first clause> in <query expression>	O
F857	Top-level <fetch first clause> in <query expression>	O
F861	Top-level <result offset clause> in <query expression>	O
F863	Nested <result offset clause> in <query expression>	O
S091-03	Arrays expressions	X
S111	ONLY in query expressions	X
T121	WITH (excluding RECURSIVE) in query expression	O
T581	Regular expression substring function	X

11.5 Pseudo Columns

Pseudo column is not only similar to function, but also it is similar to table column because it can return di fferent value in row unit every time the pseudo column is executed.

Table 11-29 Supported pseudo column

Name	Description	Refer to
CURRVAL	It is a pseudo column which is related to a sequenc e.	**CURRVAL**
NEXTVAL	It is a pseudo column which is related to a sequenc e.	**ROWID Pseudo Column**
ROWNUM	It is the row number which satisfies the condition.	**ROWNUM**
ROWID	It returns the record identifier in database.	**ROWID Pseudo Column**
CLUSTER_GROUP_ID	It returns the record identifier in database.	**CLUSTER_GROUP_ID Pseudo Colum n**
CLUSTER_MEMBER_ID	It returns the member identifier in which the recor d is stored.	**CLUSTER_MEMBER_ID Pseudo Colu mn**
CLUSTER_GROUP_NA ME	It returns the group name in which the record is st ored.	**CLUSTER_GROUP_NAME Pseudo Co lumn**
CLUSTER_MEMBER_N AME	It returns the member name in which the record is stored.	**CLUSTER_MEMBER_NAME Pseudo Column**
CLUSTER_SHARD_ID	It returns the shard identifier in which the record is stored.	**CLUSTER_SHARD_ID Pseudo Colum n**

ROWID Pseudo Column

ROWID pseudo column is a record identifier, and it returns the identification information of each databas e record.

ROWID has the following information to identify the location within the database depending on the syste m.

Standalone system
- OBJECT_ID
- TABLESPACE_ID
- PAGE_ID
- OFFSET in PAGE

Cluster system
- GRID_BLOCK_SEQUENCE
- GRID_BLOCK_ID
- MEMBER_ID
- SHARD_ID

The information stored inside in base 64 encoding is converted into the value such as A-Z, a-z, 0-9, +, / th en output when querying ROWID.

Each information to identify the address within database stored in ROWID can be obtained using the RO WID-related functions.

The address of the deleted record can be newly reassigned to the record to be inserted.

ROWID pseudo column can be used only in SELECT operation, but it can not be used in INSERT, UPDATE, DELETE operations.

For more information, refer to **ROWID**, **ROWID-related Functions**.

The following is an example of querying ROWID pseudo column.

```
gSQL> SELECT ROWID FROM T1;
                  ROWID
----------------------
AAAAAAAAFpEAACAAAEAkAAA
AAAAAAAAFpEAACAAAEAkAAB
AAAAAAAAFpEAACAAAEAkAAC
AAAAAAAAFpEAACAAAEAkAAD
AAAAAAAAFpEAACAAAEAkAAE
5 rows selected.
```

CLUSTER_GROUP_ID Pseudo Column

CLUSTER_GROUP_ID pseudo column returns the group identifier of a server in which the record is stored.

CLUSTER_GROUP_ID pseudo column can perform the SELECT, but it can not perform the INSERT, UPDAT E, or DELETE.

This information in valid in the cluster system.

The following is an example of retrieving CLUSTER_GROUP_ID pseudo column.

```
gSQL> SELECT T1.C1, T1.CLUSTER_GROUP_ID FROM T1;
C1 T1.CLUSTER_GROUP_ID
-- --------------------
A                     1
B                     2
C                     3
3 rows selected.
```

CLUSTER_MEMBER_ID Pseudo Column

CLUSTER_MEMBER_ID pseudo column returns the member identifier of a server in which the record is st ored.

CLUSTER_MEMBER_ID pseudo column can perform the SELECT, but it can not perform the INSERT, UPD ATE, or DELETE.


This information in valid in the cluster system.

The following is an example of retrieving CLUSTER_MEMBER_ID pseudo column.

```
gSQL> SELECT T1.C1, T1.CLUSTER_MEMBER_ID FROM T1;
C1 T1.CLUSTER_MEMBER_ID
-- --------------------
A                     1
B                     3
C                     5
3 rows selected.
```

CLUSTER_GROUP_NAME Pseudo Column

CLUSTER_GROUP_NAME pseudo column returns the group name of a server in which the record is store d.

CLUSTER_GROUP_NAME pseudo column can perform the SELECT, but it can not perform the INSERT, UP DATE, or DELETE.


This information in valid in the cluster system.

The following is an example of retrieving CLUSTER_GROUP_NAME pseudo column.

```
gSQL> SELECT T1.C1, T1.CLUSTER_GROUP_NAME FROM T1;
C1 T1.CLUSTER_GROUP_NAME
-- --------------------
A  G1
B  G2
C  G3
3 rows selected.
```

CLUSTER_MEMBER_NAME Pseudo Column

CLUSTER_MEMBER_NAME pseudo column returns the member name of a server in which the record is st ored.

CLUSTER_MEMBER_NAME pseudo column can perform the SELECT, but it can not perform the INSERT, UPDATE, or DELETE.


This information in valid in the cluster system.

The following is an example of retrieving CLUSTER_MEMBER_NAME pseudo column.

```
gSQL> SELECT T1.C1, T1.CLUSTER_MEMBER_NAME FROM T1;
C1 T1.CLUSTER_MEMBER_NAME
-- ----------------------
A  G1N1
B  G2N1
C  G3N1
3 rows selected.
```

CLUSTER_SHARD_ID Pseudo Column

CLUSTER_SHARD_ID pseudo column returns the shard identifier in which the record is stored.

CLUSTER_SHARD_ID pseudo column is allowed for SELECT only, but it is not allowed for INSERT, UPDATE, or DELETE.


This information in valid in the cluster system.

The following is an example of retrieving CLUSTER_SHARD_ID pseudo column.

```
gSQL> SELECT T1.C1, T1.CLUSTER_SHARD_ID FROM T1;
C1 CLUSTER_SHARD_ID
-- ----------------
A                14
B                17
C                 4
3 rows selected.
```

Compatibility

The SQL standard compatibility for pseudo column is as follows.

Table 11-30 SQL standard compatibility for pseudo column

Feature ID	Description	Availability
T176	Sequence generator support	O
T177	Sequence generator support: simple restart option	O

11.6 Operators

An operator is represented by one or more specific symbols or keywords, and it performs an operation using one or more arguments.

The operator types are various as follows.

- Arithmetic operator
- Concatenation operator
- Set operator

Arithmetic Operator

Syntax

```
<arithmetic operator> ::=
        <value term>
      ¦ <expression> + <value term>
      ¦ <expression> - <value term>
<value term> ::=
        <value factor>
      ¦ <value term> * <value factor>
      ¦ <value term> / <value factor>
<value factor> ::=
        <expression>
      ¦ + <expression>
      ¦ - <expression>
```

Description

An arithmetic operator performs an arithmetic operation of the numeric types, date/time types or interval types.

The arithmetic operator precedence is as follows.

1. **+ (POSITIVE), - (NEGATIVE)**
2. *** (MULTIPLICATION), / (DIVISION)**
3. **+ (ADDITION), - (SUBTRACTION)**

Concatenation Operator

Syntax

```
<concatenation operator> ::=
        <expression> || <expression>
```

Description

A concatenation operator returns strings which connect between values of CHARACTER STRING type or BINARY STRING type.
For more information, refer to **|| (CONCATENATE)**, **CONCATENATE**.

Set Operator

Syntax

```
<set operator> ::=
        <set operator term>
      | <subquery> UNION [ ALL | DISTINCT ] <set operator term>
      | <subquery> EXCEPT [ ALL | DISTINCT ] <set operator term>
      | <subquery> MINUS [ ALL | DISTINCT ] <set operator term>
<set operator term> ::=
        <subquery>
      | <subquery> INTERSECT [ ALL | DISTINCT ] <set operator term>
```

Description

set operator performs a set operation of the subquery results.

INTERSECT ALL/DISTINCT has a higher precedence than other set operators.

Table 11-31 Set operators

Operator	Description
UNION ALL	It is the union which does not exclude duplicated rows of the subquery result.
UNION DISTINCT	It is the union which excludes duplicated rows of the subquery result.
EXCEPT ALL	It is the difference set which does not exclude duplicated rows of the subquery result.
EXCEPT DISTINCT	It is the difference set which excludes duplicated rows of the subquery result.

Operator	Description
MINUS ALL	It is as same as EXCEPT ALL.
MINUS DISTINCT	It is as same as EXCEPT DISTINCT.
INTERSECT ALL	It is the intersection which does not exclude duplicated rows of the subquery result.
INTERSECT DISTINCT	It is the intersection which excludes duplicated rows of the subquery result.

Compatibility

The SQL standard compatibility for operator is as follows.

Table 11-32 SQL standard compatibility for operator

Feature ID	Description	Availability
E011-04	Arithmetic operators	O
E021-07	Character concatenation	O
E071-01	UNION DISTINCT table operator	O
E071-02	UNION ALL table operator	O
E071-03	EXCEPT DISTINCT table operator	O
E071-05	Columns combined via table operators need not have exactly the same data type	O
E071-06	Table operators in subqueries	O
F041-08	All comparison operators are supported (rather than just =)	O
F302-01	INTERSECT DISTINCT table operator	O
F302-02	INTERSECT ALL table operator	O
F304	EXCEPT ALL table operator	O
F846	Octet support in regular expression operators	X
J571	NEW operator	X

11.7 Functions

Functions and operators are similar in features. However, to represent arguments, functions use parentheses after its name. A function can have zero or more arguments.

The function has two types as follows.
. • Single row function
• Aggregate function

Single Row Function

Single row function creates a single result row for each row in the table or view.

The single row functions are as follows.

- Numeric function
- Character string function returning character values
- Character string function returning number values
- Datetime function
- General comparison function
- Conversion function
- Conditional function
- NULL-related function
- ROWID-related function
- Encryption function
- System information function

Numeric Functions

A numeric value is input in numeric function, and the numeric function returns a numeric result.

For more information about the numeric function types, refer to the followings.

- **ABS**
- **ACOS**
- **ASIN**
- **ATAN**
- **ATAN2**
- **BITAND**

- BITNOT
- BITOR
- BITXOR
- CBRT
- CEIL
- COS
- COT
- DEGREES
- EXP
- FACTORIAL
- FLOOR
- LN
- LOG
- MOD
- PI
- POWER
- RADIANS
- RANDOM
- ROUND(number)
- SHARD_ID
- SHIFT_LEFT
- SHIFT_RIGHT
- SIGN
- SIN
- SQRT
- TAN
- TRUNC(number)
- WIDTH_BUCKET

Character String Functions Returning Character Values

A character string type value is input in character string functions returning character values, and the func tion returns the result of character string type.

For more information about character string functions returning character values types, refer to the follo wings.

- CHR
- CONCAT
- CONCATENATE
- INITCAP
- LOWER

- LPAD
- LTRIM
- OVERLAY
- REPEAT
- REPLACE
- REVERSE
- RPAD
- RTRIM
- SPLIT_PART
- SUBSTR
- SUBSTRB
- TRANSLATE
- TRIM
- UPPER

Character String Functions Returning Number Values

A character string type value is input in character string functions returning number values the value, and the function returns the result of number type.

For more information about character string functions returning number values types, refer to the followi ngs.

- ASCII
- BIT_LENGTH
- BYTE_LENGTH
- CHAR_LENGTH
- INSTR
- LENGTH
- LENGTHB
- OCTET_LENGTH
- POSITION

Datetime Functions

The value of date/time/timestamp/interval type is input in datetime function, and the function returns the result of date/time/timestamp/interval type.

For more information about datetime functions types, refer to the followings.

- ADDDATE
- ADDTIME

- ADD_MONTHS
- DATEADD
- DATEDIFF
- DATE_ADD
- DATE_PART
- EXTRACT
- FROM_TZ
- LAST_DAY
- MONTHS_BETWEEN

General Comparison Functions

General comparison function returns a minimum value or a maximum value for the value set.

For more information about general comparison function types, refer to the followings.

- GREATEST
- LEAST

Conversion Functions

Conversion function sets the value of a particular data type.

For more information about conversion function types, refer to the followings.

- NUMTODSINTERVAL
- NUMTOYMINTERVAL
- TO_CHAR(datetime)
- TO_CHAR(number)
- TO_DATE
- TO_NATIVE_BIGINT
- TO_NATIVE_DOUBLE
- TO_NATIVE_INTEGER
- TO_NATIVE_REAL
- TO_NATIVE_SMALLINT
- TO_NUMBER
- TO_TIME
- TO_TIME_TZ
- TO_TIME_WITH_TIME_ZONE
- TO_TIMESTAMP
- TO_TIMESTAMP_TZ
- TO_TIMESTAMP_WITH_TIME_ZONE

Conditional Functions

Conditional function returns a result of specific value depending on a condition.

For more information about conditional function types, refer to the followings.

- CASE2
- DECODE

NULL-related Functions

NULL-related function returns a result of specific value depending on whether the input value is a NULL v alue.

For more information about null-related function types, refer to the followings.

- COALESCE
- NULLIF
- NVL
- NVL2

ROWID-related Functions

ROWID-related function is used to obtain information about the ROWID.

For more information about ROWID-related function types, refer to the followings.

- Valid functions in stand-alone
  - ROWID_OBJECT_ID
  - ROWID_TABLESPACE_ID
  - ROWID_PAGE_ID
  - ROWID_ROW_NUMBER

- Valid functions in cluster
  - ROWID_GRID_BLOCK_ID
  - ROWID_GRID_BLOCK_SEQ
  - ROWID_MEMBER_ID
  - ROWID_SHARD_ID

Encryption Functions

Encryption function encrypt, decrypts, or hashes the given plain text by using the specific algorithm, then returns the result.

For more information about the encryption function, refer to **DIGEST**.

System Information Functions

System information function is used to obtain information about sessions and the system.

For more information about system information function type, refer to the followings.

- CLOCK_DATE
- CLOCK_LOCALTIME
- CLOCK_LOCALTIMESTAMP
- CURRENT_CATALOG
- CURRENT_DATE
- CURRENT_SCHEMA
- CURRENT_TIME
- CURRENT_TIMESTAMP
- CURRENT_USER
- LAST_IDENTITY_VALUE
- LOCALTIME
- LOCALTIMESTAMP
- LOGON_USER
- SESSION_ID
- SESSION_SERIAL
- SESSION_USER
- SESSIONTIMEZONE
- STATEMENT_DATE
- STATEMENT_LOCALTIME
- STATEMENT_LOCALTIMESTAMP
- STATEMENT_TIME
- STATEMENT_TIMESTAMP
- STATEMENT_VIEW_SCN
- SYSDATE
- SYSTIME
- SYSTIMESTAMP
- TRANSACTION_DATE
- TRANSACTION_LOCALTIME
- TRANSACTION_LOCALTIMESTAMP
- TRANSACTION_TIME
- TRANSACTION_TIMESTAMP
- USER_ID
- VERSION

Aggregate Function

Aggregate function creates a single result row for multiple rows.

For more information about aggregate function types, refer to the followings.

- COUNT
- COUNT(*)
- SUM
- AVG
- MIN
- MAX
- STDDEV
- STDDEV_POP
- STDDEV_SAMP
- VAR_POP
- VAR_SAMP
- VARIANCE

Window Function

It returns the result of the function for the defined record range.

The defined record range is called as a window, and the execution range is defined in OVER <window name or specification>.

For more information about the window, refer to **window clause**.

Each record within the group has the result of executing the window function for the window (defined range). Therefore, unlike aggregate function, the window function returns multiple records for each group.

The window function is available in *select list* and *order by* clause.

Window functions are as follows.

- AVG() OVER
- CORR() OVER
- COUNT() OVER
- COUNT(*) OVER
- COVAR_POP() OVER
- COVAR_SAMP() OVER

- CUME_DIST() OVER
- DENSE_RANK() OVER
- FIRST() OVER
- FIRST_VALUE() OVER
- LAG() OVER
- LAST() OVER
- LAST_VALUE() OVER
- LEAD() OVER
- LISTAGG() OVER
- MAX() OVER
- MEDIAN() OVER
- MIN() OVER
- NTH_VALUE() OVER
- NTILE() OVER
- PERCENT_RANK() OVER
- PERCENTILE_CONT() OVER
- PERCENTILE_DISC() OVER
- RANK() OVER
- RATIO_TO_REPORT() OVER
- REGR_AVGX() OVER
- REGR_AVGY() OVER
- REGR_COUNT() OVER
- REGR_INTERCEPT() OVER
- REGR_R2() OVER
- REGR_SLOPE() OVER
- REGR_SXX() OVER
- REGR_SXY() OVER
- REGR_SYY() OVER
- ROW_NUMBER() OVER
- STDDEV() OVER
- STDDEV_POP() OVER
- STDDEV_SAMP() OVER
- STRING_AGG() OVER
- SUM() OVER
- VAR_POP() OVER
- VAR_SAMP() OVER
- VARIANCE() OVER

Compatibility

The SQL standard compatibility for function is as follows.

Table 11-33 SQL standard compatibility for function

Feature ID	Description	Availability
B033	Untyped SQL-invoked function arguments	X
E021-04	CHARACTER_LENGTH function	O
E021-05	OCTET_LENGTH function	O
E021-06	SUBSTRING function	O
E021-08	UPPER and LOWER functions	O
E021-09	TRIM function	O
E021-11	POSITION function	O
E091-01	AVG	O
E091-02	COUNT	O
E091-03	MAX	O
E091-04	MIN	O
E091-05	SUM	O
E091-06	ALL quantifier	O
E091-07	DISTINCT quantifier	O
F131-03	Set functions supported in queries with grouped views	O
F201	CAST function	O
F441	Extended set function support	O
F442	Mixed column references in set functions	X
F801	Full set function	X
F842	OCCURRENCES_REGEX function	X
F843	POSITION_REGEX function	X
S071	SQL paths in function and type name resolution	X
S201-02	Array as result type of functions	X
S211	User-defined cast functions	X
S241	Transform functions	X
T041-03	POSITION, LENGTH, LOWER, TRIM, UPPER, and SUBSTRING functions for LOB data types	X
T312	OVERLAY function	O
T321-01	User-defined functions with no overloading	O
T326	Table functions	X
T341	Overloading of SQL-invoked functions and SQL-invoked procedures	X
T433	Multiargument GROUPING function	X
T441	ABS and MOD functions	O
T571	Array-returning external SQL-invoked functions	X

Feature ID	Description	Availability
T572	Multiset-returning external SQL-invoked functions	X
T581	Regular expression substring function	X
T614	NTILE function	O
T615	LEAD and LAG functions	O
T616	Null treatment option for LEAD and LAG functions	O
T617	FIRST_VALUE and LAST_VALUE functions	O
T618	NTH_VALUE function	O
T619	Nested window functions	X
T621	Enhanced numeric functions	O

11.8 Conditions

Condition

Condition is an expression which is evaluated as TRUE, FALSE, UNKNOWN.

Condition can be used in the following SQL statements.
- WHERE clauses in DELETE, UPDATE statements
- WHERE and HAVING clauses in SELECT statement
- Where the BOOLEAN TYPE can be used

The condition types are as follows.
- Comparison condition
- Logical condition
- Null condition
- Compound condition
- Pattern-matching condition
- Between condition
- In condition
- Exists condition
- Distinct condition

Table 11-34 Condition precedence

Precedence	Condition type
1	Operators in condition clauses
2	=, !=, <, >, <=, >=
3	IS [NOT] NULL, [NOT] BETWEEN, [NOT] IN, LIKE, EXISTS, IS [NOT] DISTINCT FROM
4	NOT
5	AND
6	OR

Comparison Conditions

It compares both conditional expressions, and returns the boolean type of TRUE, FALSE, UNKNOWN valu es.

Table 11-35 Comparison conditions

Condition	Description
=	It checks if both conditions are equal.
!=, <>	It checks if both conditions are not equal.
>	It compares which one of both conditions is bigger.
<	It compares which one of both conditions is smaller.
>=	It compares which one of both conditions is bigger or equal.
<=	It compares which one of both conditions is smaller or equal.
ANY, SOME	If there is a condition whose left expr satisfies at least one of right expr_list (or subque ry results), then it returns TRUE. If there is not right subquery result, then it returns FALSE.
ALL	If there is a condition whose left expr satisfies all right expr_list (or subquery results), t hen it returns TRUE. If there is not right subquery result, then it returns TRUE.

For more information, refer to **Type Comparison**.

< Simple Comparison Conditions >

Syntax

```
<simple_comparison_condition> ::=
        <expr>          <comparison_operator> <expr>
      ¦ <expr>          <comparison_operator> ( <subquery> )
      ¦ ( <subquery> )  <comparison_operator> <expr>
      ¦ ( <subquery> )  <comparison_operator> ( <subquery> )
      ¦ ( <expr_list> ) <comparison_operator> ( <expr_list> )
      ¦ ( <expr_list> ) <comparison_operator> ( <subquery> )
      ¦ ( <subquery> )  <comparison_operator> ( <expr_list> )
      ¦ ( <subquery> )  <comparison_operator> ( <subquery> )
<comparison_operator> ::=
        <  =   >
      ¦ <  !=  >
      ¦ <  <   >
      ¦ <  >   >
```

```
      ¦ <  <=  >
      ¦ <  >=  >
<expr_list> ::=
        <expr>
      ¦ <expr>, ... , <expr>
      ¦ ( <expr> )
      ¦ ( <expr> , ... , <expr> )
```

For more information, refer to **Scalar Subquery Expression**.

Description

If the expr list or subquery comes to both left and right of comparison_operator, then the number of expr or subquery target to be compared should be same.
If there is a subquery, the number of result record should be one.

Example

Table 11-36 Example of simple comparison conditions

Conditional expression	Result
'abc' = 'abc'	TRUE
'abc' != 'abc'	FALSE
'abc' < 'abc'	FALSE
'abc' <= 'abc'	TRUE
'abc' > 'abc'	FALSE
'abc' >= 'abc'	TRUE
(1, 2, 3) = (1, 2, 3)	TRUE
(1, 2, 3) = (1, 2, 4)	FALSE
(1, 2, 3) != (4, 5, 6)	TRUE
(1, 2, 3) != (1, 2, 3)	FALSE
(1, 2, 3) < (1, 2, 4)	TRUE
(1, 2, 3) < (1, 2, 3)	FALSE
(1, 2, 3) <= (1, 2, 4)	TRUE
(1, 2, 3) <= (1, 2, 2)	FALSE
(1, 2, 3) > (1, 2, 2)	TRUE
(1, 2, 3) > (1, 2, 4)	FALSE
(1, 2, 3) >= (1, 2, 2)	TRUE
(1, 2, 3) >= (1, 2, 4)	FALSE

〈Group Comparison Conditions〉

Syntax

```
〈group_comparison_condition〉 ::=
   〈expr〉          〈comparison_operator〉 〈quantifier〉 ( 〈expr_list〉 )
 ¦ 〈expr〉          〈comparison_operator〉 〈quantifier〉 ( 〈subquery〉 )
 ¦ ( 〈expr_list〉 ) 〈comparison_operator〉 〈quantifier〉 ( 〈expr_list_list〉 )
 ¦ ( 〈expr_list〉 ) 〈comparison_operator〉 〈quantifier〉 ( 〈subquery〉 )
 ¦ ( 〈subquery〉 )  〈comparison_operator〉 〈quantifier〉 ( 〈expr_list〉 )
 ¦ ( 〈subquery〉 )  〈comparison_operator〉 〈quantifier〉 ( 〈expr_list_list〉 )
 ¦ ( 〈subquery〉 )  〈comparison_operator〉 〈quantifier〉 ( 〈subquery〉 )
〈comparison_operator〉 ::=
        〈 =  〉
      ¦ 〈 !=  〉
      ¦ 〈 〈  〉
      ¦ 〈 〉  〉
      ¦ 〈 〈=  〉
      ¦ 〈 〉=  〉
〈quantifier〉 ::=
        ALL
      ¦ ANY
      ¦ SOME
〈expr_list〉 ::=
        〈expr〉
      ¦ 〈expr〉, ... , 〈expr〉
      ¦ ( 〈expr〉 )
      ¦ ( 〈expr〉 , ... , 〈expr〉 )
〈expr_list_list〉 ::=
        〈expr_list〉
      ¦ 〈expr_list〉, ... , 〈expr_list〉
```

For more information, refer to **Scalar Subquery Expression**.

Description

If the expr list or subquery comes to both left and right of comparison_operator, then the number of expr or subquery target to be compared should be same.
If a subquery comes to the left of comparison_operator, the number of result record should be one.
If a subquery comes to the right of comparison_operator, the number of result record can be multiple.

Example

Table 11-37 Example of group comparison conditions

Conditional expression	Result
1 =any (1, 2, 3, 4, 5)	TRUE
1 =any (1, 2, null, 4, 5)	TRUE
1 =any (2, null, 4, 5)	NULL
1 =any (100, 2, 3, 4, 5)	FALSE
1 =all (1, +1, 1E+0)	TRUE
1 =all (1, +1, 1E+0, null)	NULL
1 =all (1, 2, 3, 4, 5)	FALSE
(1, 2) =any ((0, 1), (1, 2), (3, 4))	TRUE
(1, 2) =any ((0, 1), (1, 2), (null, null))	TRUE
(1, 2) =any ((0, 1), (2, 3), (3, 4))	FALSE
(1, 2) =all ((1, 2), (+1, +2), (1E+0, 2E+0))	TRUE
(1, 2) =all ((1, 2), (+1, +2), (null, null))	NULL
(1, 2) =all ((0, 1), (2, 3), (3, 4))	FALSE
When the result record of comparison_operator's right subquery is 0	
('X') =any (select dummy from dual where dummy = 'Y')	FALSE
('X') =all (select dummy from dual where dummy = 'Y')	TRUE

Logical Conditions

Logical conditions are such as AND, OR, NOT.

AND

Syntax

```
<boolean value expression> AND <boolean value expression>
```

Description

Table 11-38 Truth table of AND boolean operator

AND	True	False	Unknown
True	True	False	Unknown
False	False	False	False
Unknown	Unknown	False	Unknown

OR

Syntax

```
<boolean value expression> OR <boolean value expression>
```

Description

Table 11-39 Truth table of OR boolean operator

OR	True	False	Unknown
True	True	True	True
False	True	False	Unknown
Unknown	True	Unknown	Unknown

NOT

Syntax

```
NOT <boolean value expression>
```

Description

Table 11-40 Truth table of NOT boolean operator

expr	NOT
True	False
False	True
Unknown	Unknown

Null Condition

Syntax

```
<expr> IS [NOT] NULL
```

Description

It checks whether the result value of expr is NULL.

Table 11-41 Result table of IS NULL condition

expr	IS NULL	IS NOT NULL
NULL	True	False
NOT NULL	False	True

Compound Conditions

It is a conditional expression in which multiple conditions are combined.

```
compound_condition ::=
        ( condition )
      ¦ NOT condition
      ¦ condition < AND ¦ OR > condition
```

Pattern-matching Conditions

Like Condition

Syntax

```
like_condition ::=
        string [NOT] LIKE pattern [ ESCAPE escape_character ]
```

Description

It checks if a string matches the specified pattern.

Arguments such as string, pattern, escape_character can be of a character type such as CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING, or of the type which is available to be converted to a character type.
If string, pattern, escape_character are NULL, it returns NULL.

If escape_character is omitted, there is not a default value.
If escape_character is specified, the escape_character should be one character.

If pattern does not include '_' nor '%', it is processed in the same way as equal operation(string = pattern)
.
If pattern includes '_' or '%', the string checks if it matches as follows.

- '_': If it corresponds to one arbitrary character.
- '%': If it corresponds to the arbitrary character string which has zero or more characters.

Use ESCAPE syntax to compare '_' or '%' included in the pattern with characters.
Specify escape_character, and describe the specified escape_character before the pattern's '_' or '%'.

Example

```
gSQL> SELECT 'hello%' LIKE 'h%o!%' ESCAPE '!' AS RESULT FROM DUAL;
RESULT
------
TRUE
```

- 'represent' LIKE 'represent'   => TRUE
- 'represent' LIKE ' represent ' => FALSE
- 'represent' LIKE 'REPRESENT'   => FALSE
- 'represent' LIKE 'r_pr_s_nt'   => TRUE
- 'represent' LIKE 're%t'   => TRUE
- 'represent' LIKE 'rep'   => FALSE
- 'summer_vacation' LIKE 'summer_vacation' ESCAPE '\' => TRUE
- NULL LIKE 'summer_vacation' ESCAPE '\'   => NULL
- 'summer_vacation' LIKE NULL ESCAPE '\'   => NULL
- 'summer_vacation' LIKE 'summer_vacation' ESCAPE NULL => NULL

BETWEEN Condition

Syntax

```
<between condition> ::=
   <expr1> [ NOT ] BETWEEN [ ASYMMETRIC ¦ SYMMETRIC ] <expr2> AND <expr3>
```

Description

It checks whether expr1 is within the range between expr2 and expr3.

If ASYMMETRIC or SYMMETRIC is omitted, the default is ASYMMETRIC.
If data types among expr1, expr2, expr3 are different, they are converted.

For more information, refer to **Type Comparison**, **Type Conversion**.

Table 11-42 Equivalence of between conditions

A	B
X BETWEEN ASYMMETRIC Y AND Z	X BETWEEN Y AND Z
X BETWEEN Y AND Z	X >= Y AND X <= Z
X NOT BETWEEN Y AND Z	NOT(X BETWEEN Y AND Z)
X BETWEEN SYMMETRIC Y AND Z	((X BETWEEN Y AND Z) OR (X BETWEEN Z AND Y)
X NOT BETWEEN SYMMETRIC Y AND Z	NOT(X BETWEEN SYMMETRIC Y AND Z)

Example

Table 11-43 Example of between condition

Conditional expression		Result
BETWEEN [ASYMMETRIC]	3 BETWEEN 1 AND 5	TRUE
	NULL BETWEEN 1 AND 5 3 BETWEEN NULL AND 5 3 BETWEEN 1 AND NULL	NULL
	3 BETWEEN 5 AND 1	FALSE
BETWEEN SYMMETRIC	3 BETWEEN SYMMETRIC 1 AND 5	TRUE
	NULL BETWEEN SYMMETRIC 1 AND 5 3 BETWEEN SYMMETRIC NULL AND 5 3 BETWEEN SYMMETRIC 1 AND NULL	NULL
	3 BETWEEN SYMMETRIC 5 AND 1	TRUE

IN Condition

Syntax

```
<in_condition> ::=
        <expr>          [NOT] IN ( <expr_list> )
      ¦ <expr>          [NOT] IN ( <subquery> )
      ¦ ( <expr_list> ) [NOT] IN ( <expr_list_list> )
      ¦ ( <expr_list> ) [NOT] IN ( <subquery> )
      ¦ ( <subquery> )  [NOT] IN ( <expr_list> )
      ¦ ( <subquery> )  [NOT] IN ( <expr_list_list> )
      ¦ ( <subquery> )  [NOT] IN ( <subquery> )
<expr_list> ::=
        <expr>
```

```
      ¦ <expr>, ... , <expr>
      ¦ ( <expr> )
      ¦ ( <expr> , ... , <expr> )
<expr_list_list> ::=
        <expr_list>
      ¦ <expr_list>, ... , <expr_list>
```

Description

In condition returns the same result as = ANY.
NOT IN condition returns the same result as !=ALL.

For more information, refer to **Comparison Conditions**.

Example

Table 11-44 Example of IN condition

Conditional expression	Result
1 IN (1, 2, 3, 4, 5)	TRUE
1 IN (1, 2, null, 4, 5)	TRUE
1 IN (2, null, 4, 5)	NULL
1 IN (100, 2, 3, 4, 5)	FALSE
NULL IN (1, 2, 3)	NULL
1 NOT IN (2, 3, 4, 5)	TRUE
1 NOT IN (2, null, 4, 5)	NULL
1 NOT IN (1, 2, null, 4, 5)	FALSE
1 NOT IN (100, 2, 3, 4, 5)	TRUE
NULL NOT IN (1, 2, 3)	NULL

EXISTS Condition

Syntax

```
exists_conditions ::=
        EXISTS ( subquery )
```

Description

It checks whether the result record of subquery exists.
If the result record of subquery exists, it returns TRUE. Otherwise, it returns FALSE.

Example

```
gSQL> SELECT * FROM DUAL WHERE EXISTS ( SELECT * FROM DUAL );
      DUMMY
      -----
      X
      1 row selected.
gSQL> SELECT * FROM DUAL
       WHERE EXISTS ( SELECT * FROM DUAL WHERE DUMMY = 'Y' );
      no rows selected.
```

DISTINCT Condition

Syntax

```
distinct_conditions ::=
        <expr> IS [NOT] DISTINCT FROM <expr>
      | ( <expr_list> ) IS [NOT] DISTINCT FROM ( <expr_list> )
```

Description

The operand type of distinct condition should be comparable one another.
If the operand is <expr_list>, then the data in the same position becomes the comparison target.

If all operands of distinct condition is not null value,
is distinct from is as same as *not equal(!=)*,
is not distinct from returns the same result of when it is *eqaul(=)*.

DISTINCT condition processes NULL value as a general data instead of unknown, and this is what is differ ent from other comparing operators.

- IS DISTINCT FROM
  - If all operands are NULL
    - NULL is distinct from NULL => FALSE

    - If one of the operand is NULL
        - NULL is distinct from 1 => TRUE
        - 1 is distinct from NULL => TRUE
    - If all operands are not NULL
        - 1 is distinct from 1 => FALSE
        - 1 is distinct from 2 => TRUE
    - If the operand is <expr_list>
        - (1, 2, 3) is distinct from (1, 2, 3) => FALSE
        - (1, 2, 3) is distinct from (1, 3, 3) => TRUE
        - (1, 2, 3) is distinct from (4, 5, 6) => TRUE

- IS NOT DISTINCT FROM
    - If all operands are NULL
        - NULL is not distinct from NULL => TRUE
    - If one of the operand is NULL
        - NULL is not distinct from 1 => FALSE
        - 1 is not distinct from NULL => FALSE
    - If all operands are not NULL
        - 1 is not distinct from 1 => TRUE
        - 1 is not distinct from 2 => FALSE
    - If the operand is <expr_list>
        - (1, 2, 3) is not distinct from (1, 2, 3) => TRUE
        - (1, 2, 3) is not distinct from (1, 3, 3) => FALSE
        - (1, 2, 3) is not distinct from (4, 5, 6) => FALSE

Example

```
* IS DISTINCT FROM
gSQL>
SELECT i1,
       i2,
       i1 IS DISTINCT FROM i2 AS IsDistinct
  FROM t1;
  I1   I2 ISDISTINCT
---- ---- ----------
   1 null TRUE
   1    1 FALSE
   1    2 TRUE
null null FALSE
null    1 TRUE
null    2 TRUE
```

```
6 rows selected.
gSQL>
SELECT i1,
       i2,
       i3,
       ( I1, I2, I3 ) IS DISTINCT FROM ( 1, 1, 1 ) AS RESULT
  FROM t1;
  I1   I2   I3 RESULT
---- ---- ---- ------
   1    1    1 FALSE
   2 null    3 TRUE
null null null TRUE
3 rows selected.
* IS NOT DISTINCT FROM
gSQL>
SELECT i1,
       i2,
       i1 IS NOT DISTINCT FROM i2 AS IsNotDistinct
  FROM t1;
  I1   I2 ISNOTDISTINCT
---- ---- -------------
   1 null FALSE
   1    1 TRUE
   1    2 FALSE
null null TRUE
null    1 FALSE
null    2 FALSE
6 rows selected.
gSQL>
SELECT i1,
       i2,
       i3,
       ( I1, I2, I3 ) IS NOT DISTINCT FROM ( 1, 1, 1 ) AS RESULT
 FROM t1;
  I1   I2   I3 RESULT
---- ---- ---- ------
   1    1    1 TRUE
   2 null    3 FALSE
null null null FALSE
3 rows selected.
```

Compatibility

The SQL standard compatibility for condition is as follows.

Table 11-45 SQL standard compatibility for condition

Feature ID	Description	Availability
E061-01	Comparison predicate	O
E061-02	BETWEEN predicate	O
E061-03	IN predicate with list of values	O
E061-04	LIKE predicate	O
E061-05	LIKE predicate: ESCAPE clause	O
E061-06	NULL predicate	O
E061-07	Quantified comparison predicate	O
E061-08	EXISTS predicate	O
E061-09	Subqueries in comparison predicate	O
E061-11	Subqueries in IN predicate	O
E061-12	Subqueries in quantified comparison predicate	O
E061-13	Correlated subqueries	O
E061-14	Search condition	O
F051-04	Comparison predicate on DATE, TIME, and TIMESTAMP data types	X
F053	OVERLAPS predicate	X
F263	Comma-separated predicates in simple CASE expression	X
F291	UNIQUE predicate	X
F481	Expanded NULL predicate	O
F841	LIKE_REGEX predicate	X
P008	Comma-separated predicates in a CASE statement Extended CASE	X
S151	Type predicate	X
T141	SIMILAR predicate	X
T151	DISTINCT predicate	X
T152	DISTINCT predicate with negation	X
T461	Symmetric BETWEEN predicate	O
T501	Enhanced EXISTS predicate	O
T631	IN predicate with one list element	X
X090	XML document predicate	X
X091	XML content predicate	X
X141	IS VALID predicate: data-driven case	X
X142	IS VALID predicate: ACCORDING TO clause	X
X143	IS VALID predicate: ELEMENT clause	X
X144	IS VALID predicate: schema location	X
X145	IS VALID predicate outside check constraints	X

Feature ID	Description	Availability
X151	IS VALID predicate with DOCUMENT option	X
X152	IS VALID predicate with CONTENT option	X
X153	IS VALID predicate with SEQUENCE option	X
X155	IS VALID predicate: NAMESPACE without ELEMENT clause	X
X157	IS VALID predicate: NO NAMESPACE with ELEMENT clause	X

12.

SQL Languages

Structured Query Languages (SQL) are classified as follows.

- Data Definition Language
- Data Manipulation Language
- Data Query Language
- Control Language

12.1 Data Definition Language

DDL Related Statements

For more information, refer to the followings.

- Non-schema object DDL
  - **Database Related Statements**
  - **Profile Related Statements**
  - **Audit Policy Related Statement**
  - **Authorization Related Statements**
  - **Schema Related Statements**
  - **Tablespace Related Statements**

- SQL schema object DDL
  - **Table Related Statements**
  - **Index Related Statements**
  - **View Related Statements**
  - **Sequence Related Statements**
  - **Synonym Related Statements**

- Cluster object DDL
  - **Cluster System Related Statements**
  - **Cluster Group Related Statements**
  - **Cluster Member Related Statements**
  - **Cluster Location Related Statements**
  - **Global Secondary Index Related Statements**

Concepts of DDL

Data Definition Language (DDL) is an SQL language which creates, drops and alters SQL objects.

SQL objects of a database are listed in the following table. For more information, refer to the links in the f ollowing table.

Table 12-1 SQL objects types

Object type	Object	Description	Refer to
		It is an object which defines a password man	

Object type	Object	Description	Refer to
Non-schema object	Profile	agement policy.	**Profile**
	Audit policy	It is an object which defines the SQL audit p olicy.	**Audit Policy**
	User	It is a user object which consists of a set of p rivileges.	**Authorization**
	Schema	It is a logical position including SQL schema objects such as tables.	**Schema**
	Tablespace	It is a physical storage of objects such as tabl es, indexes, etc.	**Tablespace**
	Public synonym	It is a public synonym.	**Public Synonym**
SQL schema object	Table	It is a physical relation where the data is stor ed.	**Table**
	View	It is a logical relation which consists of querie s.	**View**
	Index	It is an index objects to improve query perfor mance.	**Index**
	Sequence	It is an object which generate sequence num ber.	**Sequence**
	Synonym	It is an object which declares an alias for an object.	**Synonym**
	Stored procedure	It is an user defined procedure object.	**Stored Procedure**
	Stored function	It is an user defined function object.	**Stored Function**
Cluster object	Cluster group	It is a cluster member set.	**Cluster Group**
	Cluster member	It is a data server which configures a cluster s ystem.	**Cluster Member**
	Cluster location	It is a location object of a cluster member.	**Cluster Location**
	Shard	It is a set of rows which horizontally divides a cluster table.	**Cluster Table and Shard**
	Global secondary index	It is an index for the row identifier of a cluste r.	**Global Secondary Index**

DDL and Transaction

A transaction of GOLDILOCKS includes not only DML statements such as INSERT, DELETE, UPDATE data, but also DDL statement such as CREATE, DROP, ALTER objects. Many DBMS performs implicit transaction s of DDL. On the other hand, GOLDILOCKS includes a DDL statement in the transaction, and it guarantee s the atomicity and consistency of transaction.

This features is useful when a user needs to atomically perform batch DDL such as database migration or tool installation, or to recover a mistake through ROLLBACK when statement such as DROP TABLE or TRU

NCATE TABLE is executed by user mistake.

If the property of DDL statement is auto-commit, then it automatically commits when executing the state ment. On the other hand, if it is not auto-commit, then it can rollback the transaction even after the state ment was executed. Whether the DDL is auto-commit or not is queried by using **V$SQL_COMMAND** view as follows.

```
gSQL>
SELECT command, auto_commit
  FROM V$SQL_COMMAND
 WHERE is_ddl = 'YES';
COMMAND                                                  AUTO_COMMIT
-------------------------------------------------------- -----------
ALTER AUDIT POLICY                                       YES
ALTER CLUSTER GROUP .. ADD CLUSTER MEMBER                YES
ALTER CLUSTER GROUP .. OFFLINE CLUSTER MEMBER            YES
ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS             YES
ALTER DATABASE OFFLINE INACTIVE CLUSTER MEMBERS          YES
ALTER DATABASE RESET LOCAL CLUSTER MEMBER                YES
ALTER DATABASE ADD LOGFILE GROUP                         YES
ALTER DATABASE ADD LOGFILE MEMBER                        YES
ALTER DATABASE DROP LOGFILE GROUP                        YES
ALTER DATABASE DROP LOGFILE MEMBER                       YES
ALTER DATABASE RENAME GLOBAL TRANSACTION LOGFILE         YES
ALTER DATABASE RENAME LOGFILE                            YES
ALTER DATABASE ARCHIVELOG                                YES
ALTER DATABASE NOARCHIVELOG                              YES
ALTER DATABASE DATAFILE AUTOEXTEND ..                    YES
ALTER DATABASE CLEAR PASSWORD HISTORY                    NO
ALTER FUNCTION                                           NO
ALTER INDEX AGING                                        NO
ALTER INDEX .. STORAGE                                   NO
ALTER INDEX .. RENAME                                    NO
ALTER INDEX .. REBUILD                                   YES
ALTER INDEX .. COALESCE                                  NO
ALTER PACKAGE                                            YES
ALTER PROCEDURE                                          NO
ALTER PROFILE                                            YES
ALTER SEQUENCE                                           YES
ALTER SEQUENCE .. SYNCHRONIZE                            YES
ALTER SYSTEM SWITCH LOGFILE                              YES
ALTER TABLE .. ADD COLUMN                                NO
```

```
ALTER TABLE .. SET UNUSED COLUMN                              NO
ALTER TABLE .. ALTER COLUMN .. SET DEFAULT                    NO
ALTER TABLE .. ALTER COLUMN .. DROP DEFAULT                   NO
ALTER TABLE .. ALTER COLUMN .. SET NOT NULL                   NO
ALTER TABLE .. ALTER COLUMN .. DROP NOT NULL                  NO
ALTER TABLE .. ALTER COLUMN .. SET DATA TYPE                  YES
ALTER TABLE .. ALTER COLUMN .. AS IDENTITY                    YES
ALTER TABLE .. ALTER COLUMN .. DROP IDENTITY                  YES
ALTER TABLE .. RENAME COLUMN                                  NO
ALTER TABLE .. STORAGE                                        NO
ALTER TABLE .. ADD CONSTRAINT                                 NO
ALTER TABLE .. ALTER CONSTRAINT                               NO
ALTER TABLE .. DROP CONSTRAINT                                NO
ALTER TABLE .. RENAME CONSTRAINT                              NO
ALTER TABLE .. RENAME TO ..                                   NO
ALTER TABLE .. REBALANCE ..                                   YES
ALTER TABLE .. MOVE SHARD .. TO CLUSTER GROUP ..              YES
ALTER TABLE .. SPLIT SHARD .. INTO .. AT CLUSTER GROUP ..     YES
ALTER TABLE .. MERGE SHARDS .. INTO ..                        YES
ALTER TABLE .. RENAME SHARD .. TO ..                          NO
ALTER TABLE .. ADD SUPPLEMENTAL LOG                           NO
ALTER TABLE .. ADD GLOBAL SECONDARY INDEX                     NO
ALTER TABLE .. ALTER GLOBAL SECONDARY INDEX                   NO
ALTER TABLE .. ALTER GLOBAL SECONDARY INDEX AGING             NO
ALTER TABLE .. DROP GLOBAL SECONDARY INDEX                    NO
ALTER TABLE .. ALTER GLOBAL SECONDARY INDEX REBUILD           YES
ALTER TABLE .. ALTER GLOBAL SECONDARY INDEX COALESCE          NO
ALTER TABLE .. DROP SUPPLEMENTAL LOG                          NO
ALTER TABLE .. READ ONLY                                      YES
ALTER TABLE .. READ WRITE                                     YES
ALTER TABLE .. SYNCHRONIZE IDENTITY COLUMN                    YES
ALTER TABLESPACE .. ADD                                       YES
ALTER TABLESPACE .. DROP                                      YES
ALTER TABLESPACE .. ONLINE                                    YES
ALTER TABLESPACE .. OFFLINE                                   YES
ALTER TABLESPACE .. RENAME TO                                 YES
ALTER TABLESPACE .. RENAME { DATAFILE ¦ MEMORY }              YES
ALTER USER                                                    YES
ALTER USER .. IDENTIFIED BY                                   YES
ALTER VIEW                                                    NO
ANALYZE SYSTEM COMPUTE STATISTICS                             NO
```

```
ANALYZE SYSTEM DELETE STATISTICS                          NO
ANALYZE TABLE .. [COMPUTE¦ESTIMATE] STATISTICS            YES
ANALYZE TABLE .. DELETE STATISTICS                        NO
AUDIT POLICY                                              YES
COMMENT ON .. IS                                          NO
CREATE AUDIT POLICY                                       YES
CREATE CLUSTER GROUP                                      YES
CREATE FUNCTION                                           NO
CREATE INDEX                                              NO
CREATE PACKAGE                                            YES
CREATE PACKAGE BODY                                       YES
CREATE PROCEDURE                                          NO
CREATE PROFILE                                            YES
CREATE SCHEMA                                             YES
CREATE SEQUENCE                                           YES
CREATE SYNONYM                                            NO
CREATE TABLE                                              NO
CREATE TABLE ... AS SELECT                                NO
CREATE TABLESPACE                                         YES
CREATE USER                                               YES
CREATE VIEW                                               NO
DROP AUDIT POLICY                                         YES
DROP CLUSTER GROUP                                        YES
DROP FUNCTION                                             NO
DROP INDEX                                                NO
DROP PACKAGE                                              YES
DROP PROCEDURE                                            NO
DROP PROFILE                                              YES
DROP SCHEMA                                               YES
DROP SEQUENCE                                             YES
DROP SYNONYM                                              NO
DROP TABLE                                                NO
DROP TABLESPACE                                           YES
DROP USER                                                 YES
DROP VIEW                                                 NO
GRANT .. ON DATABASE                                      NO
GRANT .. ON TABLESPACE                                    NO
GRANT .. ON SCHEMA                                        NO
GRANT .. ON TABLE                                         NO
GRANT USAGE ON ..                                         NO
GRANT .. ON PROCEDURE                                     NO
```

```
GRANT .. ON PACKAGE                              NO
NOAUDIT POLICY                                   YES
REVOKE .. ON DATABASE                            NO
REVOKE .. ON TABLESPACE                          NO
REVOKE .. ON SCHEMA                              NO
REVOKE .. ON TABLE                               NO
REVOKE USAGE ON ..                               NO
REVOKE .. ON PROCEDURE                           NO
REVOKE .. ON PACKAGE                             NO
TRUNCATE TABLE                                   NO
PURGE CONSTRAINT                                 NO
PURGE INDEX                                      NO
PURGE TABLE                                      NO
PURGE TABLESPACE                                 YES
PURGE RECYCLEBIN                                 YES
PURGE DBA_RECYCLEBIN                             YES
FLASHBACK TABLE                                  YES
128 rows selected.
```

The followings are examples of COMMIT and ROLLBACK when table-related DDL statements are included in the transactions, and examples of its effects on other transactions. The example shows that the transac tion including DDL guarantees the transaction atomicity. In addition, it ensures the reading consistency of the transaction, which is not affected by other transactions before transaction's commitment or when th e transaction is rolled back.

Creating Object and Transaction

- Before committing or rolling back a transaction of creating table

When a table is created, and transaction is not committed, the table data can be manipulated on the DDL transaction as follows. However, other transactions are not allowed to enquire the table until the creating table transaction is committed. CREATE TABLE statement, like as INSERT statement, can not be queried b y any other transaction until the transaction is committed.

  - Transaction A: Table t1 is created but the transaction is not committed.

```
gSQL> CREATE TABLE t1 ( id INTEGER, name VARCHAR(128) );
Table created.
gSQL> INSERT INTO t1 VALUES ( 1, 'leekmo' );
1 row created.
gSQL> SELECT * FROM t1;
```

```
ID NAME
-- ------
 1 leekmo
1 row selected.
```

As above, if table t1 is created on the transaction A and the transaction is not yet committed, the transact ion B in another session can not enquire the table t1 and can not create the table named as t1.

- Transaction B: Table t1 can not be enquired until committing the transaction A.

```
gSQL> SELECT * FROM t1;
ERR-42000(16040): table or view does not exist :
SELECT * FROM t1
              *
ERROR at line 1:
```

- Table t1 can not be created until the transaction A is rolled back.

```
gSQL> CREATE TABLE t1 ( emp_no INTEGER );
ERR-HYT00(14026): resource busy or timeout expired
```

- After committing a transaction of creating table

If the transaction A is committed, table t1 can be enquired by the transaction B and CREATE TABLE state ment returns a validation error to notify that the table t1 exists as follows.

- Transaction B: After committing the transaction A, the table can be enquired as follows.

```
gSQL> SELECT * FROM t1;
ID NAME
-- ------
 1 leekmo
1 row selected.
```

- After the transaction A is committed, a validation error is returned.

```
gSQL> CREATE TABLE t1 ( emp_no INTEGER );
ERR-42000(16005): name 'PUBLIC.T1' is already used by an existing object :
CREATE TABLE t1 ( emp_no INTEGER )
             *
ERROR at line 1:
```

- After rolling back a transaction of creating table

If the transaction A is rolled back, creating table t1 is also rolled back, and table t1 can be created by the t ransaction B.

- Transaction B: The transaction A is rolled back and its state becomes as same as before creating t able t1.

```
gSQL> SELECT * FROM t1;
ERR-42000(16040): table or view does not exist :
SELECT * FROM t1
              *
ERROR at line 1:
```

- The transaction A is rolled back and it can create the table t1.

```
gSQL> CREATE TABLE t1 ( emp_no INTEGER );
Table created.
```

Dropping Object and Transaction

- Before committing or rolling back a transaction of dropping table

When a table is dropped, and transaction is not committed, other transactions can enquire the dropped t able until the DROP TABLE transaction is committed. DROP TABLE statement, like as DELETE statement, r etrieves the state of before deleting the table when other transaction enquires it until the transaction is c ommitted.

The following example describes the state which the transaction A drops the table t1, then it creates new table t1, and the transaction is not committed.

- Transaction A: There is a row having two columns in the table t1 before dropping the table.

```
gSQL> SELECT * FROM t1;
ID NAME
-- ------
 1 leekmo
1 row selected.
```

- Dropping the existing table t1

```
gSQL> DROP TABLE t1;
Table dropped.
```

    - Creating the new table t1

```
gSQL> CREATE TABLE t1 ( addr VARCHAR(128) );
Table created.
```

    - Creating a new row in the newly created table t1

```
gSQL> INSERT INTO t1 VALUES ( 'Seoul, Korea' );
1 row created.
gSQL> SELECT * FROM t1;
ADDR
------------
Seoul, Korea
1 row selected.
```

If the transaction B enquires when the transaction A is not committed, the information which is before th e transaction A is executed is obtained as follows. DROP TABLE statement, like as DELETE statement, doe s not affect any other transaction until the transaction is committed.

    - Transaction B: The transaction B retrieves the table which is before the execution of the transacti on A.

```
gSQL> SELECT * FROM t1;
ID NAME
-- ------
 1 leekmo
1 row selected.
```

- After committing a transaction of dropping table

If the transaction A is committed, then the transaction B enquires the newly created table t1 as follows.

    - Transaction B: After committing the transaction A, the transaction B enquires the newly created t able t1.

```
gSQL> SELECT * FROM t1;
ADDR
------------
Seoul, Korea
1 row selected.
```

- After rolling back a transaction of dropping table

If the transaction A is rolled back, the transaction B enquires the table t1 which is before the execution of the transaction A as follows. Namely, if the transaction A is rolled back, the rollback transaction does not affect the data of which the transaction B enquires.

- Transaction B: If the transaction A is rolled back, the transaction B enquires the information whic h is before the execution of the transaction A.

```
gSQL> SELECT * FROM t1;
ID NAME
-- ------
 1 leekmo
1 row selected.
```

Altering Object and Transaction

ALTER TABLE statement is used to alter the table structure. ALTER TABLE statement also ensures atomicit y and consistency of the transaction like as CREATING TABLE or DROPPING TABLE statements. Before co mmitting a transaction in which a column is added to a table, other transactions retrieve the information of the existing table as follows. Namely, ALTER TABLE statement, like as UPDATE statement, retrieves inf ormation which is before DDL transaction in other transaction until the transaction is committed.

- Transaction A: It adds a new UPDATE_TIME column.

```
gSQL> ALTER TABLE t1 ADD COLUMN ( update_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP );
Table altered.
```

- The table t1 including the added column is retrieved.

```
gSQL> select * from t1;
ID NAME   UPDATE_TIME
-- ------ --------------------------
 1 leekmo 2014-07-10 12:50:33.540495
1 row selected.
```

If the transaction B is executed before committing the transaction A as follows, then the table t1 which is before adding a column is retrieved.

- Transaction B: The added column UPDATE_TIME is not retrieve, because the transaction A is not com mitted.

```
gSQL> SELECT * FROM t1;
ID NAME
-- ------
```

```
 1 leekmo
1 row selected.
```

12.2 Data Manipulation Language

DML Related Statements

For more information, refer to the followings.

- INSERT related statements
  - **INSERT INTO**
  - **INSERT INTO name RETURNING**
  - **INSERT INTO name RETURNING .. INTO**

- UPDATE related statements
  - **UPDATE**
  - **UPDATE name RETURNING**
  - **UPDATE name RETURNING .. INTO**
  - **UPDATE name WHERE CURRENT OF cursor_name**

- DELETE related statements
  - **DELETE FROM**
  - **DELETE FROM name RETURNING**
  - **DELETE FROM name RETURNING .. INTO**
  - **DELETE FROM name WHERE CURRENT OF cursor_name**

- SELECT related statements: **SELECT .. INTO**

- Dynamic SQL related statements
  - **EXECUTE IMMEDIATE 'sql_string'**
  - **PREPARE statement_name**
  - **EXECUTE statement_name**

Concepts of DML

Data Manipulation Language (DML) is an SQL language which manipulates and enquires data in existing tables such as INSERT, DELETE, UPDATE.

This chapter describes only DML statements which change data. For more information about queries, refe r to **Data Query Language**.

The DDL statements change the SQL object structure, but DML statements manipulate the objects conten ts. For example, ALTER TABLE statement alters the table structure, but INSERT statement adds one or mo re rows in the table.

DML statements such as inserting, deleting, updating data in table are classified as follows.

Table 12-2 Data manipulation statements

Category	Statements	Description
INSERT	INSERT .. VALUES	It adds a single row to the table.
	INSERT .. SELECT	It adds the query results to the table.
	INSERT .. RETURN .. INTO	It sets the value of the added row as a variable.
	INSERT .. RETURN	It retrieves the added row as the query result.
DELETE	DELETE .. WHERE	It deletes the row which satisfies the condition.
	DELETE .. WHERE CURRENT OF	It deletes the row which is at the cursor's position.
	DELETE .. RETURN .. INTO	It sets the value of the removed row as a variable.
	DELETE .. RETURN	It retrieves the removed row as the query result.
UPDATE	UPDATE .. WHERE	It updates the row which satisfies the condition.
	UPDATE .. WHERE CURRENT OF	It updates the row which is at the cursor's position.
	UPDATE .. RETURN .. INTO	It sets the value of the updated row as a variable.
	UPDATE .. RETURN	It retrieves the updated row as the query result.

Inserting Data

It adds data in a row unit when adding data to a table. The INSERT statement cad add one or more rows to a table. Even when the data in some columns are omitted, all rows are added with completed columns to the table.

The following is an example of a table.

```
CREATE TABLE t1
(
    id   NUMBER(10,0),
    name VARCHAR(128),
    addr VARCHAR(1024) DEFAULT 'n/a'
);
```

The most basic way to add a row is as follows.

```
INSERT INTO t1 VALUES ( 1, 'leekmo', 'Seoul, Korea' );
```

The values listed in the VALUES clause are inserted in accordance with the sequence of the listed column when the table is created.
However, the example above can cause an unexpected failure when inserting or deleting columns, so it is recommended to explicitly specify the column name as follows.

```
INSERT INTO t1 (id, name, addr) VALUES ( 1, 'leekmo', 'Seoul, Korea' );
INSERT INTO t1 (name, addr, id) VALUES ( 'leekmo', 'Seoul, Korea', 1 );
```

The two INSERT statements above are listed in different order from the columns, but the added row of th e table t1 has the same data.

If the table does not list all the columns, unspecified column is set to the default value to complete the ro w. The addr column which was not used in the statement stores the default value (n/a) which was specifi ed when creating table as follows.

```
INSERT INTO t1 ( id, name ) VALUES ( 1, 'leekmo' );
INSERT INTO t1 ( id, name ) SELECT id, name FROM emp;
```

Use DEFAULT to explicitly specify the default value for the column as follows.

```
INSERT INTO t1 ( id, name, addr ) VALUES ( 1, 'leekmo', DEFAULT );
```

Use either of the following two statements to set all the columns to the default value.

```
INSERT INTO t1 ( id, name, addr ) VALUES ( DEFAULT, DEFAULT, DEFAULT );
INSERT INTO t1 DEFAULT VALUES;
```

Use a single INSERT statement to add multiple rows. The following is an example of adding three new ro ws using a single INSERT statement.

```
INSERT INTO t1 (id, name, addr) VALUES
  ( 1, 'leekmo', 'Seoul, Korea' ),
  ( 2, 'mkkim', 'Seoul, Korea' ),
  ( 3, 'xcom', 'Inchon, Korea' );
```

Use the SELECT query results to add multiple rows. The following is an example of adding rows to the tab le t1 by retrieving employees who joined the company more than three years ago.

```
INSERT INTO t1 ( id, name, addr )
SELECT id, name, addr
  FROM emp
 WHERE DATEDIFF( YEAR, SYSDATE, join_date ) >= 3;
```

Deleting Data

DELETE statement deletes data from the table in a row unit like when inserting data. Rows can be delete d by using WHERE condition or by using row's ID(ROWID).

The following is an example of deleting rows which satisfy WHERE condition.

```
DELETE FROM t1 WHERE id = 1;
```

The following is an example of deleting the row using ROWID.

```
gSQL> SELECT rowid FROM t1 WHERE id = 1;
                  ROWID
----------------------
AAAAAAAAFNHAACAAAAAiAAA
1 row selected.
gSQL> DELETE FROM t1 WHERE ROWID = 'AAAAAAAAFNHAACAAAAAiAAA';
1 row deleted.
```

DELETE statement without a WHERE clause deletes the entire row in a table. DELETE statement without a WHERE clause is similar to TRUNCATE TABLE statement in terms of deleting entire row, but it is recom mended to use the TRUNCATE TABLE statement.

```
DELETE FROM t1;
TRUNCATE TABLE t1;
```

Updating Data

Update data by using UPDATE statement. One or more rows and columns can be updated. Other column s which is not specified in the UPDATE statement is not affected.

The following is an example of updating a column in rows which satisfy the condition.

```
UPDATE t1 SET page_view = page_view + 1 WHERE id = 1;
```

The followings are examples of updating multiple columns, and the two UPDATE statements mean the sa me.

```
UPDATE t1 SET page_view = page_view + 1, status = 'F' WHERE id = 1;
UPDATE t1 SET (page_view, status) = (page_view + 1, 'F') WHERE id = 1;
```

Use DEFAULT as follows to set the column value to a default value.

```
UPDATE t1 SET addr = DEFAULT WHERE id = 1;
```

Manipulating Data Using Cursor

Cursor is a session object to execute queries and to manipulate the query result. Use cursor to update or delete the query result set.

The following is an example of declaring updatable cursor, and updating or deleting the row at the positi on of the current cursor using the updatable cursor.

```
gSQL> DECLARE cur1 CURSOR FOR SELECT id, data FROM t1 FOR UPDATE;
Cursor declared.
gSQL> OPEN cur1;
Cursor is open.
gSQL> \var v_id INTEGER
gSQL> \var v_data VARCHAR(128)
gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   1 data_1
1 row fetched.
gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   2 data_2
1 row fetched.
gSQL> DELETE FROM t1 WHERE CURRENT OF cur1;
1 row deleted.
gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   3 data_3
1 row fetched.
gSQL> UPDATE t1 SET id = id + :v_id WHERE CURRENT OF cur1;
1 row updated.
gSQL> COMMIT;
Commit complete.
gSQL> SELECT * FROM t1 ORDER BY 1;
```

```
ID DATA
-- ------
 1 data_1
 6 data_3
2 rows selected.
```

The examples above describe the followings.
Use **DECLARE cursor_name** to declare FOR UPDATE cursor, and use **OPEN cursor_name** to open the curs
or.
Use **FETCH cursor_name** to move the cursor to the specified position.
Use **DELETE FROM name WHERE CURRENT OF cursor_name** to delete the row in the specified position.
Use **UPDATE name WHERE CURRENT OF cursor_name** to update the row in the specified position.

FOR UPDATE cursor is closed using **CLOSE cursor_name**, or it is closed when committing transaction.

DML Query

When executing DML statements changing the data, use RETURNING clause to retrieve the changed data.
The RETURNING clause of DML statements, like as SELECT, can retrieve data, so the execution of the DM
L statement and the SELECT statement can be replaced with the DML query.

The following is an example of using **INSERT INTO name RETURNING** syntax to insert data and retrieve th
e result. The join_date value which was input by using SYSDATE function can be retrieved by a single DM
L query.

```
gSQL> INSERT INTO t1 ( id, join_date )  VALUES ( 1, SYSDATE ) RETURNING id, join_date;
ID JOIN_DATE
-- ----------
 1 2014-07-18
1 row created.
```

The following is an example of using **DELETE FROM name RETURNING** syntax to delete data and retrieve
the result. The result data can be manipulated by using operation in RETURNING clause.

```
gSQL> DELETE FROM t1 RETURNING ( id || ': ' || join_date ) AS id_and_join_date;
ID_AND_JOIN_DATE
----------------
1: 2014-07-18
1 row deleted.
```

The following is an example of using **UPDATE name RETURNING** syntax to update rows and retrieve the

updated values. The value which is before updating can be retrieved by using OLD clause.

- It updates the row and retrieves the updated value.

```
gSQL> UPDATE t1 SET page_view = page_view + 1 WHERE id = 1 RETURNING page_view;
PAGE_VIEW
---------
      102
1 row updated.
```

- It updates the row and retrieves the value which is before the updating.

```
gSQL> UPDATE t1 SET page_view = page_view + 1 WHERE id = 1 RETURNING OLD page_view;
PAGE_VIEW
---------
      102
1 row updated.
```

RETURNING clause of DML statements, like as SELECT query, can retrieve multiple query results. However, if the DML is executed only for a single row, then the host variable can be obtained by using RETURNING INTO clause. In this case, the number of changed rows should be one or less, like **SELECT .. INTO** clause .

The following is an example of setting the value to the host variable using RETURNING .. INTO clause of e ach DML statement.

- It declares the host variable.

```
gSQL> \var v_id        INTEGER
gSQL> \var v_page_view BIGINT
gSQL> \var v_date      DATE
```

- After inserting the row, a value is set to the host variable.

```
gSQL> INSERT INTO t1 ( id, join_date ) VALUES ( 1, SYSDATE ) RETURNING join_date INTO :v_date;
V_DATE
--------------------------
2014-07-18 16:57:11.000000
1 row created.
```

- After updating the row, a value is set to the host variable.

```
gSQL> UPDATE t1 SET page_view = page_view + 1 WHERE id = 1 RETURN page_view INTO :v_page_view;
V_PAGE_VIEW
```

```
------------
         101
1 row updated.
```

- After deleting the row, a value is set to the host variable.

```
gSQL> DELETE FROM t1 WHERE id = 1 RETURN id, page_view INTO :v_id, :v_page_view;
V_ID V_PAGE_VIEW
---- ------------
   1          101
1 row deleted.
```

For more information about DML query, refer to the followings.

- **INSERT INTO name RETURNING**
- **INSERT INTO name RETURNING .. INTO**
- **DELETE FROM name RETURNING**
- **DELETE FROM name RETURNING .. INTO**
- **UPDATE name RETURNING**
- **UPDATE name RETURNING .. INTO**

12.3 Data Query Language

Query Related Statements

For more information, refer to the followings.

- SELECT query related statements
  - SELECT
  - SELECT .. FOR UPDATE

- DML query related statements
  - INSERT INTO name RETURNING
  - UPDATE name RETURNING
  - DELETE FROM name RETURNING

- Cursor related statements
  - DECLARE cursor_name
  - OPEN cursor_name
  - FETCH cursor_name
  - CLOSE cursor_name
  - DELETE FROM name WHERE CURRENT OF cursor_name
  - UPDATE name WHERE CURRENT OF cursor_name

Concepts of Query

Query means a series of operations to retrieve data for one or more of the table or view. By using the que ry, a user can get the result data which satisfies the specific condition in the desired form from the stored data.

Query is a SELECT statement which is at the top in the entire SQL statement separated by ';'. Top-level SE LECT statement can include another SELECT statement in it. In this case, the subordinate SELECT stateme nt is called as a subquery.

In GOLDILOCKS, query is divided into SELECT query, DML query, and cursor. SELECT query returns the res ult by using the SELECT statement. DML query returns the result by using the RETURNING phrase in INSE RT, DELETE, UPDATE statements. Cursor temporarily saves the result sets when it is enquired once, and ra ndomly accesses to a row of the saved result set, then brings the result.
A user can get the results at once by using SELECT query and DML query. On the other hand, by using th

e cursor, SELECT statement specified with DECLARE cursor is executed in OPEN cursor, and then it holds t he result set until CLOSE cursor is called. Then it repeatedly brings the result by randomly accessing to a r ow of the result set using FETCH cursor.

This chapter describes SELECT query, DML query and cursor.

Basic Query

The basic form of a query is *SELECT <select list> FROM <table expression>*. *<select list>* which is between SELECT and FROM keywords, specifies one or more columns or expressions to be included in rows which are the result for the table or view described in *<table expression>*.

```
SELECT n_name
     , INITCAP( n_name )
  FROM nation
 WHERE n_regionkey = 1;
N_NAME                    INITCAP( N_NAME )
------------------------- -------------------------
ARGENTINA                 Argentina
BRAZIL                    Brazil
CANADA                    Canada
PERU                      Peru
UNITED STATES             United States
5 rows selected.
```

One or more tables or views can be described in <table expression>, and same column can be included in two tables or views. In this case, the name of table or view should be specified together when describing that column in <select list>. When describing a table or a column, it is recommended to clearly describe t he schema names and table names, etc.

- A wrong example

```
SELECT n_name
  FROM nation   AS n
     , v_nation AS v
 WHERE n.n_nationkey = v.n_nationkey
   AND v.n_regionkey = 1;
ERR-42000(16142): column ambiguously defined :
SELECT n_name
       *
```

```
ERROR at line 1:
```

- A correct example

```
SELECT n.n_name
  FROM nation   AS n
     , v_nation AS v
 WHERE n.n_nationkey = v.n_nationkey
   AND v.n_regionkey = 1;
N_NAME
-------------------------
ARGENTINA
BRAZIL
CANADA
PERU
UNITED STATES
5 rows selected.
```

<select list> supports the alias name. It changes output column names in each columns which are separated by comma (,). Alias name can be used only in the <order by clause>, and it can not to be used in other phrases.

```
SELECT p_type
     , p_retailprice * 0.9 AS discount_price
  FROM part
 ORDER BY discount_price
 FETCH 5;
    2     3     4     5
P_TYPE                 DISCOUNT_PRICE
---------------------- --------------
PROMO BURNISHED COPPER           810.9
ECONOMY BRUSHED NICKEL           810.9
LARGE BRUSHED BRASS              811.8
LARGE BRUSHED NICKEL             811.8
PROMO ANODIZED STEEL             811.8
5 rows selected.
```

In addition to <select list>, <hint clause> and <set quantifier> can also be used between SELECT and FROM keywords. <hint clause> allows a user to directly adjust the query execution plan. For more information, refer to **SQL Hint**. <set quantifier> removes the duplicate data of the row which is returned as the result. For more information, refer to **query specification**.

- An example of using hint

```
SELECT
       /*+ INDEX( part ) */
       p_type
     , p_retailprice
  FROM part
 WHERE p_partkey = 100;
P_TYPE               P_RETAILPRICE
-------------------- -------------
ECONOMY ANODIZED TIN        1000.1
1 row selected.
```

- An example of using <set quantifier>

```
SELECT DISTINCT
       o_orderpriority
  FROM orders;
O_ORDERPRIORITY
---------------
5-LOW
2-HIGH
3-MEDIUM
1-URGENT
4-NOT SPECIFIED
5 rows selected.
```

SET Operator

SET operators combine the result set of two or more queries into a single result set. SET operators are UN ION, EXCEPT, INTERSECT, and MINUS. MINUS operates as same as EXCEPT. Each SET operator has additi onal options such as ALL and DISTINCT. If the option is omitted, DISTINCT is used by default.

When using the SET operators to describe two or more queries, generally the queries are processed sequ entially from the left. When parentheses are used to explicitly specify the processing order of the queries, then those queries are processed first.

```
SELECT n_name
  FROM nation
 WHERE n_nationkey < 10
```

```
INTERSECT
( SELECT n_name
    FROM nation
   WHERE n_regionkey = 1
  UNION ALL
  SELECT n_name
    FROM nation
   WHERE n_regionkey = 2 );
N_NAME
-------------------------
BRAZIL
ARGENTINA
INDONESIA
INDIA
CANADA
5 rows selected.
```

Each query of the SET operators should have the same number of the target, and the target at the same positions of each query should have a data type which belongs to the same group.

<order by clause> of SET operator sorts the final result set. Each query of SET operators can have its own <order by clause> to sort themselves.

For more information about SET operators, refer to **set operator**.

Common Table Expression (CTE)

The temporary result set configured through <with clause> is called as Common Table Expression (CTE). CTE defined in the syntax can be referenced within the execution range. One or more CTE are configured in <with clause>, and each CTE can be configured by referencing CTE including itself. Repeatedly executin g the query to configure the temporary result set is called as recursive subquery factoring.

CTE is classified into recursive CTE and non-recursive CTE.

Referring to CTE which is currently defined within CTE (self-reference CTE) is called as recursive CTE. CTE other than recursive CTE is called as non-recursive CTE.

```
* recursive CTE
WITH RECURSIVE_CTE ( c1 ) AS
     (
          SELECT 1
            FROM dual
```

```
          UNION ALL
          SELECT c1 + 1
            FROM RECURSIVE_CTE      ❶ Self reference
           WHERE c1 < 10
     )
SELECT c1 FROM RECURSIVE_CTE;
```

```
* non-recursive CTE
WITH NON_RECURSIVE_CTE ( c1 ) AS
     (
          SELECT i1
            FROM t1
          UNION ALL
          SELECT i1
            FROM t2
     )
SELECT c1 FROM NON_RECURSIVE_CTE;
```

Recursive CTE

Recursive CTE is always consists of two query block sets using UNION ALL. The query block including Self-reference CTE is called as a recursive member query, and other query blocks are called as anchor member queries.

```
WITH CTE_RECURSIVE( c1, c2 ) AS
    (
         SELECT i1, i2                      ❶ Anchor member query
           FROM t1
          WHERE i2 IS NULL
         UNION ALL
         SELECT i1, i2                      ❷ Recursive member query
           FROM CTE_RECURSIVE, t1
          WHERE CTE_RECURSIVE.c1 = t1.i2
    )
SELECT c1, c2 FROM CTE_RECURSIVE;
```

Recursive CTE configures records acquired from an anchor member query into the temporary result set, a nd acquires those records through CTE reference in the recursive member query. Recursive CTE also confi gures results acquired from the recursive member query into the temporary result set, and tries to execut e the recursive member query again. This process is repeated until it is unable to configure the temporary result set.

〈search clause〉 is used to specify the sequence of the record configuration configured in the current step. 〈search clause〉 supports DEPTH FIRST BY and BREADTH FIRST BY. 〈search clause〉 can be described only within recursive CTE.

For more information, refer to **〈search clause〉**.

```
gSQL> SELECT * FROM t1;
I1  I2
--- ---
A   ---
AA    A
AB    A
AC    A
AAX  AA
ABX  AB
ACX  AC
7 rows selected.
* SEARCH BREADTH FIRST BY
gSQL> WITH w1( w_i1, w_i2 ) AS
    (
         SELECT i1, i2
           FROM t1
          WHERE i1 = 'A'
         UNION ALL
         SELECT i1, i2
           FROM w1, t1
          WHERE w_i1 = i2
    ) SEARCH BREADTH FIRST BY w_i1, w_i2 SET w_seq
SELECT w_i1, w_i2, w_seq
 FROM w1;
W_I1 W_I2 W_SEQ
---- ---- -----
A    ---      1
AA   A        2
AB   A        3
AC   A        4
AAX  AA       5
ABX  AB       6
ACX  AC       7
7 rows selected.
* SEARCH DEPTH FIRST BY
gSQL> WITH w1( w_i1, w_i2 ) AS
```

```
    (
         SELECT i1, i2
           FROM t1
          WHERE i1 = 'A'
         UNION ALL
         SELECT i1, i2
           FROM w1, t1
          WHERE w_i1 = i2
    ) SEARCH DEPTH FIRST BY w_i1, w_i2 SET w_seq
SELECT w_i1, w_i2, w_seq
 FROM w1;
W_I1 W_I2 W_SEQ
---- ---- -----
A    ---      1
AA   A        2
AAX  AA       3
AB   A        4
ABX  AB       5
AC   A        6
ACX  AC       7
7 rows selected.
```

When configuring the results configured in the previous step into the current temporary result set, then r ecursive CTE is infinitely executed. In this case, the system determines that the cycle occurs, so it causes cy cle detected error.

The comparing target is selected through <cycle clause> to determine whether cycle occurs, and whether cycle occurred is also ca be seen. If using <cycle clause>, it does not cause an error for cycle detected whe n cycle occurred.

When <cycle clause> is not specified, then all columns used when defining CTE are selected as targets of determining whether cycle occurred.

```
gSQL> SELECT * FROM t1;
I1  I2
--- ---
A   ---
AA  A
AB  A
AC  A
AA  AA
AAX AA
```

```
ABX AB
ACX AC
8 rows selected.
```

- When cycle occurred but cycle clause is not described

```
gSQL> WITH w1( w_i1, w_i2 ) AS
     (           SELECT i1, i2
            FROM t1
           WHERE i1 = 'A'
          UNION ALL
          SELECT i1, i2
            FROM w1, t1
           WHERE w_i1 = i2
     )
SELECT w_i1, w_i2
  FROM w1;
ERR-42000(16511): cycle detected while executing recursive WITH query
```

- When cycle occurred and cycle clause is described

```
gSQL> WITH w1( w_i1, w_i2 ) AS
     (
          SELECT i1, i2
            FROM t1
           WHERE i1 = 'A'
          UNION ALL
          SELECT i1, i2
            FROM w1, t1
           WHERE w_i1 = i2
     ) CYCLE w_i1, w_i2 SET c_cycle TO 'T' DEFAULT 'F'
SELECT w_i1, w_i2, c_cycle
  FROM w1;
W_I1 W_I2 C_CYCLE
---- ---- -------
A    ---  F
AC   A    F
AB   A    F
AA   A    F
ACX  AC   F
ABX  AB   F
AAX  AA   F
```

```
AA   AA   F
AAX  AA   F
AA   AA   T
10 rows selected.
```

Join

Join is a query that combines rows from more than one table or view in <from clause>. If there is not a join condition, the result is obtained by combining each result row of the left table or view with each result row of the right table or view.

If tables or views have a column name in common when joining two or more tables or views in <from clause>, a user should distinguish these columns by using the table or view name in <select list>, <where clause>. Otherwise, a validation error occurs.

Join queries either contain the join condition or do not contain the join condition. Join condition is for comparing columns from two different tables or views. If the join condition is not specified, each row of a table or view is combined with each row of another table or view, and the combined row is returned. If the join condition is specified, rows from each table or view which satisfies the join condition, are returned in the combined form.

Equi-join is a join whose join condition contains an equality operator(=). Equi-join condition is an important factor in optimizing the join operation by the optimizer.

Self-join is a join operation which has only the same tables in <from clause>. To describe column in <select list>, the alias name in each table is described and table alias in column is used.

CROSS JOIN

CROSS JOIN is a join operation whose join condition does not exist, and it is also called as Cartesian Product. CROSS JOIN combines each row of one table or view with each row of the other, and the combined row is returned as a result.

```
SELECT a.r_name
     , b.r_name
  FROM region AS a
     , region AS b
 FETCH 5;
R_NAME                    R_NAME
------------------------- -------------------------
AFRICA                    AFRICA
AFRICA                    AMERICA
```

```
AFRICA                    ASIA
AFRICA                    EUROPE
AFRICA                    MIDDLE EAST
5 rows selected.
```

INNER JOIN

INNER JOIN returns the rows which satisfy the join condition for two or more tables or views. INNER JOIN is when *inner join* is explicitly specified in <from clause>. Or, when tables or views are listed with comma (,) in <from clause> and its join condition is specified in <where clause>.
If there are explicit *inner join* in <from clause>, and the join condition in <where clause>, then they are pr ocessed as one inner join condition.

- An example of using an INNER JOIN statement

```
SELECT n_name
  FROM region INNER JOIN nation ON r_regionkey = n_regionkey
 WHERE r_name = 'AFRICA';
N_NAME
-------------------------
ALGERIA
ETHIOPIA
KENYA
MOROCCO
MOZAMBIQUE
5 rows selected.
```

- An example of a list using a comma (,)

```
SELECT n_name
  FROM region
     , nation
 WHERE r_regionkey = n_regionkey
   AND r_name = 'AFRICA';
N_NAME
-------------------------
ALGERIA
ETHIOPIA
KENYA
MOROCCO
MOZAMBIQUE
```

```
5 rows selected.
```

OUTER JOIN

OUTER JOIN returns all rows which satisfy the join condition for two or more tables or views. It also retur ns rows which do not satisfy the join condition for one or both side of table or view depending on the dir ection of OUTER JOIN.

OUTER JOIN can be classified as LEFT OUTER JOIN , RIGHT OUTER JOIN, FULL OUTER JOIN. All three OUT ER JOIN returns rows which satisfy the join condition, but they are distinguished by the additional results. LEFT OUTER JOIN returns all rows of the left table which do not satisfy the join condition by filling NULL f or all of its right rows. RIGHT OUTER JOIN returns all rows of the right table which do not satisfy the join condition by filling NULL for all of its left rows. FULL OUTER JOIN returns all additional return results of LE FT OUTER JOIN and RIGHT OUTER JOIN.

```
SELECT r_name
     , n_name
  FROM region LEFT OUTER JOIN nation
       ON r_regionkey = n_regionkey AND r_name = 'AFRICA';
R_NAME                    N_NAME
------------------------- -------------------------
AFRICA                    ALGERIA
AFRICA                    ETHIOPIA
AFRICA                    KENYA
AFRICA                    MOROCCO
AFRICA                    MOZAMBIQUE
AMERICA                   null
ASIA                      null
EUROPE                    null
MIDDLE EAST               null
9 rows selected.
```

GOLDILOCKS supports the outer join operator(+) for compatibility with Oracle, and which is not supporte d in the SQL standard. The outer join operator(+) lists tables by using a comma(,) in <from clause> and it adds a (+) to a node column which is operated as the outer node in the join condition of <where clause>.

When using the outer join operator (+), the sign(+) should be specified on the right side of the column as follows.

```
select * from t1, t2 where t1.i1 = t2.i1(+);
```

The syntax rules of the outer join operator (+) are as follows.

- <Join outer operator> can be used only for <where clause>.
- <Join outer operator> can be used only for <column reference> of <table reference> which is a target of <joined table>.
- <value expression> including <join outer operator>, can not be combined with other conditions whic h use <OR logical operator>.
- <column reference> including <join outer operator> can not be used as the argument of IN function.
- A <table reference> can not be used as null generated table of many outer joins (constraints on the o uter join)
- When executing the outer join of two or more tables, they are listed according to the left outer join s equence, and executes the outer join from the very left of them.
- When executing Out Join of two or more tables and multiple tables are combined into a table by the outer join, then the outer join is executed according to the sequence of calculation by an optimizer.

Even when the outer join operator (+) is specified, it is ignored in the following cases.

- <join outer operator> can be used for <value expression> which can be used as the join condition bet ween the two tables, if it can not be used as the join condition, then it is ignored and an error or war ning does not occur.
- <join outer operator> which is specified in <column reference> of the outer query is ignored, and an error or warning does not occur.
- If two <table refernce> are outer joined using <join outer operator>, then <join outer operator> shoul d be specified in all <column reference> which belong to null generated tables. Otherwise, join of tw o <table reference> is processed as an inner join, an error or warning does not occur.

Differences between Oracle and GOLDILOCKS for the outer join operator (+) are as follows.

- Quantified comparison (in, = any, = all = row, etc.)
  - GOLDILOCKS: It processes the operation as a validation error.
  - Oracle: The and/or operation is applied, and performs the validation check.
- *Or* sub-clause in an *and* clause
  - GOLDILOCKS: Validation is applied on the columns of an *or* sub-clause.
  - Oracle: Validation is ignored on the columns of an *or* sub-clause.
- When the condition clause includes a subquery
  - GOLDILOCKS: The outer join is applied and the condition is processed as a join condition.
  - Oracle: The outer join is applied but the condition is processed as a where filter.

GOLDILOCKS supports the outer join operator (+) for compatibility with Oracle. It is recommend ed to specify OUTER JOIN in <from clause>. (Oracle also recommends to specify OUTER JOIN in <from clause>.)

NATURAL JOIN

NATURAL JOIN uses the join condition of an equality operator(=) for columns of same name when joinin g two or more tables or views. It is as same as INNER JOIN except that NATURLAL JOIN has the implicit joi n condition for columns of the same name.

```
SELECT r_name
  FROM region a NATURAL JOIN region b;
R_NAME
-------------------------
AFRICA
AMERICA
ASIA
EUROPE
MIDDLE EAST
5 rows selected.
```

SEMI JOIN

SEMI JOIN returns the corresponding left rows if there exist right rows which satisfy the join condition. U nlike other join operations to return the rows which combine the left and right rows, SEMI JOIN returns o nly the left rows as the result.

- An example of using the semi join

```
SELECT r_name
  FROM region
 WHERE r_regionkey IN ( SELECT n_regionkey
                          FROM nation
                         WHERE n_nationkey < 5 );
R_NAME
-------------------------
AFRICA
AMERICA
MIDDLE EAST
3 rows selected.
```

ANTI-SEMI JOIN

ANTI-SEMI JOIN returns the corresponding left rows if there does not exist any right rows which satisfy th e join condition. It returns only the left rows as the result like as SEMI JOIN.

- An example of using the anti-semi join

```
SELECT r_name
  FROM region
 WHERE r_regionkey NOT IN ( SELECT n_regionkey
                              FROM nation
                             WHERE n_nationkey < 5 );
R_NAME
-------------------------
ASIA
EUROPE
2 rows selected.
```

For more information about the join operation, refer to **joined table**.

Hierarchical Query

A hierarchical query can process the hierarchical model data. The hierarchical model data consists of the hierarchical relationship with the connecting condition.

Recursive CTE is used to express the hierarchical query. For more information, refer to **Recursive CTE**.

The other way to configure a hierarchical query is using <hierarchical query clause>.

<hierarchical query clause> creates a hierarchy by using the given starting condition (<start with clause>) and child connecting condition (<connect by clause>), and configures the result record with depth-first m ethod. <hierarchical query clause> should include <connect by clause>.

```
SELECT *
  FROM r_region
 WHERE r_population > 10000000
 START WITH r_name = 'EARTH'              ❶ Starting condition
CONNECT BY r_domain = PRIOR r_name        ❷ Connecting condition
```

All expressions used as an argument of PRIOR operator among expressions described in <connect by> are configured into the result for the current hierarchy. Only the expressions configured into the result are us ed as targets to check whether cycle occurs. The results configured in this way can be referenced through **<hierarchy expression>**.

Whether cycle occurs in a hierarchical query is checked by repeatedly searching along the parent hierarch y based on currently configured result and checking whether the same result exists. When it is determine d that cycle occurred based on the current result, then the information is set that the parent record of the

current result record includes the record where cycle occurred.

If searching for the record including cycle occurrence information, then the system determines that the cy cle occurs, so it causes cycle detected error. If describing NOCYCLE in <connect by> statement, then it do es not cause cycle detected error, and whether cycle occurs is known through CONNECT_BY_ISCYCLE wh ich is one of <hierarchical expression>.

```
gSQL>
SELECT * FROM t1;
I1 I2
-- ----
A  null
AA A
AB A
AC A
AA AA
AB AA
6 rows selected.
gSQL>
SELECT i1, i2, CONNECT_BY_ISCYCLE
  FROM t1
START WITH i1 = 'A'
CONNECT BY NOCYCLE i2 = prior i1;
I1 I2   CONNECT_BY_ISCYCLE
-- ---- ------------------
A  null                  0
AA A                     1
AB AA                    0
AB A                     0
AC A                     0
5 rows selected.
```

Sibling records of the same parent records are sorted through <order sibling by clause>. <order sibling by clause> is applied when configuring the result record per each hierarchy.
However, <order by clause> which has a similar form sorts all records acquired from the query block. Ther efore, <order sibling by clause> and <order by clause> does not affect each other, and constraints do not exist between them.

```
gSQL>
SELECT * FROM t1;
I1  I2
--- ----
```

```
A   null
AA  A
AB  A
fAA AA
eAA AA
bAA AA
dAB AB
cAB AB
aAB AB
9 rows selected.
```

- The following is an example of defining the order of fetching sibling records of the same parent recor ds.

```
gSQL>
SELECT LEVEL, i1, i2
  FROM t1
START WITH i1 = 'A'
CONNECT BY i2 = PRIOR i1
ORDER SIBLINGS BY i1;
LEVEL I1  I2
----- --- ----
    1 A   null
    2 AA  A
    3 bAA AA
    3 eAA AA
    3 fAA AA
    2 AB  A
    3 aAB AB
    3 cAB AB
    3 dAB AB
9 rows selected.
```

- The following is an example of sorting all results retrieved in a hierarchy by using ORDER BY clause in an order of LEVEL.

```
gSQL>
SELECT LEVEL, i1, i2
  FROM t1
START WITH i1 = 'A'
CONNECT BY i2 = PRIOR i1
ORDER SIBLINGS BY i1
```

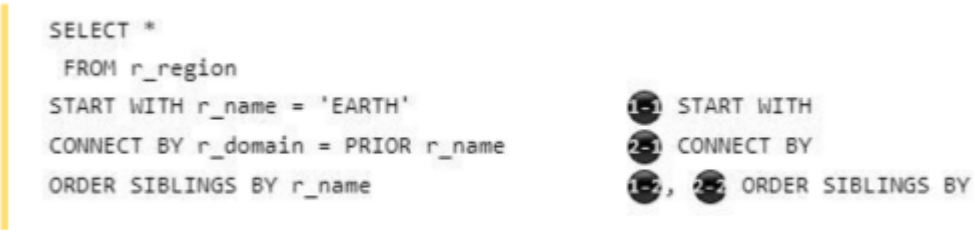
```
ORDER BY LEVEL;
LEVEL I1  I2
----- --- ----
    1 A   null
    2 AA  A
    2 AB  A
    3 bAA AA
    3 eAA AA
    3 fAA AA
    3 aAB AB
    3 cAB AB
    3 dAB AB
9 rows selected.
```

Order of evaluating hierarchical query

<hierarchical query clause> consists of <start with connect by clause> and <order siblings by clause>.

Statements in <hierarchical query clause> are executed in the following order.

Results acquired through <start with connect by clause> from each hierarchy are sorted through <order siblings by clause>. <start with clause> is evaluated for the root hierarchy and <order siblings by clause> is applied. Then, the result is configured by using <connect by clause> and <order siblings by clause> for child hierarchy which is configured later.



<hierarchical query clause> in the query block is executed in the following order.

<hierarchical query clause> is described between <where clause> and <group by clause> after <from clause>. Unlike the describing order for <hierarchical query clause>, <hierarchical query clause> is evaluated after <from clause> and before <where clause>.

Conditions described in <where clause> does not affect the evaluation of <hierarchical query clause>.

The followings are the evaluating order of <hierarchical query clause> according to the configuration of

〈from clause〉 and 〈where clause〉.

- When only a single table exists in *from* clause

```
SELECT *
 FROM r_region
WHERE r_population > 10000000          ❸ Condition in WHERE clause
START WITH r_name = 'EARTH'            ❶ START WITH
CONNECT BY r_domain = PRIOR r_name     ❷ CONNECT BY
```

- When join condition configured with multiple tables exists in *from* clause
  - When join condition is described in ON clause of FROM clause

```
SELECT *
  FROM r_region INNER JOIN s_region
       ON r_id = s_id                  ❶ Join condition in ON clause
 WHERE r_population > 10000000         ❹ Condition in WHERE clause
START WITH r_name = 'EARTH'            ❷ START WITH
CONNECT BY r_domain = PRIOR r_name     ❸ CONNECT BY
```

- When join condition is described in WHERE clause

```
SELECT *
  FROM r_region, s_region
 WHERE r_population > 10000000         ❸ Condition in WHERE clause
   AND r_id = s_id                     ❸ Join condition in WHERE clause
START WITH r_name = 'EARTH'            ❶ START WITH
CONNECT BY r_domain = PRIOR r_name     ❷ CONNECT BY
```

  - When join condition is described in both ON clause of FROM and in WHERE clause

```
SELECT *
  FROM r_region INNER JOIN s_region
       ON r_name = s_name              ❶ Join condition in ON clause
 WHERE r_population > 10000000         ❹ Condition in WHERE clause
   AND r_id = s_id                     ❹ Join condition in WHERE clause
START WITH r_name = 'EARTH'            ❷ START WITH
CONNECT BY r_domain = PRIOR r_name     ❸ CONNECT BY
```

If 〈group by clause〉 is used in a query block including 〈hierarchical query clause〉, then it is able to config ure the result set group by using 〈hierarchical expression〉.

```
SELECT COUNT(*)
  FROM r_region
 START WITH r_name = 'EARTH'
CONNECT BY r_domain = PRIOR r_name
 GROUP BY PROIR r_domain
```

Grouping Result Set (group by)

<group by clause> is used in order to group the rows which have the same column into one. <group by clause> lists and delimiters columns by using the comma (,), and GOLDILOCKS supports the group operation based on it.

```
SELECT
       o_orderpriority
     , MIN( o_totalprice ) AS min_price
     , MAX( o_totalprice ) AS max_price
  FROM orders
 GROUP BY
       o_orderpriority;
O_ORDERPRIORITY MIN_PRICE MAX_PRICE
--------------- --------- ---------
5-LOW              857.71 530604.44
2-HIGH              896.8 522720.61
3-MEDIUM           875.52 508668.52
1-URGENT            866.9 544089.09
4-NOT SPECIFIED    884.82 555285.16
5 rows selected.
```

If <group by clause> is specified, only the columns and the aggregate functions described in <group by clause> can be included in <select list>.

If only constants or parentheses instead of columns are specified in <group by clause>, it is assumed to be empty grouping set which includes the imaginary columns of the same value in each row. Then, the grouping in column base is performed. It is same when specifying <having clause> without <group by clause>.

<having clause> can be used to get the specified rows from grouped results by <group by clause>. <having clause> describes the conditions for each group, and it can describes the condition using the aggregate operation.

```
SELECT
       o_orderpriority
     , MIN( o_totalprice ) AS min_price
     , MAX( o_totalprice ) AS max_price
  FROM orders
 GROUP BY
       o_orderpriority
 HAVING
       MIN( o_totalprice ) < 870;
O_ORDERPRIORITY MIN_PRICE MAX_PRICE
--------------- --------- ---------
5-LOW              857.71 530604.44
1-URGENT            866.9 544089.09
2 rows selected.
```

For more information about the grouping, refer to **group by clause**.

Window Query

The window function returns the execution result of the function for the defined record range. The defined record range is called as a window and the record range is defined in OVER <window name or specification>. Unlike a general function or an aggregate function, the window function has OVER clause.

```
SUM( sales ) OVER ()
SUM( sales ) OVER window_name
SUM( sales ) OVER ( PARTITION BY item_no
                    ORDER BY sales
                    ROWS BETWEEN UNBOUNDED PRECEDING
                             AND CURRENT ROW )
```

The window function, like as an aggregate function, returns the execution result of function for multiple records. However, the aggregate function returns a single record per each group, but the window function returns multiple records per each group.

Each record in the group of the window function has the execution result of the function for the window (the defined record range). Therefore, unlike an aggregate function, the window function returns all records of each group.

The following is an example of executing the aggregate function and the window function.

- The following is a sample table.

```
gSQL>
SELECT * FROM store;
ITEM_NO SALES_DATE SALES
------- ---------- -----
    100 2001-01-01   150
    100 2001-01-02   100
    100 2001-01-03   170
    100 2001-01-04    90
    100 2001-01-05   200
    235 2001-01-01    70
    235 2001-01-02   130
    235 2001-01-03   190
    235 2001-01-04   150
    235 2001-01-05    50
10 rows selected.
```

- Executing result of aggregate function

```
gSQL>
SELECT SUM( sales ) AS aggrfunc_sum
  FROM store;
AGGRFUNC_SUM
------------
        1300
1 row selected.
```

- Executing result of window function

```
gSQL>
SELECT item_no,
       sales,
       SUM( sales ) OVER () as windowfunc_sum
  FROM store;
ITEM_NO SALES WINDOWFUNC_SUM
------- ----- --------------
    100   150           1300
    100   100           1300
    100   170           1300
    100    90           1300
    100   200           1300
    235    70           1300
```

```
    235   130           1300
    235   190           1300
    235   150           1300
    235    50           1300
10 rows selected.
```

The window function defines the window (the execution range of the function) in OVER <window name or specification> clause.

It is divided into groups as described in <window partition clause> PARTITION BY.
Then, the records in the group are sorted as described in <window order clause> ORDER BY.
Then, the record range, the window function target, is defined among records sorted within the group as described in <window frame clause>.

For more information about the definition for the window, refer to **window clause**.

The window function is executed per partition defined in <window partition clause> PARTITION BY, and i f PARTITION BY is omitted, then the entire result record become a single partition.

If <window order clause> ORDER BY is described, then the window frame (the execution range of the wi ndow function) is applied to each record (current row) within the partition.

The window frame is defined by describing <window frame clause>, and it defines the application unit (R OWS/ RANGE/ GROUPS), starting point, ending point and the excluded records.

If <window frame clause> is omitted, then RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT R OW is applied by default. In this case, the execution range is from the record at the beginning of the parti tion to all peer records of the current record.

For more information about <window frame clause>, refer to **<window frame clause>**.

The following examples describe the difference in results between the case when <window order clause> ORDER BY is omitted and the case when the window frame is applied by describing <window order claus e> ORDER BY.

- When <window order clause> ORDER BY is omitted
    - If PARTITION BY is described, then it calculates the sum(sales) per the specified partition.

```
gSQL>
SELECT item_no,
       sales,
       SUM( sales ) OVER ( PARTITION BY item_no ) as windowfunc_sum
  FROM store;
ITEM_NO SALES WINDOWFUNC_SUM
```

```
------- ----- --------------
    100   150            710
    100   100            710
    100   170            710
    100    90            710
    100   200            710
    235    70            590
    235   130            590
    235   190            590
    235   150            590
    235    50            590
10 rows selected.
```

    - If PARTITION BY is omitted, then the entire record is considered as a single partition, so it calculates the sum(sales) of all records.

```
gSQL>
SELECT item_no,
       sales,
       SUM( sales ) OVER ( ) as windowfunc_sum
  FROM store;
ITEM_NO SALES WINDOWFUNC_SUM
------- ----- --------------
    100   150           1300
    100   100           1300
    100   170           1300
    100    90           1300
    100   200           1300
    235    70           1300
    235   130           1300
    235   190           1300
    235   150           1300
    235    50           1300
10 rows selected.
```

- When <window order clause> ORDER BY is described

    - If PARTITION BY is described: <window frame clause> is omitted, so RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW is applied by default, and it calculates the sum(sales) from the record at the beginning of the partition to all peer records of the current record.

```
gSQL>
SELECT item_no,
       sales,
       SUM( sales ) OVER ( PARTITION BY item_no
                           ORDER BY sales ) as windowfunc_sum
  FROM store;
ITEM_NO SALES WINDOWFUNC_SUM
------- ----- --------------
    100    90             90
    100   100            190
    100   150            340
    100   170            510
    100   200            710
    235    50             50
    235    70            120
    235   130            250
    235   150            400
    235   190            590
10 rows selected.
```

- If PARTITION BY is omitted: The entire record is considered as a single partition. In this case, <window frame clause> is omitted, and RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW is applied by default, so it calculates the sum(sales) from the record at the beginning of the partition to all peer records of the current record.

```
gSQL>
SELECT item_no,
       sales,
       SUM( sales ) OVER ( ORDER BY sales ) as windowfunc_sum
  FROM store;
ITEM_NO SALES WINDOWFUNC_SUM
------- ----- --------------
    235    50             50
    235    70            120
    100    90            210
    100   100            310
    235   130            440
    100   150            740
    235   150            740
    100   170            910
    235   190           1100
    100   200           1300
```

```
10 rows selected.
```

When executing multiple window functions for the same window (the defined record range), then defin e the window name in WINDOW clause and refer to it.

```
gSQL>
SELECT item_no,
       sales_date,
       sales,
       SUM( sales ) OVER W1 cumulative_sales,
       AVG( sales ) OVER w1 avg_sales
  FROM store
WINDOW w1 AS ( PARTITION BY item_no
               ORDER BY sales_date
               ROWS BETWEEN UNBOUNDED PRECEDING
                        AND CURRENT ROW );
ITEM_NO SALES_DATE SALES CUMULATIVE_SALES AVG_SALES
------- ---------- ----- ---------------- ---------
    100 2001-01-01   150              150       150
    100 2001-01-02   100              250       125
    100 2001-01-03   170              420       140
    100 2001-01-04    90              510     127.5
    100 2001-01-05   200              710       142
    235 2001-01-01    70               70        70
    235 2001-01-02   130              200       100
    235 2001-01-03   190              390       130
    235 2001-01-04   150              540       135
    235 2001-01-05    50              590       118
10 rows selected.
```

The window function can be described in *select list* and *order by* clause.

The window function is executed for the result set after FROM, WHERE, GROUP BY and HAVING clause i s executed. If aggregate, GROUP BY and HAVING clause is used in a query, then describe the group colu mn instead of the original table's column in the window function.

For more information, refer to **Window Function** and **window clause**.

Sorting Result Set (order by)

<Order by clause> sorts the result set based on the specific columns. <order by clause> can sort based on all types of column except the LONG type.

When a positive integer value is in <order by clause>, it means the column which is located in the corresponding to an integer value for the targets in the <select list>. The range of positive integer value which can be described in <order by clause> is from 1 and to the number of the targets in <select list>.

```
SELECT
       o_orderpriority
     , MIN( o_totalprice ) AS min_price
     , MAX( o_totalprice ) AS max_price
  FROM orders
 GROUP BY
       o_orderpriority
 ORDER BY 1;
O_ORDERPRIORITY MIN_PRICE MAX_PRICE
--------------- --------- ---------
1-URGENT            866.9 544089.09
2-HIGH              896.8 522720.61
3-MEDIUM           875.52 508668.52
4-NOT SPECIFIED    884.82 555285.16
5-LOW              857.71 530604.44
5 rows selected.
```

If the column data type in <order by clause> is numeric, it is sorted by numeric comparison. If the column data type is a character type, it is sorted by character comparison.

Each column in <order by clause> can be described by using the sorting direction option such as ASC, DESC. If it is omitted, it is regarded as ASC.

For more information about sorting, refer to **order by clause**.

Subquery

Subquery supports a multi-level search request. The multi-level search request means that the result of the current query depends on the result of the subquery.
For example, the query to get people who is older than the average age of the people in a specific group is written in multi-step. The first query is to obtain the average age of people in a specific group, then the

second query is to retrieve the number of people who is older than the average using the first query.

```
SELECT e_name
  FROM emp
 WHERE e_age > ( SELECT AVG(e_age)
                   FROM emp
                  WHERE e_dept = 'RND' );
```

A subquery can be used in <from clause>, <where clause>. The subquery in <from clause> is "inline view" and the subquery in <where clause>is "nested subquery".

The column name of a table or view in nested subquery can be same as the column name of a table or view in the query including the nested subquery when using the nested subquery. If only the column name exists in <select list> of the nested subquery, then it refer to the column of the table or view in the nested subquery. If the column name in <select list> of the nested subquery does not exist in the nested subquery's table or view, then it refers to the column name of the table or view in the query which includes the nested subquery, if exists.

```
SELECT r_name
  FROM region
 WHERE EXISTS ( SELECT *
                  FROM nation
                 WHERE n_nationkey < 5             /* nation.n_nationkey */
                   AND n_regionkey = r_regionkey ) /* nation.n_regionkey = region.r_regionkey
*/
;
```

The optimizer unnests the nested subquery of <where clause> in the query which includes the nested subquery. Then it processes it as SEMI JOIN or ANTI-SEMI JOIN operation.
This is an optimization of the nested subquery, and the optimizer determines it by calculating the cost of unnesting if the subquery exists in IN, NOT IN, EXISTS, NOT EXISTS, quantify operator.
If a user wants to forcibly unnest the nested subquery, the user can use <hint clause> of nested subquery.

For more information about unnesting the nested subquery, refer to **SQL Hint**.

```
SELECT r_name
  FROM region
 WHERE r_regionkey IN ( SELECT /*+ UNNEST */
                               n_regionkey
                          FROM nation
                         WHERE n_nationkey < 5 );
```

For more information about subquery, refer to **subquery**.

12.4 Control Language

Control Language Related Statements

For more information, refer to the followings.

- Transaction control related statements
  - **COMMIT**
  - **ROLLBACK**
  - **LOCK TABLE**
  - **SAVEPOINT savepoint_specifier**
  - **RELEASE SAVEPOINT savepoint_specifier**

- Session control related statements: Refer to the followings.
  - **ALTER SESSION SET property_name**
  - **SET SESSION AUTHORIZATION user_identifier**
  - **SET SESSION CHARACTERISTICS AS transaction_mode**
  - **SET TIME ZONE**
  - **SET TRANSACTION transaction_mode**

- System control related statements: Refer to the followings.
  - **ALTER SYSTEM CHECKPOINT**
  - **ALTER SYSTEM {MOUNT | OPEN} DATABASE**
  - **ALTER SYSTEM [KILL | DISCONNECT] SESSION**
  - **ALTER SYSTEM SET property_name**
  - **ALTER SYSTEM RESET property_name**
  - **ALTER SYSTEM SWITCH LOGFILE**

Transaction Control

Transaction control statements are used to manage the changes caused by executing DML or DDL statem ents in the transactions. Transaction control statements can be committed to keep the changes permane ntly, or rolled back to undo the changes.

Transaction control statements are classified as follows.

Table 12-3 Transaction control statements

Statement	Description	Refer to
COMMIT	It terminates the transaction normally.	**COMMIT**
ROLLBACK	It undoes the transaction.	**ROLLBACK**
SAVEPOINT	It creates the savepoint.	**SAVEPOINT savepoint_specifier**
RELEASE SAVEPOINT	It removes the savepoint.	**RELEASE SAVEPOINT savepoint_specifier**
LOCK TABLE	It sets the table-level lock.	**LOCK TABLE**
SET TRANSACTION	It controls the transaction properties. (Read/write, isolation level)	**20.25 SET TRANSACTION transaction_mode**
SET CONSTRAINTS	It controls the checkpoint of the deferrable constraints.	**SET CONSTRAINTS**

A transaction is automatically created when the DML statements or DDL statements are executed for the first time. DML statements change the data and DDL statements change the SQL objects. However, the SELECT statements or the control statement do not generate a transaction.

The transaction rollbacks are classified as total rollback and partial rollback. The partial rollback is executed either in an explicit method or an implicit method. The total rollback is executed by ROLLBACK statement, and it undoes any changes performed by DML, DDL within the transaction.

The explicit partial rollback is a method which a user executes ROLLBACK statement using the savepoint as follows.
The following example describes that savepoints sp1, sp2 are declared, and then the partial rollback is explicitly executed.

```
gSQL> CREATE TABLE t1 ( id INTEGER, name VARCHAR(128) );
Table created.
gSQL> COMMIT;
Commit complete.
gSQL> INSERT INTO t1 VALUES ( 1, 'leekmo' );
1 row created.
gSQL> SAVEPOINT sp1;
Savepoint created.
gSQL> INSERT INTO t1 VALUES ( 2, 'mkkim' );
1 row created.
gSQL> SAVEPOINT sp2;
Savepoint created.
gSQL> INSERT INTO t1 VALUES ( 3, 'xcom' );
1 row created.
gSQL> ROLLBACK TO SAVEPOINT sp2;
Rollback complete.
```

```
gSQL> SELECT * FROM t1;
ID NAME
-- ------
 1 leekmo
 2 mkkim
2 rows selected.
gSQL> ROLLBACK TO SAVEPOINT sp1;
Rollback complete.
gSQL> SELECT * FROM t1;
ID NAME
-- ------
 1 leekmo
1 row selected.
gSQL> ROLLBACK WORK;
Rollback complete.
gSQL> SELECT * FROM t1;
no rows selected.
```

The implicit partial rollback is undoing only the changes of the corresponding statement when an error oc curs while executing the statement.
The following example describes the implicit partial rollback.
If a unique constraint is violated, only the INSERT statement is rolled back and the previous changes of th e transaction are retained.

```
gSQL> ALTER TABLE t1 ADD CONSTRAINT t1_uk UNIQUE(id);
Table altered.
gSQL> COMMIT;
Commit complete.
gSQL> INSERT INTO t1 VALUES ( 4, 'egonspace' );
1 row created.
gSQL> INSERT INTO t1 VALUES ( 1, 'jhkim' );
ERR-40002(16057): unique constraint (PUBLIC.T1_UK) violated
gSQL> SELECT * FROM t1;
ID NAME
-- ---------
 1 leekmo
 2 mkkim
 3 xcom
 4 egonspace
4 rows selected.
```

Session Control

Session is a logical object to manage the user's state information who accesses the database. Session control statement changes the session properties.

Session control statements are classified as follows.

Table 12-4 Session control statements

Statement	Description	Refer to
SET SESSION CHARACTERISTICS	It controls the transaction properties in the session.	**SET SESSION CHARACTERISTICS AS transaction_mode**
SET TIME ZONE	It alters the time zone of the session.	**SET TIME ZONE**
SET SESSION AUTHORIZATION	It alters the user of the session.	**SET SESSION AUTHORIZATION user_identifier**
SET SCHEMA	It alters the session schema.	**SET SCHEMA schema_name**
ALTER SESSION SET	It alters the session property.	**ALTER SESSION SET property_name**

SET TRANSACTION statement is the transaction controlling statement and SET SESSION CHARACTERISTICS statement is session controlling statement. Both SET TRANSACTION statement and SET SESSION CHARACTERISTICS statement control the transaction properties.
The difference between SET TRANSACTION statement and SET SESSION CHARACTERISTICS statement is that SET TRANSACTION statement is applied only to a transaction which will be executed next. On the other hand, SET SESSION CHARACTERISTICS statement is applied for all transactions which occur in the session afterwards.

The following is a result of CURRENT_TIMESTAMP statement which obtains the current date and time after changing the time zone by using SET TIME ZONE statement.

```
gSQL> SELECT CURRENT_TIMESTAMP FROM DUAL;
CURRENT_TIMESTAMP
---------------------------------
2014-07-21 11:42:49.828276 +09:00
1 row selected.
gSQL> SET TIME ZONE '+00:00';
Session set.
gSQL> SELECT CURRENT_TIMESTAMP FROM DUAL;
CURRENT_TIMESTAMP
---------------------------------
2014-07-21 02:43:03.437940 +00:00
1 row selected.
```

System Control

System control statements manage the database system, and they are classified as follows.

Table 12-5 System control statements

Statement	Description	Refer to
ALTER SYSTEM {OPEN|MOUNT} DATABASE	It starts up the database.	**ALTER SYSTEM {MOUNT | OPEN} DATABASE**
ALTER SYSTEM CHECKPOINT	It performs the checkpoint.	**ALTER SYSTEM CHECKPOINT**
ALTER SYSTEM KILL SESSION	It forcibly terminates the specific session.	**ALTER SYSTEM [KILL | DISCONNECT] SESSION**
ALTER SYSTEM SWITCH LOGFILE	It switches the log file.	**ALTER SYSTEM SWITCH LOGFILE**
ALTER SYSTEM SET	It sets the system properties.	**ALTER SYSTEM SET property_name**
ALTER SYSTEM RESET	It removes the system properties.	**ALTER SYSTEM RESET property_name**

The following is an example of enquiring sessions connected to the database, and terminating the specific session.

```
gSQL> SELECT USER_NAME, SESSION_ID, SERIAL_NO, SESSION_STATUS, PROGRAM_NAME FROM V$SESSION
WHERE USER_NAME = 'TEST';
USER_NAME SESSION_ID SERIAL_NO SESSION_STATUS PROGRAM_NAME
--------- ---------- --------- -------------- ------------
TEST              62        49 CONNECTED      gsql
TEST              65       109 CONNECTED      gsqlnet
TEST              66       130 CONNECTED      gsql
3 rows selected.
gSQL> ALTER SYSTEM DISCONNECT SESSION 65, 109;
System altered.
```

12.5 Processing SQL in Cluster

This chapter describes how to process various SQL statements in a cluster environment.

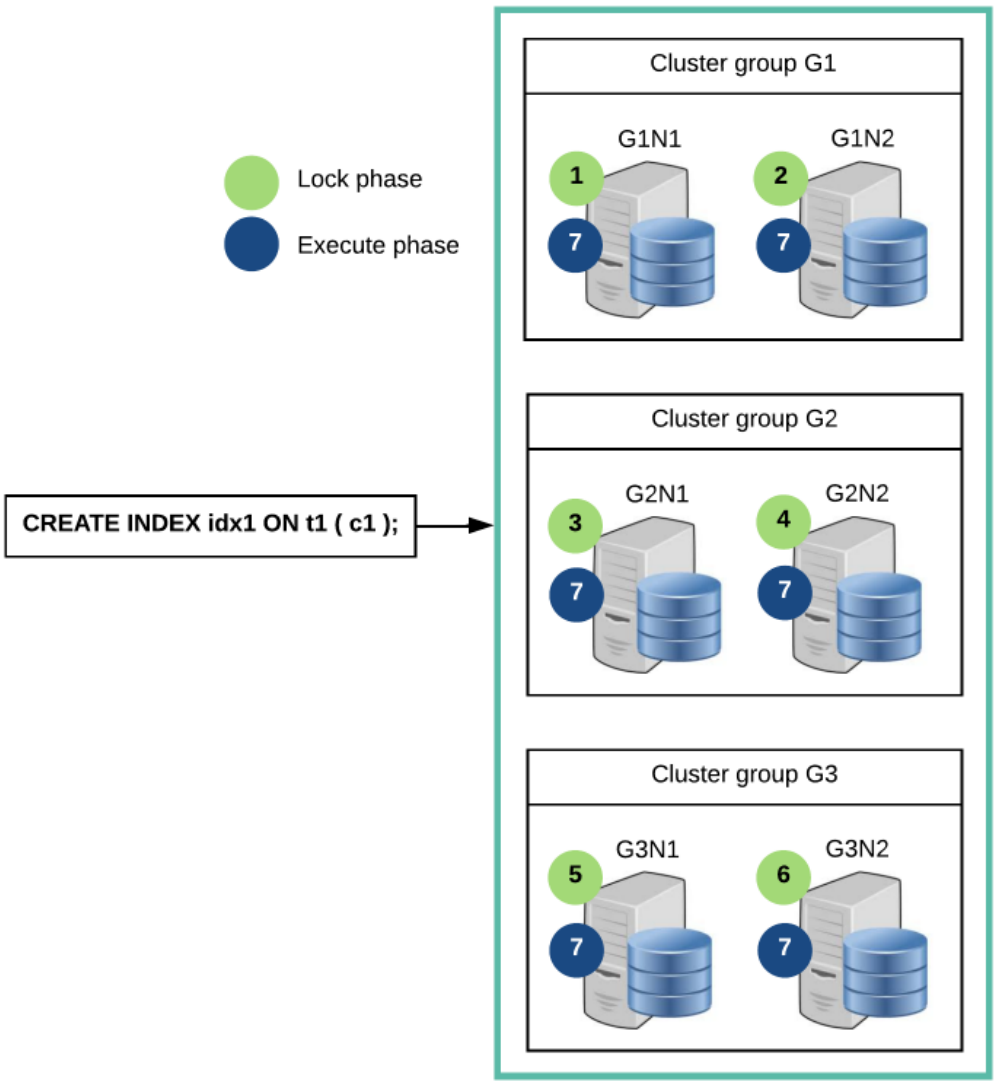
Processing DDL in Cluster

DDL Processing Procedure in Cluster

GOLDILOCKS cluster does not have a separate meta server, and a user can perform DDL in any cluster me mber configuring the cluster system.

DDL is executed following the procedure below in a cluster environment.

Figure 1 Processing DDL in cluster



DDL is performed through two phases, which are a lock phase and and execution phase. On the lock pha se, a lock which is required for performing DDL is acquired and DDL is sequentially performed on every cl uster member. On the execute phase, DDL is simultaneously performed for every cluster member.

DDL is completed when DDL is successfully performed on every cluster members. If DDL fails on a specific cluster member, then DDL operations on every cluster members are cancelled. DDL can not be performed if an error occurred in any cluster member. All cluster member synchronize meta information for objects t hrough this process.

Simultaneous DDL Execution

The cluster object DDL which changes the cluster system configuration and the SQL object DDL can not b e simultaneously performed. The availability of simultaneously performing the cluster object DDL and the SQL object DDL are as follows.

Table 12-6 Availability of simultaneously performing DDL

DDL	Cluster object DDL	SQL object DDL
Cluster object DDL	X	X
SQL object DDL	X	O

The following DDL can not be simultaneously performed as given above.

- Cluster object DDL and cluster object DDL
  - (X) CREATE CLUSTER GROUP g2 CLUSTER MEMBER g2n1 HOST '192.168.0.21' PORT 10210;
  - (X) ALTER CLUSTER GROUP g1 ADD CLUSTER MEMBER g1n2 HOST '192.168.0.12' PORT 10120;
- Cluster object DDL and SQL object DDL
  - (X) CREATE CLUSTER GROUP g2 CLUSTER MEMBER g2n1 HOST '192.168.0.21' PORT 10210;
  - (X) CREATE TABLE t1 (c1 INTEGER);
- SQL object DDL and SQL object DDL
  - (O) CREATE TABLE t1 (c1 INTEGER);
  - (O) CREATE TABLE t2 (a1 INTEGER);

Processing SELECT in Cluster

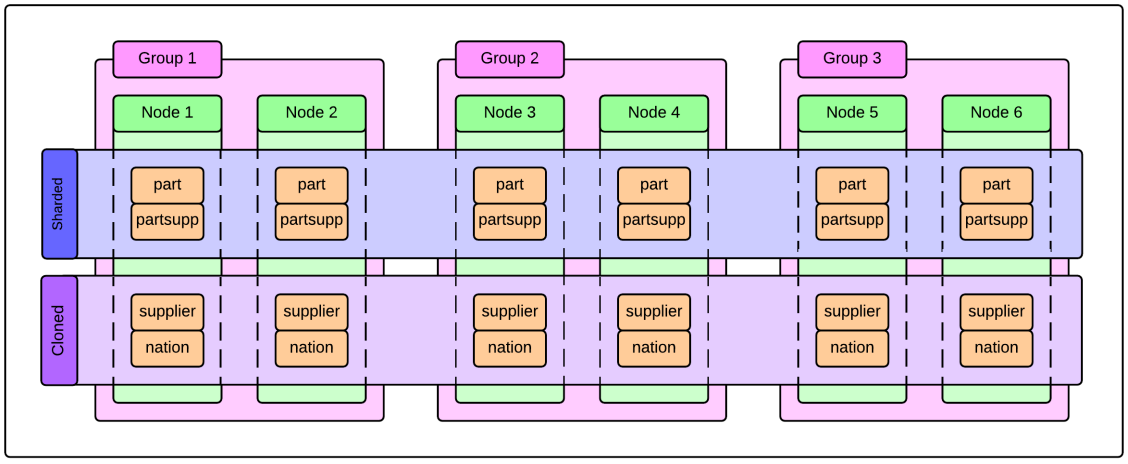
Generally, a query processing of the cluster is similar to that of the stand alone. However, when the data exist on both a local server and a remote server, it is different that the cluster requests the query processi ng to a remote server and receives the result from it.

There are a sharded table and a cloned table (refer to **Cluster Table and Shard**) in a cluster environment, and each table data is stored in a local server and a remote server. The sharded table data is dividedly stor ed in a group, and its duplicated data is stored in members of the same group. The data is duplicated and

stored in every group and member of a cloned table.

The following figure describes a 3 x 2 GOLDILOCKS cluster and tables stored in that cluster.

Figure 2 3 x 2 cluster configuration and tables



The following DDL creates the tables given above.

```
CREATE TABLE part
(
    p_partkey     INTEGER
  , p_name        VARCHAR(55)
  , p_brand       CHAR(10)
  , p_type        VARCHAR(25)
  , p_size        INTEGER
  , p_retailprice NUMERIC(12,2)
  , CONSTRAINT part_pk PRIMARY KEY( p_partkey ) INDEX part_pk_index
)
    SHARDING BY HASH(p_partkey)
    SHARD COUNT 3;
CREATE TABLE partsupp
(
    ps_partkey    INTEGER
  , ps_suppkey    INTEGER
  , ps_availqty   INTEGER
  , ps_supplycost NUMERIC(12,2)
  , CONSTRAINT partsupp_pk PRIMARY KEY( ps_partkey, ps_suppkey ) INDEX partsupp_pk_index
)
   SHARDING BY HASH(ps_partkey)
   SHARD COUNT 3;
CREATE TABLE supplier
(
```

```
    s_suppkey     INTEGER
  , s_name        CHAR(25)
  , s_nationkey   INTEGER
  , s_phone       CHAR(15)
  , CONSTRAINT supplier_pk PRIMARY KEY( s_suppkey ) INDEX supplier_pk_index
)  CLONED;
CREATE TABLE nation
(
    n_nationkey   INTEGER
  , n_name        CHAR(25)
)  CLONED;
```

In the figure above, the part table and the partsupp table is a sharded table, and their data are dividedly s tored per a group. The supplier table is a cloned table and its data is duplicated and stored in every node.

GOLDILOCKS in a cluster environment processes a query depending on the table type and the location of search target data. Cluster processes the query according to the method to collect the data and the meth od to manipulate the fetched data.

Query Processing Method in a Cluster

A cluster should collect data from both a local server and a remote server to process a query. It creates SQ L statement, delivers it to a local server and a remote server, then collects the results of query processing t o collect the data.

The data is manipulated according to the statement type of the collected data. For example, the cluster q uery for *group by* statement is processed by grouping after collecting the data.

Collecting and manipulating data is performed by the plan node called as a cluster puller.

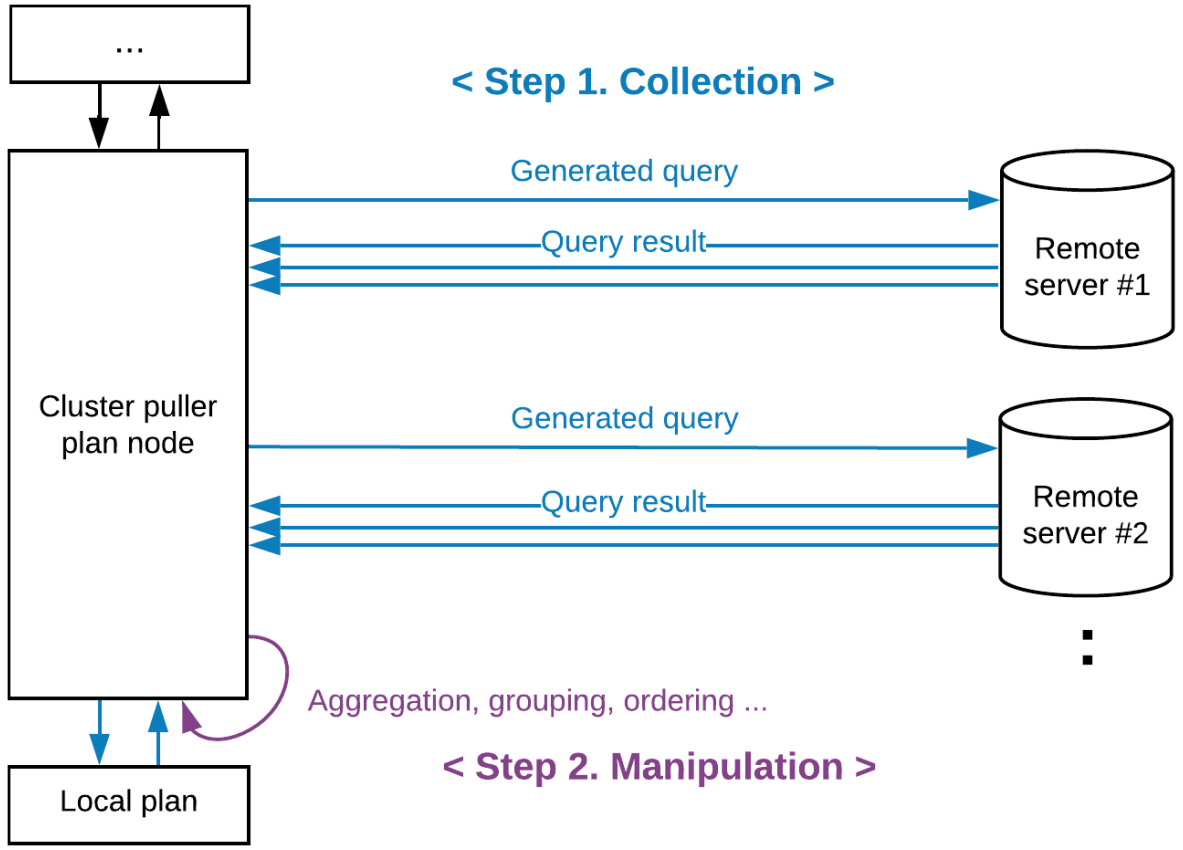
Cluster Puller

Cluster puller node collects and manipulates the data.

- The data collection is a process to collect the data from multiple servers.
- The data manipulation is a process to create a new data based on the collected data.

The following is a process to perform the cluster puller.

Figure 3 Performing cluster puller



A cluster puller is classified according to the data fetching method from a local server and whether to dist inguish a remote delivering server when fetching data from a remote server.

A cluster puller is divided into the following three plan nodes.

Table 12-7 Cluster puller node

Cluster puller node	How to collect the local data	How to collect the remote data
Plan based cluster	It performs a plan configured in t he local.	It performs a generated query. (It collects the result without distinguishing the re mote server)
Single cluster	It performs a generated query.	It performs a generated query. (It collects the result without distinguishing the re mote server)
Multiple cluster	It performs a generated query.	It perform a generated query. (It collects the result by distinguishing the remote s erver)

A generated query is configured to collect or update the data while performing a query provided by a use r. For more information, refer to **Generated Query**.

Those paragraphs below describes how to collect the data per each feature in available in each cluster pul ler node and how to manipulate the data.

How to Collect Data

- By pass: It collects the processed results in the delivered sequence.
- Merge sort: It sequentially collects the processed results according to the given order of sorting.

How to Manipulate the Collected Data

- No manipulation: It does not manipulate the data.
- Aggregation: It aggregate the collected data.
- Grouping: It groups the collected data.
- Ordering: It orders for the collected data.
- Intersect key group: It classifies the collected data based on the delivering server, and groups each ser ver data by a given key, then intersects them in group unit.
- Distinct key group: It classifies the collected data based on the delivering server, and groups each serv er data by a given key, then performs distinct on them in group unit.

Cluster Puller Feature

The cluster puller node collects data from a local server or a remote server. It determines the server from which to collect he data, then fetches data by transferring a generated query.

The following is an example of retrieving a single table without *where* condition.

```
gSQL> \EXPLAIN PLAN SELECT c1 FROM t_shard_1;
C1
--
 1
 2
 3
3 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                          |              ROWS |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                          |                 3 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")               |                 3 |
```

```
┆    2  ┆      PLAN BASED CLUSTER                    ┆ LOCAL/REMOTE     3 ┆
┆    3  ┆        INDEX ACCESS ("T_SHARD_1", "IDX")   ┆ (         1)     1 ┆
================================================================================
     1  -  TARGET : T_SHARD_1.C1
     2  -  SQL : SELECT /*+ INDEX( _A1, "PUBLIC"."IDX" ) */ "_A1"."C1" FROM
"PUBLIC"."T_SHARD_1"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 1 rows, G3(G3N1,G3N2) 1 rows
     3  -  RANGE SHARD ( # 3 )
           READ INDEX COLUMN : T_SHARD_1.C1
<<<  end print plan
```

After performing the query, a plan based cluster is used as cluster puller. The plan based cluster in the ex ecution plan above is a plan whose idx is 2 in <Execution Plan>.

The following information is output on ROWS field of the cluster puller.

- LOCAL ONLY n: The number of data fetched from a local server.
- REMOTE ONLY n: The number of data fetched from a remote server.
- LOCAL/REMOTE n: The total number of data fetched from a local server and a remote server.

The following is the detailed information about PLAN BASED CLUSTER.

- SQL: Generated query
- TARGET DOMAIN: The target group and the target member to transfer a generated query and the n umber of data fetched from the group

Selecting Target Server to Perform in Cluster Puller

It analyzes the following information and selects the target to perform a generated query.

- Table replication placement strategy
- **Cluster Domain**
- **Reducing Target Domain of Cluster Puller**

It analyzes the information listed above for each tables included in a generated query, and find a server in common, then selects it as a target server to perform the generated query. They are classified as target d omains on the cluster puller node.

The following is a summary of the replication placement strategy of defined tables to describe **Generated Query**.

```
t_shard_1 (shard table)  : at cluster group G1, G2, G3
t_shard_2 (shard table)  : at cluster group G1, G3
t_clone_1 (cloned table) : at cluster group G1, G2, G3 (cluster wide)
```

```
t_clone_2 (cloned table) : at cluster group G2, G3
```

The following is an example of selecting the execution target server according to the table replication pla cement strategy.

- Execution target server: All servers in G1, G3

```
gSQL> SELECT c1 FROM t_shard_2;
```

- Execution target server: All servers in G1, G2, G3

```
gSQL> SELECT c1 FROM t_clone_1;
```

- Execution target server: All servers in G1, G3 (Commonly included cluster group)

```
gSQL> SELECT c1 FROM t_shard_2, t_clone_1;
```

The following is an example of selecting the execution target server according the description of the clust er domain.

- Execution target server: All servers in G1, G2, G3

```
gSQL> SELECT c1 FROM t_shard_1@GLOBAL;
```

- Execution target server: The server to which the user query was requested

```
gSQL> SELECT c1 FROM t_shard_1@LOCAL;
```

- Execution target server: All servers in G2

```
gSQL> SELECT c1 FROM t_shard_1@G2;
```

- Execution target server: G3N1 server

```
gSQL> SELECT c1 FROM t_shard_1@G3N1;
```

- Execution target server: Target server does not exist

```
gSQL> SELECT c1 FROM t_shard_1@G4;
```

The following is an example of selecting the execution target server according the sharding key search co ndition.

- Execution target server: All servers in G1, G2, G3

```
gSQL> SELECT c1 FROM t_shard_1;
```

- Execution target server: All servers in G1, G2, G3

```
gSQL> SELECT c1 FROM t_shard_1 WHERE c1 = 1;
```

- Execution target server: All servers in G3

```
gSQL> SELECT c1 FROM t_shard_1 WHERE shard_key = 500;
```

- Execution target server: Target server does not exist

```
gSQL> SELECT c1 FROM t_shard_1 WHERE shard_key = 100 AND shard_key = 500;
```

It looks for servers in common through the table replication placement strategy and cluster domain, shard ing key searching strategy when performing the generated query searching for the data. If multiple server s in the same cluster group are targets to perform, then the generated query is performed for only one ac cessible server per each cluster group considering the network condition of when it is performed.

The following is an example of selecting a target server to perform.

- Table replication placement strategy: All servers in G1, G2, G3

- Cluster domain
- t_shard_1: All servers in G2
- t_clone_1: All servers in G1, G2, G3
- join: All servers in G1, G2, G3

- Sharding key searching strategy: All servers in G2

- Target servers to perform the generated query: A single server in G2

```
gSQL> \EXPLAIN PLAN ONLY SELECT * FROM t_shard_1@G2, t_clone_1 WHERE t_shard_1.shard_key =
300;
>>>  start print plan
< Execution Plan >
=================================================================
|  IDX  |  NODE DESCRIPTION                      |           ROWS |
-----------------------------------------------------------------
|    0  |  SELECT STATEMENT                      |              0 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")           |              0 |
|    2  |      PLAN BASED CLUSTER                |              0 |
|    3  |        NESTED JOIN (INNER JOIN)        |              0 |
```

```
¦    4  ¦          TABLE ACCESS ("T_SHARD_1")    ¦             0 ¦
¦    5  ¦          TABLE ACCESS ("T_CLONE_1")    ¦             0 ¦
=============================================================
     1  -  TARGET : T_SHARD_1.SHARD_KEY, T_SHARD_1.C1, T_CLONE_1.C1
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE USE_NL_IN( _A1 ) FULL( _A2 ) FULL( _A1 ) */
"_A2"."SHARD_KEY", "_A2"."C1", "_A1"."C1" FROM ( "PUBLIC"."T_SHARD_1"@LOCAL AS "_A2" INNER
JOIN "PUBLIC"."T_CLONE_1"@LOCAL AS "_A1" ON true ) ALIAS "_A3" WHERE "_A2"."SHARD_KEY" = :_V0
           TARGET DOMAIN : G2(G2N1,G2N2) 0 rows
     3  -  JOINED COLUMN : T_SHARD_1.SHARD_KEY, T_SHARD_1.C1, T_CLONE_1.C1
     4  -  RANGE SHARD ( # 3 )
           READ COLUMN : T_SHARD_1.SHARD_KEY, T_SHARD_1.C1
             PHYSICAL FILTER : T_SHARD_1.SHARD_KEY = 300
     5  -  CLONED
           READ COLUMN : T_CLONE_1.C1
<<<  end print plan
```

Utilizing Cluster Puller

It selects a cluster puller node according to the data manipulation method. If the data manipulation is not required, then it can use a plan based cluster or a single cluster. If the ordering for the results are required, then it uses multiple clusters. The data collection method is determined according to the data manipulati on method.

Table 12-8 Features supported by cluster puller node

Cluster	How to collect the data	How to manipulate the data
Plan based cluster	By pass	No manipulation
Single cluster	By pass	No manipulation Aggregation Grouping
Multiple cluster	Merge sort	Ordering Grouping Intersect key group Distinct key group

Cluster Puller Plan Node

The cluster puller plan node is classified according to the data collection method from a local server and a remote server. For more information about the classification of cluster puller plan nodes according to the data collection, refer to **Cluster Puller Plan Node**.

Plan Based Cluster

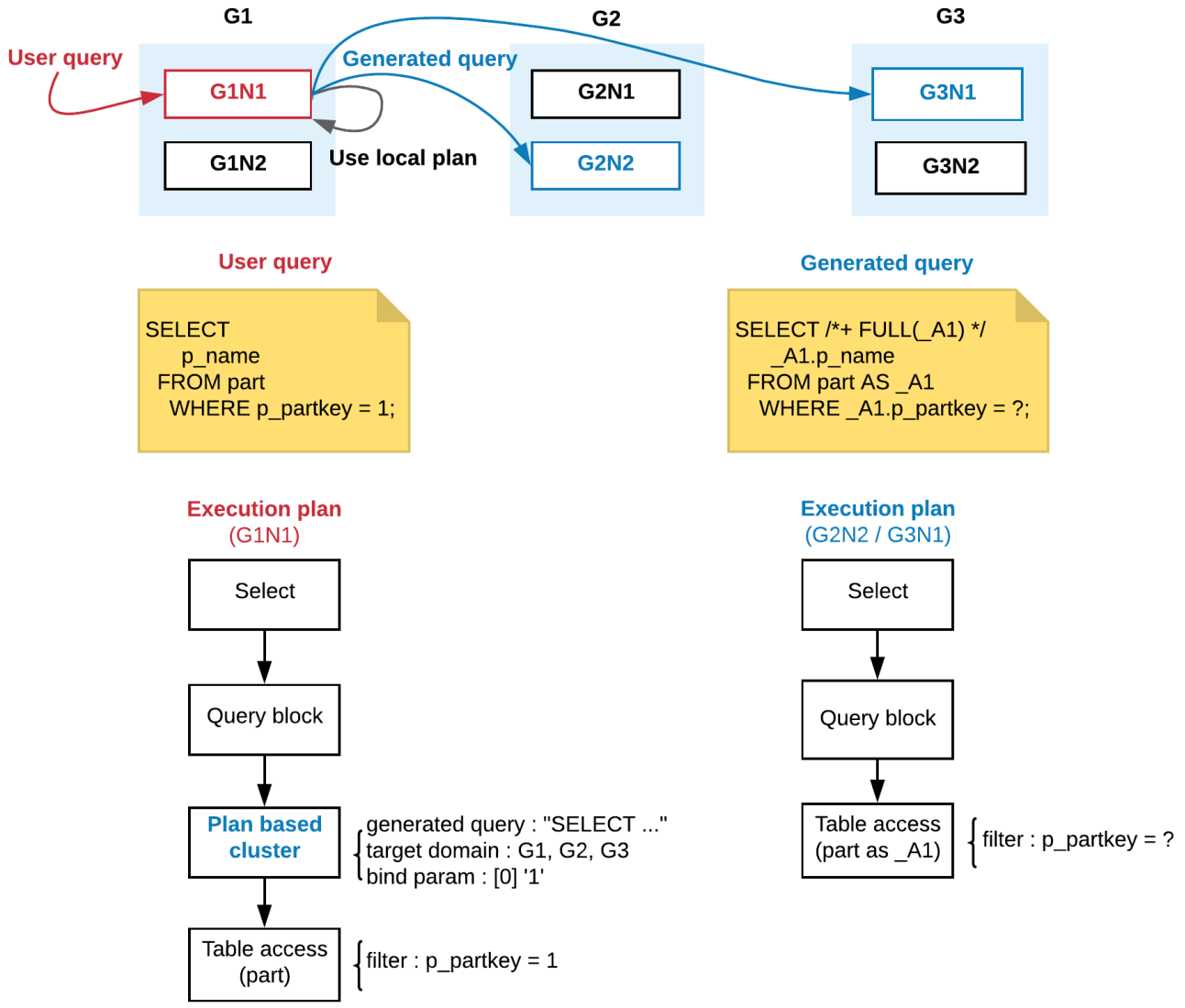
It collects the execution results in an order received from the data collection phase.

- Collecting local data: It performs the plan configured in local.
- Collecting remote data: It performs the generated query.

It does not manipulate anything except for applying a filter to the collected data. (No manipulation)

The following is an example of performing the plan based cluster.

Figure 4 Plan based cluster



Single Cluster

It collects the execution results in an order received from the data collection phase.

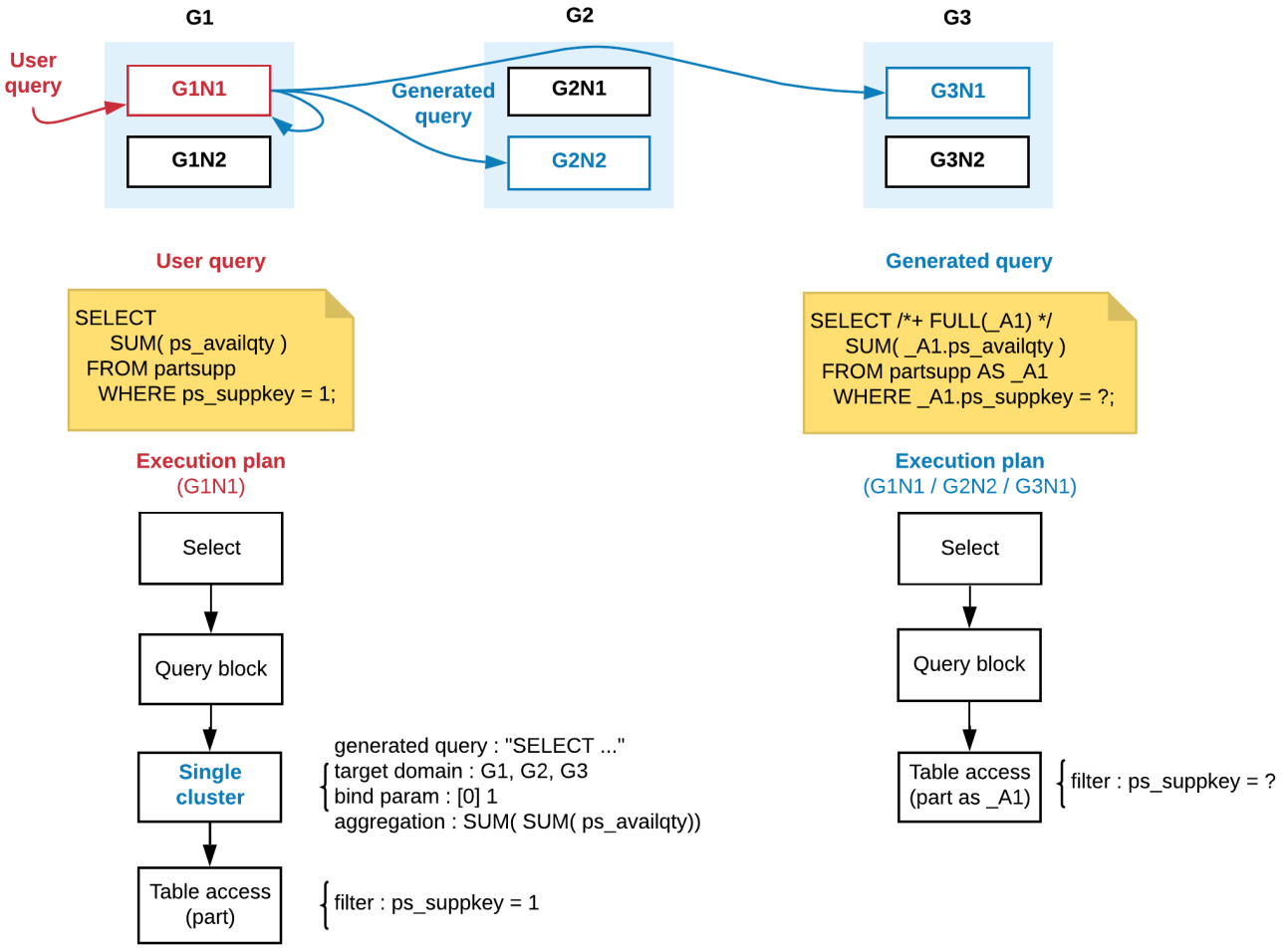
- Collecting local data: It performs the generated query.
- Collecting remote data: It performs the generated query.

A single cluster supports the following data manipulation methods.

- No manipulation
- Aggregation
- Grouping

The following is an example of performing a single cluster.

Figure 5 Single cluster



Multiple Clusters

Multiple clusters perform the generated query by using each different cluster executor per group. They co llect data by performing merge sort the data received with all cluster executors on the data collection pha se.

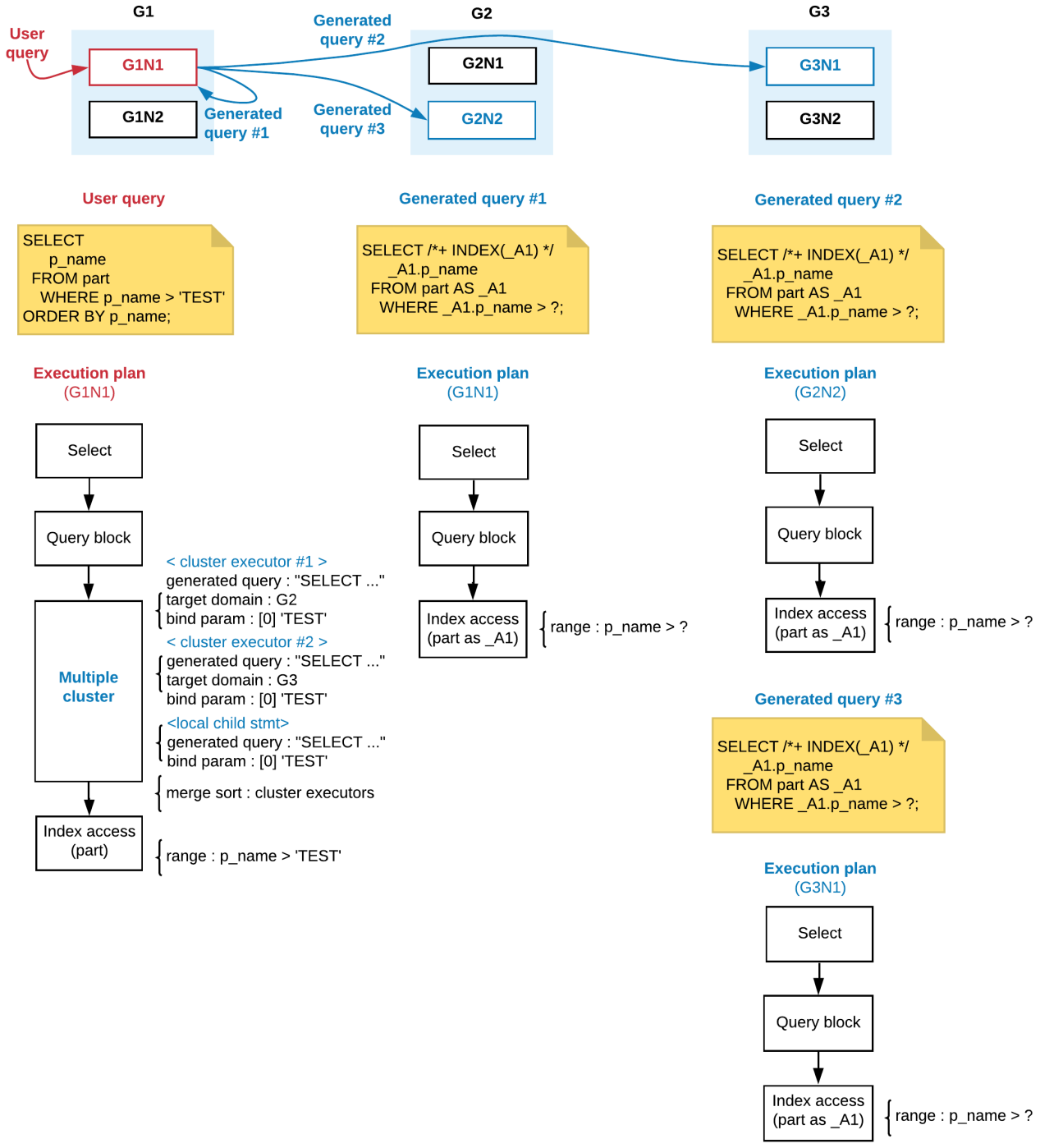
- Collecting local data: It performs the generated query.
- Collecting remote data: It performs the generated query.

Multiple clusters support the following data manipulation methods.

- Ordering
- Grouping
- Intersect key group
- Distinct key group

The following is an example of performing multiple clusters.

Figure 6 Multiple clusters



Cluster Domain

Cluster domain collects data only from limited servers in cluster environment. For example, if it queries as follows by setting the cluster group corresponding to employees in a specific range as a cluster domain, t hen it can find specific employees from the sharded table which consists of salary class.

- It configures a sharded table per a salary class.

```
gSQL> CREATE TABLE t1( name VARCHAR(128), salary INTEGER )
    SHARDING BY RANGE( salary )
        SHARD s1 VALUES LESS THAN ( 200 )       AT CLUSTER GROUP G1,
        SHARD s2 VALUES LESS THAN ( 400 )       AT CLUSTER GROUP G2,
        SHARD s3 VALUES LESS THAN ( MAXVALUE )  AT CLUSTER GROUP G3;
Table created.
gSQL> INSERT INTO t1 VALUES ( 'A', 500 );
1 row created.
gSQL> INSERT INTO t1 VALUES ( 'B', 100 );
1 row created.
gSQL> INSERT INTO t1 VALUES ( 'C', 300 );
1 row created.
```

- It searches for employees in a specific salary range.

```
gSQL> SELECT name, salary FROM t1@G2;
NAME SALARY
---- ------
C       300
1 row selected.
```

Cluster domain is defined by targeting a table or a view specified in **from clause** and referring to **〈cluster domain〉**, and one of the followings can be selected.

- Cluster member performing a user query
- Cluster member performing a user query targeting offline table
- All cluster groups
- A cluster group
- A cluster member

When a cluster member is selected as a cluster domain, then it access the corresponding server, then coll ects data. If the server does not have the data distribution for the target table, then the query result does not exist.

It supports *@LOCAL_OFFLINE* cluster domain to retrieve the offline table data from the server to which th

e user is connected. If the table is online, then an error occurs.

If a cluster group is selected as a cluster domain, then it has the data distribution for the corresponding ta ble in the group, and it collects data by accessing to a server which can communicate with. If an accessibl e server does not exist, then the query result does not exist.

If all cluster groups are selected as a cluster domain, then data from each cluster group are collected and transferred to a user.

A cluster domain can not be used to determine the data update target. In other words, <cluster domain> can not be applied to a target table of DML.

A cluster domain for a target table of DML causes a syntax error as follows.

```
gSQL> INSERT INTO t1@LOCAL VALUES ( 1 );
ERR-42000(16062): syntax error :
INSERT INTO t1@LOCAL VALUES ( 1 )
             *
ERROR at line 1:
gSQL> UPDATE t1@GLOBAL SET c1 = 1;
ERR-42000(16062): syntax error :
UPDATE t1@GLOBAL SET c1 = 1
         *
ERROR at line 1:
gSQL> DELETE FROM t1@G1;
ERR-42000(16062): syntax error :
DELETE FROM t1@G1
             *
ERROR at line 1:
```

A cluster domain for a target table of SELECT FOR UPDATE causes a syntax error as follows.

- Cluster domain for the data collection target

```
gSQL> SELECT c1 FROM t1@LOCAL;
I1
--
 1
1 row selected.
```

- Cluster domain for the data update target

```
gSQL> SELECT c1 FROM t1@LOCAL FOR UPDATE;
ERR-42000(16062): syntax error :
```

```
SELECT c1 FROM t1@LOCAL FOR UPDATE
                *
ERROR at line 1:
```

Reducing Target Domain of Cluster Puller

The methods to reduce the target domain to be processed by transferring a generated query are as follo ws.

- **Cluster Domain**: It reduces the target domain by specifying the domain.
- Shard key filter: It reduces the target domain by providing the sharding key condition.
- Rowinfo domain filter: It reduces the target domain by providing the pseudo column condition relate d to a table.

The combination of the shard key filter and the rowinfo domain filter are called as the domain filter.

The following is an example of retrieving a single table by specifying the retrieving target domain in the t able.

```
gSQL> \EXPLAIN PLAN SELECT c1 FROM t_shard_1@G2;
C1
--
 2
1 row selected.
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                          |              ROWS |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                          |                 1 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")               |                 1 |
|    2  |      PLAN BASED CLUSTER                    | REMOTE ONLY     1 |
|    3  |        INDEX ACCESS ("T_SHARD_1", "IDX")   | (         0)    0 |
================================================================================
     1  -  TARGET : T_SHARD_1.C1
     2  -  SQL : SELECT /*+ INDEX( _A1, "PUBLIC"."IDX" ) */ "_A1"."C1" FROM
"PUBLIC"."T_SHARD_1"@LOCAL AS "_A1"
           TARGET DOMAIN : G2(G2N1,G2N2) 1 rows
     3  -  RANGE SHARD ( # 3 )
           READ INDEX COLUMN : T_SHARD_1.C1
<<<  end print plan
```

When the retrieving target domain is specified, then TARGET DOMAIN is reduced to G2 as in the exampl

e above. For more information about domain, refer to **Cluster Domain**.

The following is an example of restricting TARGET DOMAIN by adding the shard key condition on where clause.

```
gSQL> \EXPLAIN PLAN SELECT c1 FROM t_shard_1 WHERE shard_key = 555;
C1
--
 3
1 row selected.
>>>  start print plan
< Execution Plan >
================================================================================
¦  IDX  ¦  NODE DESCRIPTION                      ¦                ROWS ¦
--------------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                      ¦                   1 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")          ¦                   1 ¦
¦    2  ¦      PLAN BASED CLUSTER               ¦ REMOTE ONLY       1 ¦
¦    3  ¦        TABLE ACCESS ("T_SHARD_1")     ¦                   0 ¦
================================================================================
     1  -  TARGET : T_SHARD_1.C1
     2  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."C1" FROM "PUBLIC"."T_SHARD_1"@LOCAL AS "_A1"
WHERE "_A1"."SHARD_KEY" = :_V0
           TARGET DOMAIN : G3(G3N1,G3N2) 1 rows
     3  -  RANGE SHARD ( # 3 )
           READ COLUMN : T_SHARD_1.SHARD_KEY, T_SHARD_1.C1
             PHYSICAL FILTER : T_SHARD_1.SHARD_KEY = 555
<<<  end print plan
```

When the shard key condition which has a constant value is added, then TARGET DOMAIN is reduced as i n the example above.

The retrieving target group can be restricted by using the shard key condition. The shard key condition w hich has a constant value can determine the retrieving target while configuring the plan. The shard key c ondition which is not a constant value determines the retrieving target group at the time of executing the query.
In other words, adding the shard key condition which is not a constant value does not reduce TARGET D OMAIN. Instead, SHARD KEY FILTER condition which is the information to determine the domain is confi gured at the time of executing the query.

The following is an example of restricting TARGET DOMAIN by adding the shard key condition which is n ot a constant value to where clause.

```
gSQL> VAR V1 INTEGER
gSQL> EXEC :V1 := 555
gSQL> \EXPLAIN PLAN SELECT c1 FROM t_shard_1 WHERE shard_key = :V1;
C1
--
 3
1 row selected.
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                      ¦                ROWS ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                      ¦                   1 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")          ¦                   1 ¦
¦    2  ¦      PLAN BASED CLUSTER               ¦ REMOTE ONLY       1 ¦
¦    3  ¦        TABLE ACCESS ("T_SHARD_1")     ¦                   0 ¦
=============================================================================
     1  -  TARGET : T_SHARD_1.C1
     2  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."C1" FROM "PUBLIC"."T_SHARD_1"@LOCAL AS "_A1"
WHERE "_A1"."SHARD_KEY" = :_V0
           TARGET DOMAIN : G1(G1N1,G1N2) 0 rows, G2(G2N1,G2N2) 0 rows, G3(G3N1,G3N2) 1 rows
             SHARD KEY FILTER : ( T_SHARD_1.SHARD_KEY = :V1 )
     3  -  RANGE SHARD ( # 3 )
           READ COLUMN : T_SHARD_1.SHARD_KEY, T_SHARD_1.C1
             PHYSICAL FILTER : T_SHARD_1.SHARD_KEY = :V1
<<<  end print plan
```

TARGET DOMAIN can also be reduced by providing the **Pseudo Columns** condition related to the table. It can be used as a domain filter only when using equal (=) condition for the pseudo column.

Table 12-9 Pseudo column which can be used as the domain filter

Pseudo column	Whether to use the pseudo column as the domain filter
CURRVAL	Not available
NEXTVAL	Not available
ROWNUM	Not available
ROWID	Available
CLUSTER_GROUP_ID	Available
CLUSTER_MEMBER_ID	Available
CLUSTER_GROUP_NAME	Available
CLUSTER_NAME_ID	Available
CLUSTER_SHARD_ID	Available

The following is an example of restricting TARGET DOMAIN by adding the pseudo column condition.

```
gSQL> \EXPLAIN PLAN SELECT c1 FROM t_shard_1 WHERE cluster_group_name = 'G3';
C1
--
 3
1 row selected.
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                          |                ROWS |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                          |                   1 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")               |                   1 |
|    2  |      PLAN BASED CLUSTER                    | REMOTE ONLY       1 |
|    3  |        INDEX ACCESS ("T_SHARD_1", "IDX")   | (         0)      0 |
================================================================================
     1  -  TARGET : T_SHARD_1.C1
     2  -  SQL : SELECT /*+ INDEX( _A1, "PUBLIC"."IDX" ) */ "_A1"."C1" FROM
"PUBLIC"."T_SHARD_1"@LOCAL AS "_A1" WHERE "_A1"."CLUSTER_GROUP_NAME" = :_V0
           TARGET DOMAIN : G3(G3N1,G3N2) 1 rows
     3  -  RANGE SHARD ( # 3 )
           READ INDEX COLUMN : T_SHARD_1.C1
             LOGICAL KEY FILTER : T_SHARD_1.CLUSTER_GROUP_NAME = 'G3'
<<<  end print plan
```

The pseudo column condition using the constant value reduces TARGET DOMAIN.

The following is an example of using the pseudo column condition which is not a constant value.

```
gSQL> VAR V1 VARCHAR( 10 )
gSQL> EXEC :V1 := 'G3N1'
gSQL> \EXPLAIN PLAN SELECT c1 FROM t_shard_1 WHERE cluster_member_name = :V1;
C1
--
 3
1 row selected.
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                          |                ROWS |
--------------------------------------------------------------------------------
```

```
┆    0  ┆  SELECT STATEMENT                          ┆                 1 ┆
┆    1  ┆    QUERY BLOCK ("$QB_IDX_2")               ┆                 1 ┆
┆    2  ┆      PLAN BASED CLUSTER                    ┆ REMOTE ONLY     1 ┆
┆    3  ┆        INDEX ACCESS ("T_SHARD_1", "IDX")   ┆ (       0)      0 ┆
=============================================================================
     1  -  TARGET : T_SHARD_1.C1
     2  -  SQL : SELECT /*+ INDEX( _A1, "PUBLIC"."IDX" ) */ "_A1"."C1" FROM
"PUBLIC"."T_SHARD_1"@LOCAL AS "_A1" WHERE "_A1"."CLUSTER_MEMBER_NAME" = :_V0
           TARGET DOMAIN : G1(G1N1,G1N2) 0 rows, G2(G2N1,G2N2) 0 rows, G3(G3N1,G3N2) 1 rows
             ROWINFO DOMAIN FILTER : T_SHARD_1.CLUSTER_MEMBER_NAME = :V1
     3  -  RANGE SHARD ( # 3 )
           READ INDEX COLUMN : T_SHARD_1.C1
             LOGICAL KEY FILTER : T_SHARD_1.CLUSTER_MEMBER_NAME = :V1
<<<  end print plan
```

Adding the pseudo column condition which is not a constant value does not reduce TARGET DOMAIN. In stead, ROWINFO DOMAIN FILTER condition which is the information to determine the domain is configur ed at the time of execution.

Generated Query

The generated query is generated to refer to or to update the data in another server when processing a q uery given by a user.

```
SELECT c1 FROM t1;
```

When the query is given by a user as above, then the server to which the query is given generates a simila r generated query to collects the t1 data from all related cluster groups as follows.

```
SELECT c1 FROM t1@LOCAL;
```

The following example table is configured to describe the generated query.

```
CREATE TABLE t_shard_1( shard_key INTEGER, c1 INTEGER )
    SHARDING BY RANGE( shard_key )
        SHARD s1 VALUES LESS THAN ( 200 )       AT CLUSTER GROUP G1,
        SHARD s2 VALUES LESS THAN ( 400 )       AT CLUSTER GROUP G2,
        SHARD s3 VALUES LESS THAN ( MAXVALUE )  AT CLUSTER GROUP G3;
CREATE TABLE t_shard_2( shard_key INTEGER, c1 INTEGER )
    SHARDING BY RANGE( shard_key )
        SHARD s1 VALUES LESS THAN ( 300 )       AT CLUSTER GROUP G1,
        SHARD s2 VALUES LESS THAN ( MAXVALUE )  AT CLUSTER GROUP G3;
```

```
CREATE TABLE t_clone_1( c1 INTEGER ) CLONED AT CLUSTER WIDE;
CREATE TABLE t_clone_2( c1 INTEGER ) CLONED AT CLUSTER GROUP G2, G3;
```

The generated query is a query which was configured again based on the cluster puller plan node.

Each cluster puller plan node configures the generated query for **data collection method** according to **Util izing Cluster Puller**.

Generated Query for No Manipulation

When the collected data is not manipulated, then the plan based cluster or the single cluster is used as th e cluster puller plan node.

The generated query is configured and output on the explain plan result for the cluster puller plan node which does not manipulate the data, but the information about manipulation is not output.

The following is an example of no manipulation using the plan based cluster.

```
gSQL> \EXPLAIN PLAN SELECT c1 FROM t_shard_1;
C1
--
 1
 3
 2
3 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
¦  IDX  ¦  NODE DESCRIPTION                          ¦             ROWS ¦
--------------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                          ¦                3 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")              ¦                3 ¦
¦    2  ¦      PLAN BASED CLUSTER                   ¦ LOCAL/REMOTE   3 ¦
¦    3  ¦        TABLE ACCESS ("T_SHARD_1")         ¦                1 ¦
================================================================================
     1  -  TARGET : T_SHARD_1.C1
     2  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."C1" FROM "PUBLIC"."T_SHARD_1"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 1 rows, G3(G3N1,G3N2) 1 rows
     3  -  RANGE SHARD ( # 3 )
           READ COLUMN : T_SHARD_1.C1
<<<  end print plan
```

The following is an example of no manipulation using a single cluster.

```
gSQL> \EXPLAIN PLAN SELECT t_shard_1.c1 FROM t_shard_1, t_shard_2 WHERE t_shard_1.shard_key =
t_shard_2.c1;
C1
--
 1
 2
2 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
| IDX | NODE DESCRIPTION                                  |           ROWS |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                                  |              2 |
|   1 |   QUERY BLOCK ("$QB_IDX_2")                       |              2 |
|   2 |     SINGLE CLUSTER                                | LOCAL/REMOTE 2 |
|   3 |       CLUSTER PUSHER ("_$NI_6")                   |              3 |
|   4 |         PLAN BASED CLUSTER                        | LOCAL/REMOTE 3 |
|   5 |           TABLE ACCESS ("T_SHARD_2")              |              2 |
|   6 |       SELECT STATEMENT                            |              0 |
|   7 |         QUERY BLOCK ("$QB_IDX_2")                 |              0 |
|   8 |           NESTED JOIN (INNER JOIN)                |              0 |
|   9 |             TABLE ACCESS ("T_SHARD_1" AS _A2)     |              1 |
|  10 |             PUSHER TABLE ACCESS ("_$NI_6" AS _A1) |              0 |
================================================================================
     1  -  TARGET : T_SHARD_1.C1
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE USE_NL_IN( _A1 ) FULL( _A2 ) FULL( _A1 ) */
"_A2"."C1" FROM ( "PUBLIC"."T_SHARD_1"@LOCAL AS "_A2" INNER JOIN
"SESSION_SCHEMA"."_$NI_6"@LOCAL AS "_A1" ON "_A2"."SHARD_KEY" = "_A1"."C1") ALIAS "_A3"
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 1 rows, G3(G3N1,G3N2) 0 rows
     3  -  SQL : DECLARE INSTANT TABLE "SESSION_SCHEMA"."_$NI_6" ( "C1" NUMBER(10, 0) )
           COLUMN : T_SHARD_2.C1 AS C1
           SHARDED : T_SHARD_2.C1
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 2 rows, G3(G3N1,G3N2) 0 rows
     4  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."C1" FROM "PUBLIC"."T_SHARD_2"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G3(G3N1,G3N2) 2 rows
     5  -  RANGE SHARD ( # 4 )
           READ COLUMN : T_SHARD_2.C1
     7  -  TARGET : _A2.C1
     8  -  JOINED COLUMN : _A2.SHARD_KEY, _A1.C1, _A2.C1
             ON FILTER : _A2.SHARD_KEY = _A1.C1
     9  -  RANGE SHARD ( # 3 )
```

```
           READ COLUMN : _A2.SHARD_KEY, _A2.C1
    10  -  READ COLUMN : _A1.C1
<<<  end print plan
```

Generated Query for Aggregation

The generated query for aggregation supports aggregation per each group. It aggregates the data collect ed through the generated query again, then configures the final result.

Table 12-10 How to manipulate aggregation data

Aggregation in user query	Operation form included in the generated query	Operation for the aggregation result
COUNT()	COUNT()	SUM(COUNT())
SUM()	SUM()	SUM(SUM())
AVG()	COUNT(), SUM()	SUM(SUM()) / SUM(COUNT())
MIN()	MIN()	MIN(MIN())
MAX()	MAX()	MAX(MAX())

The generated query configures a query including COUNT operation to process COUNT operation of the user query. It gathers the data collected through the generated query, and accumulates COUNT results p er each group. The accumulated value of COUNT result for every group becomes the result of user COUN T operation.

The following is an example of an aggregation using a single cluster.

```
gSQL> \EXPLAIN PLAN SELECT COUNT( c1 ) FROM t_shard_1;
COUNT( C1 )
-----------
          3
1 row selected.
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                              |            ROWS |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                              |               1 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                   |               1 |
|    2  |      SINGLE CLUSTER                            | LOCAL/REMOTE  1 |
|    3  |        SELECT STATEMENT                        |               1 |
|    4  |          QUERY BLOCK ("$QB_IDX_2")             |               1 |
|    5  |            TABLE ACCESS ("T_SHARD_1" AS _A1)   |               1 |
================================================================================
```

```
     1  -  TARGET : COUNT( T_SHARD_1.C1 )
     2  -  SQL : SELECT /*+ FULL( _A1 ) */ COUNT( "_A1"."C1" ) FROM "PUBLIC"."T_SHARD_1"@LOCAL
AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 1 rows, G3(G3N1,G3N2) 1 rows
           RE-AGGREGATION
             AGGREGATION : SUM( COUNT( T_SHARD_1.C1 ) )
     4  -  TARGET : COUNT( _A1.C1 )
     5  -  RANGE SHARD ( # 3 )
           READ COLUMN : _A1.C1
           AGGREGATION : COUNT( _A1.C1 )
<<<  end print plan
```

When DISTINCT is specified in the aggregation operation as if COUNT(DISTINCT c1), then the generated query including the aggregation can not be configured, and the cluster puller plan node does not support the data manipulation related to the aggregation.

The following is an example of COUNT(DISTINCT) operation.

```
gSQL> \EXPLAIN PLAN SELECT COUNT( DISTINCT c1 ) FROM t_shard_1;
COUNT( DISTINCT C1 )
--------------------
                   3
1 row selected.
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                              |            ROWS |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                              |               1 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                   |               1 |
|    2  |      AGGREGATION BY HASH                       |               1 |
|    3  |        PLAN BASED CLUSTER                      | LOCAL/REMOTE  3 |
|    4  |          TABLE ACCESS ("T_SHARD_1")            |               1 |
================================================================================
     1  -  TARGET : COUNT( DISTINCT T_SHARD_1.C1 )
     2  -  DISTINCT AGGREGATION : COUNT( DISTINCT T_SHARD_1.C1 )
     3  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."C1" FROM "PUBLIC"."T_SHARD_1"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 1 rows, G3(G3N1,G3N2) 1 rows
     4  -  RANGE SHARD ( # 3 )
           READ COLUMN : T_SHARD_1.C1
<<<  end print plan
```

When the aggregation including DISTINCT is used, the generated query of the cluster puller does not include the aggregation. The data collected by the cluster puller performs the aggregation operation through separate plan node.

Generated Query for Grouping

The generated query for grouping performs grouping per each group. It groups the data collected through the generated query again.

HAVING condition is applied after the grouping is completed on the cluster puller plan node.

The following is an example of grouping on the cluster puller plan node.

```
gSQL> \EXPLAIN PLAN SELECT COUNT( c1 ) FROM t_shard_1 AS A GROUP BY c1 HAVING MIN( shard_key )
> 1;
COUNT( C1 )
-----------
          1
          1
          1
3 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                              |            ROWS |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                              |               3 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                   |               3 |
|    2  |      SINGLE CLUSTER                            | LOCAL/REMOTE  3 |
|    3  |        SELECT STATEMENT                        |               1 |
|    4  |          QUERY BLOCK ("$QB_IDX_2")             |               1 |
|    5  |            GROUP HASH INSTANT                  |               1 |
|    6  |              TABLE ACCESS ("T_SHARD_1" AS _A1) |               1 |
================================================================================
     1  -  TARGET : COUNT( A.C1 )
     2  -  SQL : SELECT /*+ USE_GROUP_HASH(100) FULL( _A1 ) */ "_A1"."C1", MIN(
"_A1"."SHARD_KEY" ), COUNT( "_A1"."C1" ) FROM "PUBLIC"."T_SHARD_1"@LOCAL AS "_A1" GROUP BY
"_A1"."C1"
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 1 rows, G3(G3N1,G3N2) 1 rows
           RE-GROUPING
             GROUP KEY : A.C1
             AGGREGATION : MIN( MIN( A.SHARD_KEY ) ), SUM( COUNT( A.C1 ) )
             PHYSICAL FILTER : MIN( A.SHARD_KEY ) > 1
```

```
     4  -  TARGET : _A1.C1, MIN( _A1.SHARD_KEY ), COUNT( _A1.C1 )
     5  -  GROUP KEY : _A1.C1
           RECORD COLUMN : MIN( _A1.SHARD_KEY ), COUNT( _A1.C1 )
           READ KEY COLUMN : _A1.C1
           READ RECORD COLUMN : MIN( _A1.SHARD_KEY ), COUNT( _A1.C1 )
     6  -  RANGE SHARD ( # 3 )
           READ COLUMN : _A1.SHARD_KEY, _A1.C1
<<<  end print plan
```

The cluster puller plan node is determined according to **data collection method** grouped per group. Whe n the data is collected in the way of by pass, a single cluster is used. When the data is collected in the way of merge sort, multiple clusters are used.

When using the preserved order of the subordinate node of the cluster puller plan node for grouping, the data is collected in the way of merge sort .

The following is an example of grouping which uses a single cluster.

```
gSQL> \EXPLAIN PLAN SELECT c1 FROM t_shard_1 AS A GROUP BY c1;
C1
--
 1
 2
 3
3 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
| IDX | NODE DESCRIPTION                                |            ROWS |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                                |               3 |
|   1 |  QUERY BLOCK ("$QB_IDX_2")                      |               3 |
|   2 |   SINGLE CLUSTER                                | LOCAL/REMOTE  3 |
|   3 |    SELECT STATEMENT                             |               1 |
|   4 |     QUERY BLOCK ("$QB_IDX_2")                   |               1 |
|   5 |      GROUP                                      |               1 |
|   6 |       INDEX ACCESS ("T_SHARD_1" AS _A1, "IDX") | (      1)     1 |
================================================================================
     1  -  TARGET : A.C1
     2  -  SQL : SELECT /*+ INDEX( _A1, "PUBLIC"."IDX" ) */ "_A1"."C1" FROM
"PUBLIC"."T_SHARD_1"@LOCAL AS "_A1" GROUP BY "_A1"."C1"
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 1 rows, G3(G3N1,G3N2) 1 rows
```

```
          RE-GROUPING
            GROUP KEY : A.C1
     4  -  TARGET : _A1.C1
     5  -  GROUP KEY : _A1.C1
     6  -  RANGE SHARD ( # 3 )
           READ INDEX COLUMN : _A1.C1
<<<  end print plan
```

The grouping which uses a single cluster can get the grouping result after all data is collected.

The following is an example of grouping which uses multiple clusters.

```
gSQL> \EXPLAIN PLAN SELECT /*+ MERGE_GROUP */ c1 FROM t_shard_1 AS A GROUP BY c1;
C1
--
 1
 2
 3
3 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
| IDX | NODE DESCRIPTION                                 |            ROWS |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                                 |               3 |
|   1 |  QUERY BLOCK ("$QB_IDX_2")                       |               3 |
|   2 |   MULTIPLE CLUSTER                               | LOCAL/REMOTE  3 |
|   3 |    SELECT STATEMENT                              |               1 |
|   4 |     QUERY BLOCK ("$QB_IDX_2")                    |               1 |
|   5 |      GROUP                                       |               1 |
|   6 |       INDEX ACCESS ("T_SHARD_1" AS _A1, "IDX") | (      1)     1 |
================================================================================
     1  -  TARGET : A.C1
     2  -  SQL : SELECT /*+ INDEX( _A1, "PUBLIC"."IDX" ) */ "_A1"."C1" FROM
"PUBLIC"."T_SHARD_1"@LOCAL AS "_A1" GROUP BY "_A1"."C1" ORDER BY "_A1"."C1" ASC NULLS LAST
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 1 rows, G3(G3N1,G3N2) 1 rows
           MERGE GROUPING
             SORT KEY : A.C1
             GROUP KEY : A.C1
     4  -  TARGET : _A1.C1
     5  -  GROUP KEY : _A1.C1
     6  -  RANGE SHARD ( # 3 )
```

```
           READ INDEX COLUMN : _A1.C1
<<<  end print plan
```

The grouping which uses multiple clusters can get the grouping result after the data collection per grouping key is completed.

However, if all sharding keys are used as a grouping condition, then the cluster puller using no manipulation is configured. The cluster puller configures the generated query including grouping, and transfers the result to the superordinate plan without manipulating the collected data.

The following is an example of grouping with no manipulation when a user query includes GROUP BY statement.

```
gSQL> \EXPLAIN PLAN SELECT shard_key FROM t_shard_1 AS A GROUP BY shard_key;
SHARD_KEY
---------
      111
      555
      333
3 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                          |              ROWS |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                          |                 3 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")               |                 3 |
|    2  |      PLAN BASED CLUSTER                    | LOCAL/REMOTE    3 |
|    3  |        GROUP HASH INSTANT                  |                 1 |
|    4  |          TABLE ACCESS ("T_SHARD_1" AS A)   |                 1 |
================================================================================
     1  -  TARGET : A.SHARD_KEY
     2  -  SQL : SELECT /*+ USE_GROUP_HASH(10) FULL( _A1 ) */ "_A1"."SHARD_KEY" FROM
"PUBLIC"."T_SHARD_1"@LOCAL AS "_A1" GROUP BY "_A1"."SHARD_KEY"
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 1 rows, G3(G3N1,G3N2) 1 rows
     3  -  GROUP KEY : A.SHARD_KEY
           READ KEY COLUMN : A.SHARD_KEY
     4  -  RANGE SHARD ( # 3 )
           READ COLUMN : A.SHARD_KEY
<<<  end print plan
```

Generated Query for Ordering

The generated query for ordering performs ordering per each group. It orders the data collected through the generated query again, then configures the result.

The data collection for ordering is supported by the merge sort method. The merge sort method uses mul tiple clusters.

The following is an example of processing the ordering by using the preserved order in multiple clusters.

```
gSQL> \EXPLAIN PLAN SELECT c1 FROM t_shard_1 ORDER BY c1;
C1
--
 1
 2
 3
3 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
¦ IDX ¦ NODE DESCRIPTION                                ¦           ROWS ¦
--------------------------------------------------------------------------------
¦   0 ¦SELECT STATEMENT                                 ¦              3 ¦
¦   1 ¦  QUERY BLOCK ("$QB_IDX_2")                      ¦              3 ¦
¦   2 ¦    MULTIPLE CLUSTER                             ¦ LOCAL/REMOTE 3 ¦
¦   3 ¦      SELECT STATEMENT                           ¦              1 ¦
¦   4 ¦        QUERY BLOCK ("$QB_IDX_2")                ¦              1 ¦
¦   5 ¦          INDEX ACCESS ("T_SHARD_1" AS _A1, "IDX") ¦ (      1)    1 ¦
================================================================================
     1  -  TARGET : T_SHARD_1.C1
     2  -  SQL : SELECT /*+ INDEX( _A1, "PUBLIC"."IDX" ) */ "_A1"."C1" FROM
"PUBLIC"."T_SHARD_1"@LOCAL AS "_A1" ORDER BY "_A1"."C1" ASC NULLS LAST
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 1 rows, G3(G3N1,G3N2) 1 rows
           MERGE SORTING
             SORT KEY : T_SHARD_1.C1
     4  -  TARGET : _A1.C1
     5  -  RANGE SHARD ( # 3 )
           READ INDEX COLUMN : _A1.C1
<<<  end print plan
```

The following is an example of processing the ordering when the preserved order does not exist in multip le clusters.

```
gSQL> \EXPLAIN PLAN SELECT /*+ FULL( t_shard_1 ) */ c1 FROM t_shard_1 ORDER BY c1;
C1
--
 1
 2
 3
3 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
¦  IDX  ¦  NODE DESCRIPTION                              ¦            ROWS ¦
--------------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                              ¦               3 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                  ¦               3 ¦
¦    2  ¦      MULTIPLE CLUSTER                         ¦ LOCAL/REMOTE  3 ¦
¦    3  ¦        SELECT STATEMENT                       ¦               1 ¦
¦    4  ¦          QUERY BLOCK ("$QB_IDX_2")            ¦               1 ¦
¦    5  ¦            SORT INSTANT                       ¦               1 ¦
¦    6  ¦              TABLE ACCESS ("T_SHARD_1" AS _A1) ¦               1 ¦
================================================================================
     1  -  TARGET : T_SHARD_1.C1
     2  -  SQL : SELECT /*+ USE_ORDER_SORT FULL( _A1 ) */ "_A1"."C1" FROM
"PUBLIC"."T_SHARD_1"@LOCAL AS "_A1" ORDER BY "_A1"."C1" ASC NULLS LAST
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 1 rows, G3(G3N1,G3N2) 1 rows
           MERGE SORTING
             SORT KEY : T_SHARD_1.C1
     4  -  TARGET : _A1.C1
     5  -  SORT KEY : "_A1.C1 ASC NULLS LAST"
           READ KEY COLUMN : _A1.C1
     6  -  RANGE SHARD ( # 3 )
           READ COLUMN : _A1.C1
<<<  end print plan
```

Generated Query for Intersect Key Group

The intersect key group evaluation determines whether it received the same data from all groups.

It is not applied to all collected data. The intersect is applied only when key group values are same and all of nil expression values are null.

If nil expression value is not null, then the result is configured without evaluating the intersect key group.

If all nil expression values are null, then it configures the intersect key group result only when it received t

he same data from all groups.

The generated query for the intersect key group is ordered in an order of the key group. The data collecte d through the generated query is ordered again in an order of the key group. The intersect key group is a pplied to the sorted data according to the nil expression value.

The following is an example of processing the intersect key group in multiple clusters.

```
gSQL> \EXPLAIN PLAN
      SELECT /*+ REMOTE_JOIN( t_clone_1 ) */ t_clone_1.c1, t_shard_1.c1
        FROM t_clone_1
             LEFT OUTER JOIN
             t_shard_1
             ON t_clone_1.c1 = t_shard_1.c1;
C1   C1
-- ----
 1    1
 3    3
 5 null
3 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
¦ IDX ¦ NODE DESCRIPTION                                ¦            ROWS ¦
--------------------------------------------------------------------------------
¦   0 ¦ SELECT STATEMENT                                ¦               3 ¦
¦   1 ¦  QUERY BLOCK ("$QB_IDX_2")                      ¦               3 ¦
¦   2 ¦   MULTIPLE CLUSTER                              ¦ LOCAL/REMOTE  3 ¦
¦   3 ¦    SELECT STATEMENT                             ¦               3 ¦
¦   4 ¦     QUERY BLOCK ("$QB_IDX_2")                   ¦               3 ¦
¦   5 ¦      SORT INSTANT                               ¦               3 ¦
¦   6 ¦       NESTED JOIN (LEFT OUTER JOIN)             ¦               3 ¦
¦   7 ¦        TABLE ACCESS ("T_CLONE_1" AS _A2)        ¦               3 ¦
¦   8 ¦        INDEX ACCESS ("T_SHARD_1" AS _A1, "IDX") ¦ (    1)      1 ¦
================================================================================
     1  -  TARGET : T_CLONE_1.C1, T_SHARD_1.C1
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE USE_NL_IN( _A1 ) FULL( _A2 ) INDEX( _A1,
"PUBLIC"."IDX" ) */ "_A2"."C1", "_A1"."C1" FROM (
"PUBLIC"."T_CLONE_1"@"G1N1"¦"G1N2"¦"G2N1"¦"G2N2"¦"G3N1"¦"G3N2" AS "_A2" LEFT OUTER JOIN
"PUBLIC"."T_SHARD_1"@"G1N1"¦"G1N2"¦"G2N1"¦"G2N2"¦"G3N1"¦"G3N2" AS "_A1" ON "_A1"."C1" =
"_A2"."C1") ALIAS "_A3" ORDER BY "_A2"."C1" ASC NULLS LAST
           TARGET DOMAIN : G1(G1N1,G1N2) 3 rows, G2(G2N1,G2N2) 3 rows, G3(G3N1,G3N2) 3 rows
```

```
            INTERSECT KEY GROUP
              KEY GROUP : T_CLONE_1.C1
              Nil Expression : T_SHARD_1.C1
     4  -  TARGET : _A2.C1, _A1.C1
     5  -  SORT KEY : "_A2.C1 ASC NULLS LAST"
           RECORD COLUMN : _A1.C1
           READ KEY COLUMN : _A2.C1
           READ RECORD COLUMN : _A1.C1
     6  -  JOINED COLUMN : _A2.C1, _A1.C1
     7  -  CLONED
           READ COLUMN : _A2.C1
     8  -  RANGE SHARD ( # 3 )
           READ INDEX COLUMN : _A1.C1
             MIN RANGE : _A1.C1 = {_A2.C1}
             MAX RANGE : _A1.C1 = {_A2.C1}
<<<  end print plan
```

Generated Query for Distinct Key Group

The distinct key group evaluation determines whether it received the same data from two or more groups.

Distinct is applied when the key group values are same and the data was received from each group. Disti nct is not applied to the data from the same group.

The generated query for the distinct key group is ordered in an order of the key group. The data collected through the generated query is ordered again in an order of the key group. The distinct key group is appli ed to the sorted data according to the received group.

The following is an example of processing the distinct key group in multiple clusters.

```
gSQL> \EXPLAIN PLAN
      SELECT t_clone_1.c1
        FROM t_clone_1
       WHERE t_clone_1.c1 IN ( SELECT /*+ REMOTE_UNNEST */ t_shard_1.c1 FROM t_shard_1 );
C1
--
 1
 3
2 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
¦ IDX ¦  NODE DESCRIPTION                                ¦            ROWS ¦
```

```
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                                |               2 |
|   1 |  QUERY BLOCK ("$QB_IDX_2")                      |               2 |
|   2 |   MULTIPLE CLUSTER                              | LOCAL/REMOTE  2 |
|   3 |    SELECT STATEMENT                             |               1 |
|   4 |     QUERY BLOCK ("$QB_IDX_2")                   |               1 |
|   5 |      SORT INSTANT                               |               1 |
|   6 |       HASH JOIN (SEMI)                          |               1 |
|   7 |        TABLE ACCESS ("T_CLONE_1" AS _A2)        |               3 |
|   8 |        HASH JOIN INSTANT (UNIQUE)               |               1 |
|   9 |         INDEX ACCESS ("T_SHARD_1" AS _A1, "IDX") | (     1)      1 |
===============================================================================
     1  -  TARGET : T_CLONE_1.C1
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE USE_HASH_IN( _A1, 100 ) FULL( _A2 ) INDEX( _A1,
"PUBLIC"."IDX" ) */ "_A2"."C1" FROM (
"PUBLIC"."T_CLONE_1"@"G1N1"¦"G1N2"¦"G2N1"¦"G2N2"¦"G3N1"¦"G3N2" AS "_A2" SEMI JOIN
"PUBLIC"."T_SHARD_1"@"G1N1"¦"G1N2"¦"G2N1"¦"G2N2"¦"G3N1"¦"G3N2" AS "_A1" ON "_A1"."C1" =
"_A2"."C1") ALIAS "_A3" ORDER BY "_A2"."C1" ASC NULLS LAST
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 0 rows, G3(G3N1,G3N2) 1 rows
           DISTINCT KEY GROUP
             KEY GROUP : T_CLONE_1.C1
     4  -  TARGET : _A2.C1
     5  -  SORT KEY : "_A2.C1 ASC NULLS LAST"
           READ KEY COLUMN : _A2.C1
     6  -  JOINED COLUMN : _A2.C1
     7  -  CLONED
           READ COLUMN : _A2.C1
     8  -  HASH KEY : _A1.C1
           READ KEY COLUMN : _A1.C1
             HASH FILTER : _A1.C1 = _A2.C1
           FETCH ONE ROW
     9  -  RANGE SHARD ( # 3 )
           READ INDEX COLUMN : _A1.C1
<<<  end print plan
```

Constraints of Generated Query Configuration

The generated query configuration is constrained in the following cases.

- **Offset & limit**
- **Using non-deterministic expression**

- **When unable to unnest subquery**

Offset & limit

The information about **offset limit clause** is not included in a generated query.

- The generated query does not include the offset information.

```
gSQL> \EXPLAIN PLAN ONLY SELECT c1 FROM t_shard_1 OFFSET 1;
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                ¦        ROWS ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                ¦           0 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                   ¦           0 ¦
¦    2  ¦      PLAN BASED CLUSTER                        ¦           0 ¦
¦    3  ¦        INDEX ACCESS ("T_SHARD_1", "IDX")       ¦           0 ¦
=============================================================================
     1  -  TARGET : T_SHARD_1.C1
     2  -  SQL : SELECT /*+ INDEX( _A1, "PUBLIC"."IDX" ) */ "_A1"."C1" FROM
"PUBLIC"."T_SHARD_1"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 0 rows, G2(G2N1,G2N2) 0 rows, G3(G3N1,G3N2) 0 rows
     3  -  RANGE SHARD ( # 3 )
           READ INDEX COLUMN : T_SHARD_1.C1
<<<  end print plan
```

- The generated query does not include the limit information.

```
gSQL> \EXPLAIN PLAN ONLY SELECT c1 FROM t_shard_1 LIMIT 1;
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                ¦        ROWS ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                ¦           0 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                   ¦           0 ¦
¦    2  ¦      PLAN BASED CLUSTER                        ¦           0 ¦
¦    3  ¦        INDEX ACCESS ("T_SHARD_1", "IDX")       ¦           0 ¦
=============================================================================
     1  -  TARGET : T_SHARD_1.C1
```

```
     2  -  SQL : SELECT /*+ INDEX( _A1, "PUBLIC"."IDX" ) */ "_A1"."C1" FROM
"PUBLIC"."T_SHARD_1"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 0 rows, G2(G2N1,G2N2) 0 rows, G3(G3N1,G3N2) 0 rows
     3  -  RANGE SHARD ( # 3 )
           READ INDEX COLUMN : T_SHARD_1.C1
<<<  end print plan
```

Using non-deterministic expression

When using the non-deterministic expression, if non-deterministic expression can not be made into a constant, then it is not included in the generated query.

- The generated query does not include a sequence related expression.

```
gSQL> \EXPLAIN PLAN ONLY SELECT seq.nextval FROM t_shard_1;
>>>  start print plan
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                              |         ROWS |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                              |            0 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                   |            0 |
|    2  |      PLAN BASED CLUSTER                        |            0 |
|    3  |        INDEX ACCESS ("T_SHARD_1", "IDX")       |            0 |
=============================================================================
     1  -  TARGET : NEXTVAL(SEQ)
     2  -  SQL : SELECT /*+ INDEX( _A1, "PUBLIC"."IDX" ) */ NULL FROM
"PUBLIC"."T_SHARD_1"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 0 rows, G2(G2N1,G2N2) 0 rows, G3(G3N1,G3N2) 0 rows
     3  -  RANGE SHARD ( # 3 )
           READ INDEX COLUMN : NOTHING
<<<  end print plan
```

- The non-deterministic expression which can be made into a constant is made into a constant by the subordinate cluster puller node.
- The generated query includes the non-deterministic expression in a bind parameter form.

```
gSQL> \EXPLAIN PLAN ONLY SELECT c1 FROM t_shard_1 WHERE shard_key = random( 1, 100 );
>>>  start print plan
< Execution Plan >
=============================================================================
```

```
┆  IDX  ┆  NODE DESCRIPTION                       ┆              ROWS ┆
-------------------------------------------------------------------------
┆    0  ┆  SELECT STATEMENT                       ┆                 0 ┆
┆    1  ┆    QUERY BLOCK ("$QB_IDX_2")            ┆                 0 ┆
┆    2  ┆      PLAN BASED CLUSTER                 ┆                 0 ┆
┆    3  ┆        TABLE ACCESS ("T_SHARD_1")       ┆                 0 ┆
=========================================================================
     1  -  TARGET : T_SHARD_1.C1
     2  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."C1" FROM "PUBLIC"."T_SHARD_1"@LOCAL AS "_A1"
WHERE "_A1"."SHARD_KEY" = :_V0
           TARGET DOMAIN : G1(G1N1,G1N2) 0 rows, G2(G2N1,G2N2) 0 rows, G3(G3N1,G3N2) 0 rows
     3  -  RANGE SHARD ( # 3 )
           READ COLUMN : T_SHARD_1.SHARD_KEY, T_SHARD_1.C1
             PHYSICAL FILTER : T_SHARD_1.SHARD_KEY = RANDOM(1,100)
<<<  end print plan
```

- The non-deterministic expression which can not be made into a constant is not included in the gener ated query.
- The non-deterministic expression is processed after the data is collected on the cluster puller node.

```
gSQL> \EXPLAIN PLAN ONLY
SELECT 1 FROM t_shard_1 WHERE shard_key = random( c1, 100 );
>>>  start print plan
< Execution Plan >
=========================================================================================================
====
┆  IDX  ┆  NODE DESCRIPTION                                                    ┆
ROWS ┆
---------------------------------------------------------------------------------------------------------
----
┆    0  ┆  SELECT STATEMENT                                                    ┆
┆
┆    1  ┆    FILTER                                                            ┆
0 ┆
┆    2  ┆      CLUSTER ACCESS ("T_SHARD_1") [RANGE SHARDING]                   ┆
0 ┆
┆    3  ┆        TABLE ACCESS ("T_SHARD_1") [RANGE SHARDING]                   ┆
0 ┆
=========================================================================================================
====
     1  -  READ COLUMNS : NOTHING
```

```
           FILTER : T_SHARD_1.SHARD_KEY = RANDOM(T_SHARD_1.C1,100)
     2  -  SQL : SELECT /*+ FULL("_A1") */ "_A1"."SHARD_KEY","_A1"."C1" FROM
"PUBLIC"."T_SHARD_1"@LOCAL "_A1"
     3  -  READ COLUMNS : SHARD_KEY, C1
<<<  end print plan
```

When unable to unnest subquery

A subquery which can be made into a constant is included in the generated query in a bind para meter form.
A subquery which can not be made into a constant is not included in the generated query.

For more information about unnesting a subquery, refer to **Subquery**.

- The generated query can not include the unnested subquery.
- The subquery is processed on the cluster puller node or the superordinate node.

```
gSQL> \EXPLAIN PLAN
      SELECT shard_key
        FROM t_shard_1
       WHERE t_shard_1.c1 IN ( SELECT /*+ NO_QUERY_TRANSFORMATION */ t_clone_1.c1 FROM
t_clone_1 );
SHARD_KEY
---------
      111
      555
2 rows selected.
>>>  start print plan
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                      |                ROWS |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                      |                   2 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")           |                   2 |
|    2  |      PLAN BASED CLUSTER                | LOCAL/REMOTE      2 |
|    3  |        TABLE ACCESS ("T_SHARD_1")      |                   1 |
|    4  |  SUB QUERY LIST                        |                     |
|    5  |    INLINE_VIEW ("$V5") (MATERIALIZED)  |                   2 |
|    6  |      QUERY BLOCK ("$QB_IDX_6")         |                   3 |
|    7  |        TABLE ACCESS ("T_CLONE_1")      |                   3 |
=============================================================================
```

```
     1  -  TARGET : T_SHARD_1.SHARD_KEY
     2  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."SHARD_KEY", "_A1"."C1" FROM
"PUBLIC"."T_SHARD_1"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 1 rows, G3(G3N1,G3N2) 1 rows
             POST FILTER : ( T_SHARD_1.C1 ) IN ( $V5.C1 )
     3  -  RANGE SHARD ( # 3 )
           READ COLUMN : T_SHARD_1.SHARD_KEY, T_SHARD_1.C1
     5  -  COLUMN : T_CLONE_1.C1 AS C1
     6  -  TARGET : T_CLONE_1.C1
     7  -  CLONED
           READ COLUMN : T_CLONE_1.C1
<<<  end print plan
```

- The generated query can not include a subquery which can not be made into a constant.

```
gSQL> \EXPLAIN PLAN
      SELECT shard_key
        FROM t_shard_1
       WHERE shard_key = ( SELECT c1 FROM dual );
no rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                          |               ROWS |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                          |                  0 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")               |                  0 |
|    2  |      PLAN BASED CLUSTER                    | LOCAL/REMOTE     0 |
|    3  |        TABLE ACCESS ("T_SHARD_1")          |                  1 |
|    4  |      SUB QUERY LIST                        |                    |
|    5  |        INLINE_VIEW ("$V5")                 |                  3 |
|    6  |          QUERY BLOCK ("$QB_IDX_6")         |                  3 |
|    7  |            FAST DUAL ACCESS ("DUAL")       |                  3 |
================================================================================
     1  -  TARGET : T_SHARD_1.SHARD_KEY
     2  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."SHARD_KEY", "_A1"."C1" FROM
"PUBLIC"."T_SHARD_1"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 1 rows, G3(G3N1,G3N2) 1 rows
             POST FILTER : T_SHARD_1.SHARD_KEY = $V5.C1
     3  -  RANGE SHARD ( # 3 )
           READ COLUMN : T_SHARD_1.SHARD_KEY, T_SHARD_1.C1
```

```
     5  -  COLUMN : {T_SHARD_1.C1} AS C1
     6  -  TARGET : {T_SHARD_1.C1}
     7  -  READ COLUMN : NOTHING
<<<  end print plan
```

- The generated query includes a subquery which can be made into a constant in a bind parameter for m.

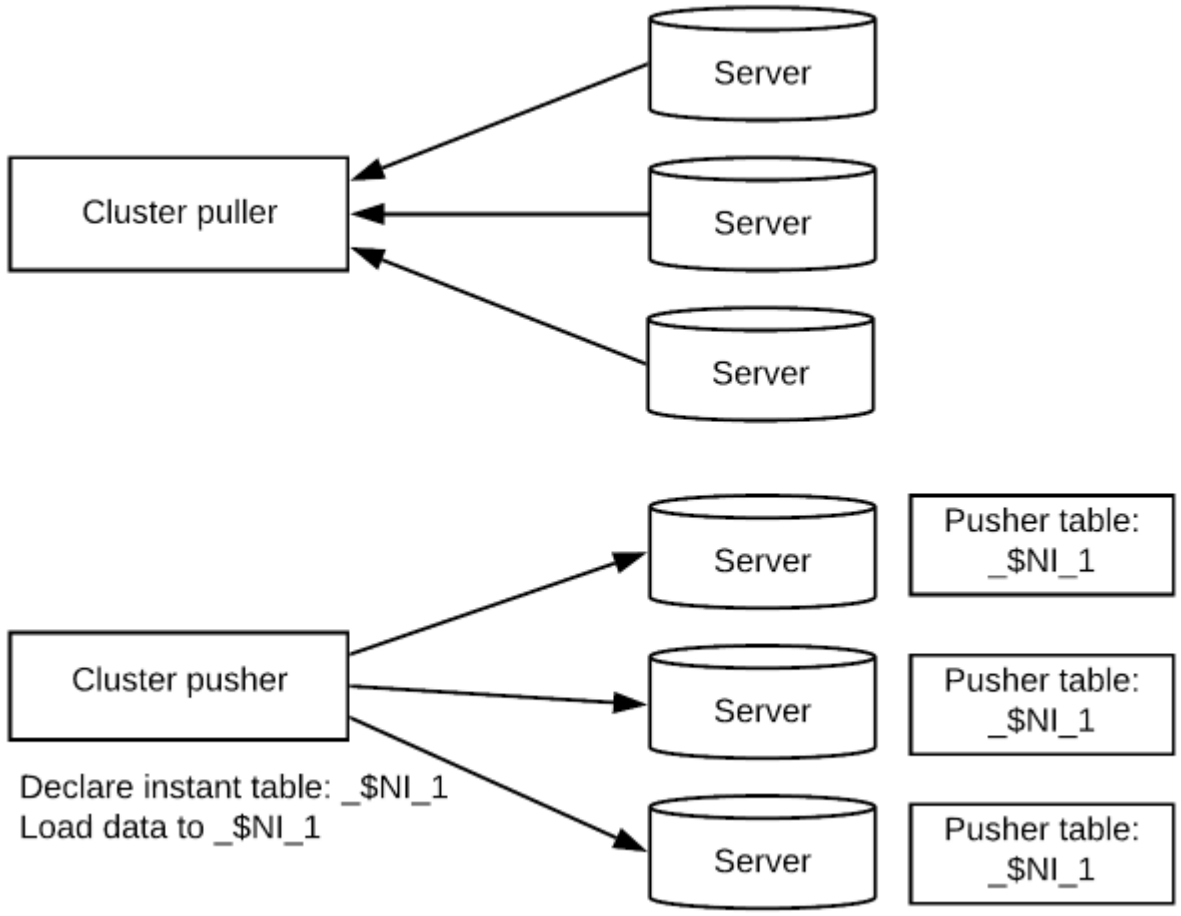
```
gSQL> \EXPLAIN PLAN
SELECT shard_key
  FROM t_shard_1
 WHERE shard_key = ( SELECT 111 FROM dual );
SHARD_KEY
---------
      111
1 row selected.
>>>  start print plan
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                      |               ROWS |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                      |                  1 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")           |                  1 |
|    2  |      PLAN BASED CLUSTER                | LOCAL ONLY       1 |
|    3  |        TABLE ACCESS ("T_SHARD_1")      |                  1 |
|    4  |  SUB QUERY LIST                        |                    |
|    5  |    INLINE_VIEW ("$V5")                 |                  1 |
|    6  |      QUERY BLOCK ("$QB_IDX_6")         |                  1 |
|    7  |        FAST DUAL ACCESS ("DUAL")       |                  1 |
=============================================================================
     1  -  TARGET : T_SHARD_1.SHARD_KEY
     2  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."SHARD_KEY" FROM "PUBLIC"."T_SHARD_1"@LOCAL
AS "_A1" WHERE "_A1"."SHARD_KEY" = :_V0
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 0 rows, G3(G3N1,G3N2) 0 rows
             SHARD KEY FILTER : ( T_SHARD_1.SHARD_KEY = $V5.$C0 )
     3  -  RANGE SHARD ( # 3 )
           READ COLUMN : T_SHARD_1.SHARD_KEY
             PHYSICAL FILTER : T_SHARD_1.SHARD_KEY = $V5.$C0
     5  -  COLUMN : 111 AS $C0
     6  -  TARGET : 111
     7  -  READ COLUMN : NOTHING
```

```
<<<  end print plan
```

Cluster Pusher

The cluster pusher node creates and manages a virtual table for the efficient query process of the cluster puller node.

Figure 7 Cluster puller vs cluster pusher

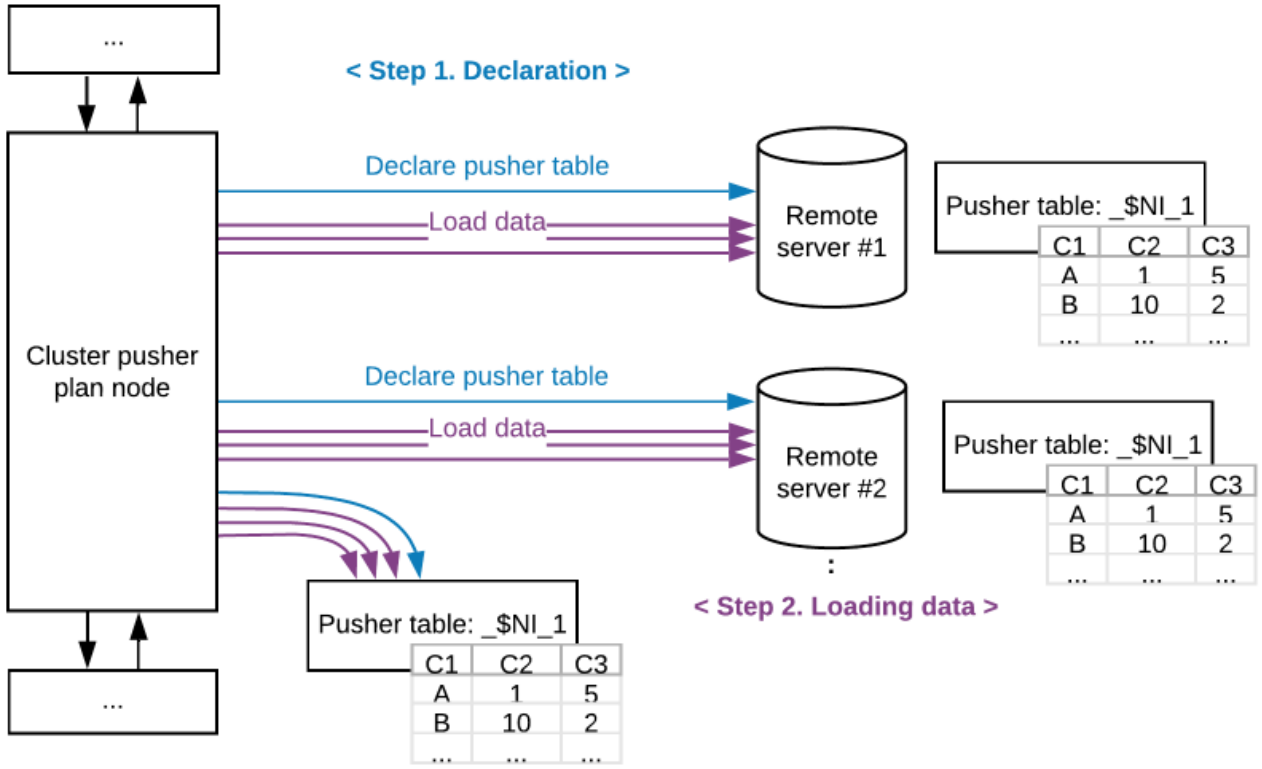


The cluster puller node collects the data. The cluster pusher node distributes the data in a new table form.

The cluster pusher node declares a pusher table and loads the data.

- Declaring pusher table: It creates an instant table in a local server and a remote server.
- Loading data: It loads the data received from the subordinate node of the cluster pusher node on the pusher table.

Figure 8 Performing cluster pusher



The following is an example of using the cluster pusher.

```
gSQL> \EXPLAIN PLAN ONLY
      SELECT A.c1
        FROM t_shard_1 AS A, t_shard_1 AS B
       WHERE A.shard_key = B.c1;
>>>  start print plan
< Execution Plan >
================================================================================
¦  IDX  ¦  NODE DESCRIPTION                                    ¦      ROWS ¦
--------------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                    ¦         0 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                       ¦         0 ¦
¦    2  ¦      SINGLE CLUSTER                                ¦         0 ¦
¦    3  ¦        CLUSTER PUSHER ("_$NI_7")                   ¦         0 ¦
¦    4  ¦          PLAN BASED CLUSTER                        ¦         0 ¦
¦    5  ¦            INDEX ACCESS ("T_SHARD_1" AS B, "IDX")  ¦         0 ¦
¦    6  ¦        HASH JOIN (INNER JOIN)                      ¦         0 ¦
¦    7  ¦          TABLE ACCESS ("T_SHARD_1" AS A)           ¦         0 ¦
¦    8  ¦          HASH JOIN INSTANT                         ¦         0 ¦
¦    9  ¦            PUSHER TABLE ACCESS ("_$NI_7")          ¦         0 ¦
```

```
================================================================================
     1  -  TARGET : A.C1
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE USE_HASH_IN( _A1, 100 ) FULL( _A2 ) FULL( _A1 )
*/ "_A2"."C1" FROM ( "PUBLIC"."T_SHARD_1"@LOCAL AS "_A2" INNER JOIN
"SESSION_SCHEMA"."_$NI_7"@LOCAL AS "_A1" ON "_A1"."C1" = "_A2"."SHARD_KEY") ALIAS "_A3"
           TARGET DOMAIN : G1(G1N1,G1N2) 0 rows, G2(G2N1,G2N2) 0 rows, G3(G3N1,G3N2) 0 rows
     3  -  SQL : DECLARE INSTANT TABLE "SESSION_SCHEMA"."_$NI_7" ( "C1" NUMBER(10, 0) )
           COLUMN : B.C1 AS C1
           SHARDED : B.C1
           TARGET DOMAIN : G1(G1N1,G1N2) 0 rows, G2(G2N1,G2N2) 0 rows, G3(G3N1,G3N2) 0 rows
     4  -  SQL : SELECT /*+ INDEX( _A1, "PUBLIC"."IDX" ) */ "_A1"."C1" FROM
"PUBLIC"."T_SHARD_1"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 0 rows, G2(G2N1,G2N2) 0 rows, G3(G3N1,G3N2) 0 rows
     5  -  RANGE SHARD ( # 3 )
           READ INDEX COLUMN : B.C1
     6  -  JOINED COLUMN : A.C1
     7  -  RANGE SHARD ( # 3 )
           READ COLUMN : A.SHARD_KEY, A.C1
     8  -  HASH KEY : _$NI_7.C1
           READ KEY COLUMN : _$NI_7.C1
             HASH FILTER : _$NI_7.C1 = A.SHARD_KEY
<<<  end print plan
```

In the execution plan above, the plan whose idx of <Execution Plan> is 3 is the cluster pusher.

The pusher table name is output on NODE DESCRIPTION of the cluster pusher.

The detailed information about CLUSTER PUSHER is as follows.

- SQL: It is a query declaring a pusher table.
- COLUMN: It is an original expression per each column of the pusher table.
- SHARDED: They are columns of a pusher table used based on the pusher table data distribution.
- TARGET DOMAIN: It is the member and group to configure the pusher table, and the number of dat a transferred that group.

Pusher Table

The pusher table is an instant table in user query unit which is configured by the query processor to effici ently use the cluster puller. It is used to build the collected data in the driver server in a relation form.

The configured pusher table can be referred from the generated query like as a general table.

The following is an example of performing the join query which does not use the pusher table.

```
gSQL> \EXPLAIN PLAN
      SELECT /*+ LOCAL_JOIN( t_shard_1 ) */ t_shard_1.c1
        FROM t_shard_1, t_shard_2
       WHERE t_shard_1.shard_key = t_shard_2.shard_key;
C1
--
 1
 3
 2
3 rows selected.
>>>  start print plan
< Execution Plan >
========================================================================
¦  IDX  ¦  NODE DESCRIPTION                      ¦               ROWS ¦
------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                      ¦                  3 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")          ¦                  3 ¦
¦    2  ¦      HASH JOIN (INNER JOIN)           ¦                  3 ¦
¦    3  ¦        PLAN BASED CLUSTER             ¦ LOCAL/REMOTE     3 ¦
¦    4  ¦          TABLE ACCESS ("T_SHARD_1")   ¦                  1 ¦
¦    5  ¦        HASH JOIN INSTANT              ¦                  3 ¦
¦    6  ¦          PLAN BASED CLUSTER           ¦ LOCAL/REMOTE     3 ¦
¦    7  ¦            TABLE ACCESS ("T_SHARD_2") ¦                  1 ¦
========================================================================
     1  -  TARGET : T_SHARD_1.C1
     2  -  JOINED COLUMN : T_SHARD_1.C1
     3  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."SHARD_KEY", "_A1"."C1" FROM
"PUBLIC"."T_SHARD_1"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 1 rows, G3(G3N1,G3N2) 1 rows
     4  -  RANGE SHARD ( # 3 )
           READ COLUMN : T_SHARD_1.SHARD_KEY, T_SHARD_1.C1
     5  -  HASH KEY : T_SHARD_2.SHARD_KEY
           READ KEY COLUMN : T_SHARD_2.SHARD_KEY
             HASH FILTER : T_SHARD_2.SHARD_KEY = T_SHARD_1.SHARD_KEY
     6  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."SHARD_KEY" FROM "PUBLIC"."T_SHARD_2"@LOCAL
AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G3(G3N1,G3N2) 2 rows
     7  -  RANGE SHARD ( # 4 )
           READ COLUMN : T_SHARD_2.SHARD_KEY
<<<  end print plan
```

Two cluster puller plans are used when performing the join as given above. It is because a single generate d query can not process the join for two tables with each different sharding strategy.

The following is an example of performing the join query by using the pusher table.

```
gSQL> \EXPLAIN PLAN
SELECT /*+ REMOTE_JOIN( t_shard_1 ) */ t_shard_1.c1
  FROM t_shard_1, t_shard_2
 WHERE t_shard_1.shard_key = t_shard_2.shard_key;
C1
--
 1
 2
 3
3 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
| IDX | NODE DESCRIPTION                                |            ROWS |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                                |               3 |
|   1 |  QUERY BLOCK ("$QB_IDX_2")                      |               3 |
|   2 |   SINGLE CLUSTER                                | LOCAL/REMOTE  3 |
|   3 |    CLUSTER PUSHER ("_$NI_7")                    |               3 |
|   4 |     PLAN BASED CLUSTER                          | LOCAL/REMOTE  3 |
|   5 |      TABLE ACCESS ("T_SHARD_2")                 |               1 |
|   6 |    SELECT STATEMENT                             |               1 |
|   7 |     QUERY BLOCK ("$QB_IDX_2")                   |               1 |
|   8 |      HASH JOIN (INNER JOIN)                     |               1 |
|   9 |       TABLE ACCESS ("T_SHARD_1" AS _A2)         |               1 |
|  10 |       HASH JOIN INSTANT                         |               1 |
|  11 |        PUSHER TABLE ACCESS ("_$NI_7" AS _A1)    |               1 |
================================================================================
     1  -  TARGET : T_SHARD_1.C1
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE USE_HASH_IN( _A1, 10 ) FULL( _A2 ) FULL( _A1 )
*/ "_A2"."C1" FROM ( "PUBLIC"."T_SHARD_1"@LOCAL AS "_A2" INNER JOIN
"SESSION_SCHEMA"."_$NI_7"@LOCAL AS "_A1" ON "_A1"."SHARD_KEY" = "_A2"."SHARD_KEY") ALIAS "_A3"
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 1 rows, G3(G3N1,G3N2) 1 rows
     3  -  SQL : DECLARE INSTANT TABLE "SESSION_SCHEMA"."_$NI_7" ( "SHARD_KEY" NUMBER(10, 0) )
           COLUMN : T_SHARD_2.SHARD_KEY AS SHARD_KEY
           SHARDED : T_SHARD_2.SHARD_KEY
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 1 rows, G3(G3N1,G3N2) 1 rows
```

```
     4  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."SHARD_KEY" FROM "PUBLIC"."T_SHARD_2"@LOCAL
AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G3(G3N1,G3N2) 2 rows
     5  -  RANGE SHARD ( # 4 )
           READ COLUMN : T_SHARD_2.SHARD_KEY
     7  -  TARGET : _A2.C1
     8  -  JOINED COLUMN : _A2.C1
     9  -  RANGE SHARD ( # 3 )
           READ COLUMN : _A2.SHARD_KEY, _A2.C1
    10  -  HASH KEY : _A1.SHARD_KEY
           READ KEY COLUMN : _A1.SHARD_KEY
             HASH FILTER : _A1.SHARD_KEY = _A2.SHARD_KEY
    11  -  READ COLUMN : _A1.SHARD_KEY
<<<  end print plan
```

The generated query of the cluster puller configured in the execution result above includes the join.

CLUSTER PUSHER which is configured under SINGLE CLUSTER defines the pusher table "_$NI_7" through DECLARE statement.

The data collected from table t_shard_2 in CLUSTER PUSHER are distributed to the pusher table in a local server and a remote server by using the sharding strategy of t_shard_1. The join for those two tables can be processed by expressing them as a single generated query, because t_shard_1 and pusher table use th e same sharding strategy.

The pusher table has following features.

- It declares the pusher table on the cluster pusher plan node, and loads the data.
- It is configured as a table included in SESSION_SCHEMA.
- It can not configure an index.
- Multiple pusher tables can be configured in a single user query.
- Pusher tables are not shared among user queries.
- The performance cycle of the user query and the management cycle of the pusher table are same.
- The pusher table configured in each server has either a replicated data or a distributed data.

Pusher Table Configured with Replicated Data

The data configuration of the pusher table varies upon performing the cluster puller.

The pusher table configured to process the outer join as given below processes the generated query inclu ding the outer join with all servers having the same data for t_shard_2.

```
gSQL> \EXPLAIN PLAN ONLY
      SELECT A.c1
```

```
        FROM t_shard_1 AS A LEFT OUTER JOIN t_shard_2 AS B ON A.c1 = B.c1;
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                                    |      ROWS |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                    |         0 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                         |         0 |
|    2  |      SINGLE CLUSTER                                  |         0 |
|    3  |        CLUSTER PUSHER ("_$NI_5")                     |         0 |
|    4  |          PLAN BASED CLUSTER                          |         0 |
|    5  |            TABLE ACCESS ("T_SHARD_2" AS B)           |         0 |
|    6  |        HASH JOIN (INVERTED LEFT OUTER JOIN)          |         0 |
|    7  |          PUSHER TABLE ACCESS ("_$NI_5")              |         0 |
|    8  |          HASH JOIN INSTANT                           |         0 |
|    9  |            INDEX ACCESS ("T_SHARD_1" AS A, "IDX")    |         0 |
================================================================================
     1  -  TARGET : A.C1
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE USE_HASH_IN( _A1, 100 ) FULL( _A2 ) INDEX( _A1,
"PUBLIC"."IDX" ) */ "_A1"."C1" FROM (
"PUBLIC"."T_SHARD_1"@"G1N1"¦"G1N2"¦"G2N1"¦"G2N2"¦"G3N1"¦"G3N2" AS "_A1" LEFT OUTER JOIN
"SESSION_SCHEMA"."_$NI_5"@LOCAL AS "_A2" ON "_A1"."C1" = "_A2"."C1") ALIAS "_A3"
           TARGET DOMAIN : G1(G1N1,G1N2) 0 rows, G2(G2N1,G2N2) 0 rows, G3(G3N1,G3N2) 0 rows
     3  -  SQL : DECLARE INSTANT TABLE "SESSION_SCHEMA"."_$NI_5" ( "C1" NUMBER(10, 0) )
           COLUMN : B.C1 AS C1
           CLONED
           TARGET DOMAIN : G1(G1N1,G1N2) 0 rows, G2(G2N1,G2N2) 0 rows, G3(G3N1,G3N2) 0 rows
     4  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."C1" FROM "PUBLIC"."T_SHARD_2"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 0 rows, G3(G3N1,G3N2) 0 rows
     5  -  RANGE SHARD ( # 4 )
           READ COLUMN : B.C1
     6  -  JOINED COLUMN : A.C1
     8  -  HASH KEY : A.C1
           READ KEY COLUMN : A.C1
             HASH FILTER : A.C1 = _$NI_5.C1
     9  -  RANGE SHARD ( # 3 )
           READ INDEX COLUMN : A.C1
<<<  end print plan
```

The pusher table configured with the replicated data is output as CLONED as the result above.

Pusher Table Configured with Distributed Data

The following is an example of processing outer join by using the sharding key of t_shard_1 as the join co ndition.

```
gSQL> \EXPLAIN PLAN ONLY
      SELECT A.c1
        FROM t_shard_1 AS A LEFT OUTER JOIN t_shard_2 AS B ON A.shard_key = B.c1;
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                              ¦          ROWS ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                              ¦             0 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                  ¦             0 ¦
¦    2  ¦      SINGLE CLUSTER                           ¦             0 ¦
¦    3  ¦        CLUSTER PUSHER ("_$NI_7")              ¦             0 ¦
¦    4  ¦          PLAN BASED CLUSTER                   ¦             0 ¦
¦    5  ¦            TABLE ACCESS ("T_SHARD_2" AS B)    ¦             0 ¦
¦    6  ¦        HASH JOIN (LEFT OUTER JOIN)            ¦             0 ¦
¦    7  ¦          TABLE ACCESS ("T_SHARD_1" AS A)      ¦             0 ¦
¦    8  ¦          HASH JOIN INSTANT                    ¦             0 ¦
¦    9  ¦            PUSHER TABLE ACCESS ("_$NI_7")     ¦             0 ¦
=============================================================================
     1  -  TARGET : A.C1
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE USE_HASH_IN( _A1, 10 ) FULL( _A2 ) FULL( _A1 )
*/ "_A2"."C1" FROM ( "PUBLIC"."T_SHARD_1"@"G1N1"¦"G1N2"¦"G2N1"¦"G2N2"¦"G3N1"¦"G3N2" AS "_A2"
LEFT OUTER JOIN "SESSION_SCHEMA"."_$NI_7"@LOCAL AS "_A1" ON "_A1"."C1" = "_A2"."SHARD_KEY")
ALIAS "_A3"
           TARGET DOMAIN : G1(G1N1,G1N2) 0 rows, G2(G2N1,G2N2) 0 rows, G3(G3N1,G3N2) 0 rows
     3  -  SQL : DECLARE INSTANT TABLE "SESSION_SCHEMA"."_$NI_7" ( "C1" NUMBER(10, 0) )
           COLUMN : B.C1 AS C1
           SHARDED : B.C1
           TARGET DOMAIN : G1(G1N1,G1N2) 0 rows, G2(G2N1,G2N2) 0 rows, G3(G3N1,G3N2) 0 rows
     4  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."C1" FROM "PUBLIC"."T_SHARD_2"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 0 rows, G3(G3N1,G3N2) 0 rows
     5  -  RANGE SHARD ( # 4 )
           READ COLUMN : B.C1
     6  -  JOINED COLUMN : A.C1
     7  -  RANGE SHARD ( # 3 )
           READ COLUMN : A.SHARD_KEY, A.C1
     8  -  HASH KEY : _$NI_7.C1
```

```
           READ KEY COLUMN : _$NI_7.C1
             HASH FILTER : _$NI_7.C1 = A.SHARD_KEY
<<<  end print plan
```

The pusher table divides the data for t_shard_2 per a group. The data is distributed based on C1 column of t_shard_2 according to the sharding strategy configured in the sharding key of t_shard_1.

The pusher table which consists of the distributed data is output SHARDED as the detailed result above.

Processing Cluster Query per SELECT Statement

The chapters above describes the **Cluster Puller** and **Cluster Pusher** plan node to process SELECT in cluster. This chapter describes how to process cluster query per SELECT statement by using them.

FROM Statement (Single Table)

Processing a query in a single table is divided into the processing in a sharded table and the processing in a cloned table. If it is processed in a sharded table, then it is divided into n groups and stored, so it reques ts the query to both a local server and a remote server, then collects them and makes them into a result. GOLDILOCKS uses the plan based cluster or the single cluster plan node to request the query to the remo te server and receive the result from it. Those cluster pullers simultaneously requests the query to both a l ocal server and a remote server, then collect the results in parallel to make them into a result set.

The following is an example of processing the query in *part* table which is a sharded table.

```
gSQL> \EXPLAIN PLAN
      SELECT p_name, p_brand, p_type, cluster_group_id
        FROM part;
P_NAME P_BRAND    P_TYPE CLUSTER_GROUP_ID
------ ---------- ------ ----------------
Part#3 Brand#2    STEEL                 1
Part#2 Brand#1    NICKEL                2
Part#5 Brand#3    STEEL                 2
Part#1 Brand#1    COPPER                3
Part#4 Brand#3    NICKEL                3
5 rows selected.
>>>  start print plan
< Execution Plan >
============================================================
¦  IDX  ¦  NODE DESCRIPTION             ¦              ROWS ¦
------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT             ¦                 5 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")  ¦                 5 ¦
```

```
┆    2  ┆      PLAN BASED CLUSTER        ┆ LOCAL/REMOTE     5 ┆
┆    3  ┆        TABLE ACCESS ("PART")   ┆                  1 ┆
============================================================
     1  -  TARGET : PART.P_NAME, PART.P_BRAND, PART.P_TYPE, PART.CLUSTER_GROUP_ID
     2  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."CLUSTER_GROUP_ID", "_A1"."P_NAME",
"_A1"."P_BRAND", "_A1"."P_TYPE" FROM "PUBLIC"."PART"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 2 rows, G3(G3N1,G3N2) 2 rows
     3  -  HASH SHARD ( # 3 )
           READ COLUMN : PART.P_NAME, PART.P_BRAND, PART.P_TYPE
<<<  end print plan
```

When the cloned table is created by using the cloned strategy as AT CLUSTER WIDE strategy, then all nod es have replications, so most of local servers can process the query about the cloned table. However, if a new group or a new member is added, then the group or the member dos not have data about the clone d table. Therefore, it should bring the data from the remote server, so it configures the cluster puller at th is moment. For more information about the cloned strategy, refer to **Cloned Strategy**.

The following is an example of processing the query in *supplier* table which is a cloned table.

```
gSQL> \EXPLAIN PLAN
      SELECT s_name, s_nationkey
        FROM supplier;
S_NAME                    S_NATIONKEY
------------------------- ---------------
Supplier#1                FRANCE
Supplier#2                KOREA
Supplier#3                GERMANY
Supplier#4                UNITED STATES
Supplier#5                CANADA
5 rows selected.
>>>  start print plan
< Execution Plan >
======================================================================
┆  IDX  ┆  NODE DESCRIPTION                        ┆           ROWS ┆
----------------------------------------------------------------------
┆    0  ┆  SELECT STATEMENT                        ┆              5 ┆
┆    1  ┆    QUERY BLOCK ("$QB_IDX_2")             ┆              5 ┆
┆    2  ┆      TABLE ACCESS ("SUPPLIER")           ┆              5 ┆
======================================================================
     1  -  TARGET : SUPPLIER.S_NAME, SUPPLIER.S_NATIONKEY
     2  -  CLONED
           READ COLUMN : SUPPLIER.S_NAME, SUPPLIER.S_NATIONKEY
```

```
<<<  end print plan
```

It can bring all data about *supplier* table from a local server as given above, so it does not configure the cl uster puller.

The following is an example of processing the query in *supplier* table which is a cloned table by using the domain.

```
gSQL> \EXPLAIN PLAN
      SELECT s_name, s_nationkey
        FROM supplier@G2;
S_NAME                    S_NATIONKEY
------------------------- ---------------
Supplier#1                FRANCE
Supplier#2                KOREA
Supplier#3                GERMANY
Supplier#4                UNITED STATES
Supplier#5                CANADA
5 rows selected.
>>>  start print plan
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                      |               ROWS |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                      |                  5 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")           |                  5 |
|    2  |      PLAN BASED CLUSTER                | REMOTE ONLY      5 |
|    3  |        TABLE ACCESS ("SUPPLIER")       |                  0 |
=============================================================================
     1  -  TARGET : SUPPLIER.S_NAME, SUPPLIER.S_NATIONKEY
     2  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."S_NAME", "_A1"."S_NATIONKEY" FROM
"PUBLIC"."SUPPLIER"@LOCAL AS "_A1"
           TARGET DOMAIN : G2(G2N1,G2N2) 5 rows
     3  -  CLONED
           READ COLUMN : SUPPLIER.S_NAME, SUPPLIER.S_NATIONKEY
<<<  end print plan
```

It configures the cluster puller as given above to bring the data about supplier table from the remote serv er.

FROM Statement (Join)

The data collection for the cluster puller plan node are processed in one of the two following forms.

- Data collection from a single group: It is used for a query which consists of cloned tables only or whe n the cluster domain is limited to a single group.
- Data collection from multiple groups: It is used for a query which includes a sharded table.

Join using the cluster puller plan node are also performed in the two forms above.

When performing join by collecting data from a single group, then the join between two cloned tables co nfigures cluster puller by collecting the data.
If the common cluster domain does not exist between two cloned tables, then one of the two following methods can be selected to perform it.

First, it configures the cluster puller per each join target table, then performs the join.

```
gSQL> \EXPLAIN PLAN
      SELECT s_suppkey, n_name
        FROM supplier@g2, nation@g3
       WHERE s_nationkey = n_nationkey;
S_SUPPKEY N_NAME
--------- -------------------------
        1 FRANCE
        2 INDIA
        3 GERMANY
        4 CANADA
        5 UNITED STATES
5 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                          |               ROWS |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                          |                  5 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")               |                  5 |
|    2  |      HASH JOIN (INNER JOIN)                |                  5 |
|    3  |        PLAN BASED CLUSTER                  | REMOTE ONLY      5 |
|    4  |          TABLE ACCESS ("SUPPLIER")         |                  0 |
|    5  |        HASH JOIN INSTANT                   |                  5 |
|    6  |          PLAN BASED CLUSTER                | REMOTE ONLY     30 |
|    7  |            TABLE ACCESS ("NATION")         |                  0 |
================================================================================
```

```
     1  -  TARGET : SUPPLIER.S_SUPPKEY, NATION.N_NAME
     2  -  JOINED COLUMN : SUPPLIER.S_SUPPKEY, NATION.N_NAME
     3  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."S_SUPPKEY", "_A1"."S_NATIONKEY" FROM
"PUBLIC"."SUPPLIER"@LOCAL AS "_A1"
           TARGET DOMAIN : G2(G2N1,G2N2) 5 rows
     4  -  CLONED
           READ COLUMN : SUPPLIER.S_SUPPKEY, SUPPLIER.S_NATIONKEY
     5  -  HASH KEY : NATION.N_NATIONKEY
           RECORD COLUMN : NATION.N_NAME
           READ KEY COLUMN : NATION.N_NATIONKEY, NATION.N_NAME
             HASH FILTER : NATION.N_NATIONKEY = SUPPLIER.S_NATIONKEY
     6  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."N_NATIONKEY", "_A1"."N_NAME" FROM
"PUBLIC"."NATION"@LOCAL AS "_A1"
           TARGET DOMAIN : G3(G3N1,G3N2) 30 rows
     7  -  CLONED
           READ COLUMN : NATION.N_NATIONKEY, NATION.N_NAME
<<<  end print plan
```

However, if it is performed as given above, the problem is that it should bring all data from the two table s.

Second, it configures the pusher table for a table, then performs the join query including the pusher table. In this case, a single group collects the data as follows.

```
gSQL> \EXPLAIN PLAN
      SELECT /*+ REMOTE_JOIN( supplier ) */ s_suppkey, n_name
        FROM supplier@g2, nation@g3
       WHERE s_nationkey = n_nationkey;
S_SUPPKEY N_NAME
--------- -------------------------
        4 CANADA
        1 FRANCE
        3 GERMANY
        2 INDIA
        5 UNITED STATES
5 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                          |             ROWS |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                          |                5 |
```

```
┆    1  ┆    QUERY BLOCK ("$QB_IDX_2")                ┆                5 ┆
┆    2  ┆      SINGLE CLUSTER                         ┆ REMOTE ONLY    5 ┆
┆    3  ┆        CLUSTER PUSHER ("_$NI_7")            ┆                5 ┆
┆    4  ┆          PLAN BASED CLUSTER                 ┆ REMOTE ONLY    5 ┆
┆    5  ┆            TABLE ACCESS ("SUPPLIER")        ┆                0 ┆
┆    6  ┆        HASH JOIN (INNER JOIN)               ┆                0 ┆
┆    7  ┆          TABLE ACCESS ("NATION")            ┆                0 ┆
┆    8  ┆          HASH JOIN INSTANT                  ┆                0 ┆
┆    9  ┆            PUSHER TABLE ACCESS ("_$NI_7")   ┆                0 ┆
=============================================================================
     1  -  TARGET : _$NI_7.S_SUPPKEY, NATION.N_NAME
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE USE_HASH_IN( _A1, 5 ) FULL( _A2 ) FULL( _A1 ) */
"_A1"."S_SUPPKEY", "_A2"."N_NAME" FROM ( "PUBLIC"."NATION"@LOCAL AS "_A2" INNER JOIN
"SESSION_SCHEMA"."_$NI_7"@LOCAL AS "_A1" ON "_A1"."S_NATIONKEY" = "_A2"."N_NATIONKEY") ALIAS
"_A3"
           TARGET DOMAIN : G3(G3N1,G3N2) 5 rows
     3  -  SQL : DECLARE INSTANT TABLE "SESSION_SCHEMA"."_$NI_7" ( "S_NATIONKEY" NUMBER(10,
0), "S_SUPPKEY" NUMBER(10, 0) )
           COLUMN : SUPPLIER.S_NATIONKEY AS S_NATIONKEY, SUPPLIER.S_SUPPKEY AS S_SUPPKEY
           CLONED
           TARGET DOMAIN : G3(G3N1,G3N2) 5 rows
     4  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."S_SUPPKEY", "_A1"."S_NATIONKEY" FROM
"PUBLIC"."SUPPLIER"@LOCAL AS "_A1"
           TARGET DOMAIN : G2(G2N1,G2N2) 5 rows
     5  -  CLONED
           READ COLUMN : SUPPLIER.S_SUPPKEY, SUPPLIER.S_NATIONKEY
     6  -  JOINED COLUMN : _$NI_7.S_SUPPKEY, NATION.N_NAME
     7  -  CLONED
           READ COLUMN : NATION.N_NATIONKEY, NATION.N_NAME
     8  -  HASH KEY : _$NI_7.S_NATIONKEY
           RECORD COLUMN : _$NI_7.S_SUPPKEY
           READ KEY COLUMN : _$NI_7.S_NATIONKEY, _$NI_7.S_SUPPKEY
             HASH FILTER : _$NI_7.S_NATIONKEY = NATION.N_NATIONKEY
<<<  end print plan
```

When performing the join by collecting the data from multiple groups, then the data should be collected from multiple groups when processing the query because the data are distributed into multiple groups in the sharded table.

Joins including the sharded table are classified as follows.

- Case 1: Joining sharded table and cloned table

    - The cloned table data is distributed to all groups where sharded table data were distributed.
- Case 2: Joining sharded table and cloned table
    - The cloned table data is distributed not distributed to one or more groups where sharded table d ata were distributed.
- Case 3: Joining sharded table and sharded table
    - Equi-join condition between sharding keys of two sharded tables exists.
- Case 4: Joining sharded table and sharded table
    - Equi-join condition using the sharding key of a sharded table exists.
- Case 5: Joining sharded table and sharded table
    - Equi-join condition using the sharding key does not exist.

In case 1, if the cloned table data exists in all groups where the sharded table data is distributed, then the total sum of join results from each group becomes the entire join result.

```
gSQL> \EXPLAIN PLAN
      SELECT ps_partkey, s_name
        FROM partsupp, supplier
       WHERE ps_suppkey = s_suppkey;
PS_PARTKEY S_NAME
---------- -------------------------
         3 Supplier#1
         3 Supplier#4
         2 Supplier#2
         2 Supplier#5
         5 Supplier#1
         5 Supplier#4
         1 Supplier#2
         1 Supplier#3
         4 Supplier#3
         4 Supplier#5
10 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
¦IDX¦ NODE DESCRIPTION                                ¦           ROWS ¦
--------------------------------------------------------------------------------
¦ 0 ¦ SELECT STATEMENT                                ¦             10 ¦
¦ 1 ¦  QUERY BLOCK ("$QB_IDX_2")                      ¦             10 ¦
¦ 2 ¦   PLAN BASED CLUSTER                            ¦ LOCAL/REMOTE 10 ¦
¦ 3 ¦    HASH JOIN (INNER JOIN)                       ¦              2 ¦
¦ 4 ¦     INDEX ACCESS ("PARTSUPP", "PARTSUPP_PK_INDEX")¦ (    2)       2 ¦
¦ 5 ¦     HASH JOIN INSTANT                           ¦              2 ¦
```

```
┆ 6 ┆      TABLE ACCESS ("SUPPLIER")                    ┆              5 ┆
=============================================================================
     1  -  TARGET : PARTSUPP.PS_PARTKEY, SUPPLIER.S_NAME
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE USE_HASH_IN( _A1, 5 ) INDEX( _A2,
"PUBLIC"."PARTSUPP_PK_INDEX" ) FULL( _A1 ) */ "_A2"."PS_PARTKEY", "_A1"."S_NAME" FROM (
"PUBLIC"."PARTSUPP"@LOCAL AS "_A2" INNER JOIN "PUBLIC"."SUPPLIER"@LOCAL AS "_A1" ON
"_A1"."S_SUPPKEY" = "_A2"."PS_SUPPKEY") ALIAS "_A3"
           TARGET DOMAIN : G1(G1N1,G1N2) 2 rows, G2(G2N1,G2N2) 4 rows, G3(G3N1,G3N2) 4 rows
     3  -  JOINED COLUMN : PARTSUPP.PS_PARTKEY, SUPPLIER.S_NAME
     4  -  HASH SHARD ( # 3 )
           READ INDEX COLUMN : PARTSUPP.PS_PARTKEY, PARTSUPP.PS_SUPPKEY
     5  -  HASH KEY : SUPPLIER.S_SUPPKEY
           RECORD COLUMN : SUPPLIER.S_NAME
           READ KEY COLUMN : SUPPLIER.S_SUPPKEY, SUPPLIER.S_NAME
             HASH FILTER : SUPPLIER.S_SUPPKEY = PARTSUPP.PS_SUPPKEY
           FETCH ONE ROW
     6  -  CLONED
           READ COLUMN : SUPPLIER.S_SUPPKEY, SUPPLIER.S_NAME
<<<  end print plan
```

In case 2, it can not configure the generated query including the join. In this case, it configures the cluster puller for each table which is a join target, then performs the join as follows.

```
gSQL> \EXPLAIN PLAN
      SELECT ps_partkey, s_name
        FROM partsupp, supplier@G2¦G3
       WHERE ps_suppkey = s_suppkey;
PS_PARTKEY S_NAME
---------- -------------------------
         3 Supplier#1
         3 Supplier#4
         1 Supplier#2
         1 Supplier#3
         4 Supplier#3
         4 Supplier#5
         2 Supplier#2
         2 Supplier#5
         5 Supplier#1
         5 Supplier#4
10 rows selected.
>>>  start print plan
```

```
< Execution Plan >
================================================================================
¦IDX¦ NODE DESCRIPTION                                ¦           ROWS ¦
--------------------------------------------------------------------------------
¦ 0 ¦SELECT STATEMENT                                 ¦             10 ¦
¦ 1 ¦ QUERY BLOCK ("$QB_IDX_2")                       ¦             10 ¦
¦ 2 ¦  HASH JOIN (INNER JOIN)                         ¦             10 ¦
¦ 3 ¦   PLAN BASED CLUSTER                            ¦ LOCAL/REMOTE 10 ¦
¦ 4 ¦    INDEX ACCESS ("PARTSUPP", "PARTSUPP_PK_INDEX") ¦(    2)       2 ¦
¦ 5 ¦   HASH JOIN INSTANT                             ¦             10 ¦
¦ 6 ¦    PLAN BASED CLUSTER                           ¦ REMOTE ONLY   5 ¦
¦ 7 ¦     TABLE ACCESS ("SUPPLIER")                   ¦              0 ¦
================================================================================
     1  -  TARGET : PARTSUPP.PS_PARTKEY, SUPPLIER.S_NAME
     2  -  JOINED COLUMN : PARTSUPP.PS_PARTKEY, SUPPLIER.S_NAME
     3  -  SQL : SELECT /*+ INDEX( _A1, "PUBLIC"."PARTSUPP_PK_INDEX" ) */ "_A1"."PS_PARTKEY",
"_A1"."PS_SUPPKEY" FROM "PUBLIC"."PARTSUPP"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 2 rows, G2(G2N1,G2N2) 4 rows, G3(G3N1,G3N2) 4 rows
     4  -  HASH SHARD ( # 3 )
           READ INDEX COLUMN : PARTSUPP.PS_PARTKEY, PARTSUPP.PS_SUPPKEY
     5  -  HASH KEY : SUPPLIER.S_SUPPKEY
           RECORD COLUMN : SUPPLIER.S_NAME
           READ KEY COLUMN : SUPPLIER.S_SUPPKEY, SUPPLIER.S_NAME
             HASH FILTER : SUPPLIER.S_SUPPKEY = PARTSUPP.PS_SUPPKEY
     6  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."S_SUPPKEY", "_A1"."S_NAME" FROM
"PUBLIC"."SUPPLIER"@LOCAL AS "_A1"
           TARGET DOMAIN : G2(G2N1,G2N2) 5 rows, G3(G3N1,G3N2) 0 rows
     7  -  CLONED
           READ COLUMN : SUPPLIER.S_SUPPKEY, SUPPLIER.S_NAME
<<<  end print plan
```

In case 3, if sharding strategies for two tables are same, then the total sum of join results from each grou p is the entire join result. If sharding strategies of two tables are different, then the situation is as same as the case 4.

If sharding strategies of two tables are same, them the case 3 is performed as follows.

```
gSQL> \EXPLAIN PLAN
      SELECT p_name, ps_suppkey
        FROM part, partsupp
       WHERE p_partkey = ps_partkey;
P_NAME PS_SUPPKEY
```

```
------ ----------
Part#3          4
Part#3          1
Part#2          5
Part#2          2
Part#5          4
Part#5          1
Part#1          3
Part#1          2
Part#4          5
Part#4          3
10 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
¦IDX¦ NODE DESCRIPTION                                ¦            ROWS ¦
--------------------------------------------------------------------------------
¦ 0 ¦SELECT STATEMENT                                 ¦              10 ¦
¦ 1 ¦ QUERY BLOCK ("$QB_IDX_2")                       ¦              10 ¦
¦ 2 ¦  PLAN BASED CLUSTER                             ¦ LOCAL/REMOTE 10 ¦
¦ 3 ¦   HASH JOIN (INNER JOIN)                        ¦               2 ¦
¦ 4 ¦    TABLE ACCESS ("PART")                        ¦               1 ¦
¦ 5 ¦    HASH JOIN INSTANT                            ¦               2 ¦
¦ 6 ¦     INDEX ACCESS ("PARTSUPP", "PARTSUPP_PK_INDEX")¦ (      2)     2 ¦
================================================================================
     1  -  TARGET : PART.P_NAME, PARTSUPP.PS_SUPPKEY
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE USE_HASH_IN( _A1, 10 ) FULL( _A2 ) INDEX( _A1,
"PUBLIC"."PARTSUPP_PK_INDEX" ) */ "_A2"."P_NAME", "_A1"."PS_SUPPKEY" FROM (
"PUBLIC"."PART"@LOCAL AS "_A2" INNER JOIN "PUBLIC"."PARTSUPP"@LOCAL AS "_A1" ON
"_A1"."PS_PARTKEY" = "_A2"."P_PARTKEY") ALIAS "_A3"
           TARGET DOMAIN : G1(G1N1,G1N2) 2 rows, G2(G2N1,G2N2) 4 rows, G3(G3N1,G3N2) 4 rows
     3  -  JOINED COLUMN : PART.P_NAME, PARTSUPP.PS_SUPPKEY
     4  -  HASH SHARD ( # 3 )
           READ COLUMN : PART.P_PARTKEY, PART.P_NAME
     5  -  HASH KEY : PARTSUPP.PS_PARTKEY
           RECORD COLUMN : PARTSUPP.PS_SUPPKEY
           READ KEY COLUMN : PARTSUPP.PS_PARTKEY, PARTSUPP.PS_SUPPKEY
             HASH FILTER : PARTSUPP.PS_PARTKEY = PART.P_PARTKEY
     6  -  HASH SHARD ( # 3 )
           READ INDEX COLUMN : PARTSUPP.PS_PARTKEY, PARTSUPP.PS_SUPPKEY
<<<  end print plan
```

In case 4, it can not configure the generated query including the join for two sharded tables. It can proce ss the join as same as the case of 1 by creating the pusher table for the opposite table of the table which used the sharding key in the equi-join condition.

The case 4 is performed for two sharded tables as follows.

```
gSQL> \EXPLAIN PLAN
      SELECT /*+ REMOTE_JOIN( part ) */ p_name, ps_suppkey
        FROM part, partsupp
       WHERE p_partkey = ps_suppkey;
P_NAME PS_SUPPKEY
------ ----------
Part#3          3
Part#3          3
Part#1          1
Part#1          1
Part#4          4
Part#4          4
Part#2          2
Part#2          2
Part#5          5
Part#5          5
10 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
|IDX| NODE DESCRIPTION                                  |            ROWS |
--------------------------------------------------------------------------------
|  0|SELECT STATEMENT                                  |              10 |
|  1| QUERY BLOCK ("$QB_IDX_2")                        |              10 |
|  2|  SINGLE CLUSTER                                  | LOCAL/REMOTE 10 |
|  3|   CLUSTER PUSHER ("_$NI_7")                      |              10 |
|  4|    PLAN BASED CLUSTER                            | LOCAL/REMOTE 10 |
|  5|     INDEX ACCESS ("PARTSUPP", "PARTSUPP_PK_INDEX")| (     2)      2 |
|  6|   SELECT STATEMENT                               |               2 |
|  7|    QUERY BLOCK ("$QB_IDX_2")                     |               2 |
|  8|     HASH JOIN (INNER JOIN)                       |               2 |
|  9|      TABLE ACCESS ("PART" AS _A2)                |               1 |
| 10|      HASH JOIN INSTANT                           |               2 |
| 11|       PUSHER TABLE ACCESS ("_$NI_7" AS _A1)      |               2 |
================================================================================
     1  -  TARGET : PART.P_NAME, _$NI_7.PS_SUPPKEY
```

```
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE USE_HASH_IN( _A1, 10 ) FULL( _A2 ) FULL( _A1 )
*/ "_A2"."P_NAME", "_A1"."PS_SUPPKEY" FROM ( "PUBLIC"."PART"@LOCAL AS "_A2" INNER JOIN
"SESSION_SCHEMA"."_$NI_7"@LOCAL AS "_A1" ON "_A1"."PS_SUPPKEY" = "_A2"."P_PARTKEY") ALIAS
"_A3"
           TARGET DOMAIN : G1(G1N1,G1N2) 2 rows, G2(G2N1,G2N2) 4 rows, G3(G3N1,G3N2) 4 rows
     3  -  SQL : DECLARE INSTANT TABLE "SESSION_SCHEMA"."_$NI_7" ( "PS_SUPPKEY" NUMBER(10, 0)
)
           COLUMN : PARTSUPP.PS_SUPPKEY AS PS_SUPPKEY
           SHARDED : PARTSUPP.PS_SUPPKEY
           TARGET DOMAIN : G1(G1N1,G1N2) 2 rows, G2(G2N1,G2N2) 4 rows, G3(G3N1,G3N2) 4 rows
     4  -  SQL : SELECT /*+ INDEX( _A1, "PUBLIC"."PARTSUPP_PK_INDEX" ) */ "_A1"."PS_SUPPKEY"
FROM "PUBLIC"."PARTSUPP"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 2 rows, G2(G2N1,G2N2) 4 rows, G3(G3N1,G3N2) 4 rows
     5  -  HASH SHARD ( # 3 )
           READ INDEX COLUMN : PARTSUPP.PS_SUPPKEY
     7  -  TARGET : _A2.P_NAME, _A1.PS_SUPPKEY
     8  -  JOINED COLUMN : _A2.P_NAME, _A1.PS_SUPPKEY
     9  -  HASH SHARD ( # 3 )
           READ COLUMN : _A2.P_PARTKEY, _A2.P_NAME
    10  -  HASH KEY : _A1.PS_SUPPKEY
           READ KEY COLUMN : _A1.PS_SUPPKEY
             HASH FILTER : _A1.PS_SUPPKEY = _A2.P_PARTKEY
    11  -  READ COLUMN : _A1.PS_SUPPKEY
<<<  end print plan
```

In case 5, it can not use the sharding strategy for two sharded tables. In this case, it can perform the join by configuring one of the join target table into the pusher table in a cloned table form with as follows.

```
gSQL> \EXPLAIN PLAN
      SELECT SUM( A.p_size )
        FROM part A, part B
       WHERE A.p_type = B.p_type;
SUM( A.P_SIZE )
---------------
            109
1 row selected.
>>>  start print plan
< Execution Plan >
================================================================================
¦IDX¦  NODE DESCRIPTION                                ¦           ROWS ¦
--------------------------------------------------------------------------------
```

```
|  0|SELECT STATEMENT                                  |              1 |
|  1|  QUERY BLOCK ("$QB_IDX_2")                       |              1 |
|  2|    SINGLE CLUSTER                                | LOCAL/REMOTE 1 |
|  3|      CLUSTER PUSHER ("_$NI_6")                   |              5 |
|  4|        PLAN BASED CLUSTER                        | LOCAL/REMOTE 5 |
|  5|          TABLE ACCESS ("PART" AS B)              |              1 |
|  6|      SELECT STATEMENT                            |              1 |
|  7|        QUERY BLOCK ("$QB_IDX_2")                 |              1 |
|  8|          AGGREGATION BY HASH                     |              1 |
|  9|            HASH JOIN (INNER JOIN)                |              2 |
| 10|              PUSHER TABLE ACCESS ("_$NI_6" AS _A2) |              5 |
| 11|              HASH JOIN INSTANT                   |              2 |
| 12|                TABLE ACCESS ("PART" AS _A1)      |              1 |
================================================================================
     1  -  TARGET : SUM( A.P_SIZE )
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE USE_HASH_IN( _A1, 10 ) FULL( _A2 ) FULL( _A1 )
*/ SUM( "_A1"."P_SIZE" ) FROM ( "SESSION_SCHEMA"."_$NI_6"@LOCAL AS "_A2" INNER JOIN
"PUBLIC"."PART"@LOCAL AS "_A1" ON "_A1"."P_TYPE" = "_A2"."P_TYPE") ALIAS "_A3"
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 1 rows, G3(G3N1,G3N2) 1 rows
           RE-AGGREGATION
             AGGREGATION : SUM( SUM( A.P_SIZE ) )
     3  -  SQL : DECLARE INSTANT TABLE "SESSION_SCHEMA"."_$NI_6" ( "P_TYPE" VARCHAR(25 OCTETS)
)
           COLUMN : B.P_TYPE AS P_TYPE
           CLONED
           TARGET DOMAIN : G1(G1N1,G1N2) 5 rows, G2(G2N1,G2N2) 5 rows, G3(G3N1,G3N2) 5 rows
     4  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."P_TYPE" FROM "PUBLIC"."PART"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 2 rows, G3(G3N1,G3N2) 2 rows
     5  -  HASH SHARD ( # 3 )
           READ COLUMN : B.P_TYPE
     7  -  TARGET : SUM( _A1.P_SIZE )
     8  -  AGGREGATION : SUM( _A1.P_SIZE )
     9  -  JOINED COLUMN : _A1.P_SIZE
    10  -  READ COLUMN : _A2.P_TYPE
    11  -  HASH KEY : _A1.P_TYPE
           RECORD COLUMN : _A1.P_SIZE
           READ KEY COLUMN : _A1.P_TYPE, _A1.P_SIZE
             HASH FILTER : _A1.P_TYPE = _A2.P_TYPE
    12  -  HASH SHARD ( # 3 )
           READ COLUMN : _A1.P_TYPE, _A1.P_SIZE
<<<  end print plan
```

FROM Statement (Outer Join)

The query process in outer join is divided into the data collection from a single group and the data collecti on from multiple groups like as **FROM Statement (Join)**.

Outer join which collects data from a single group is applied only to a query which consists of cloned tabl es only or when the cluster domain is limited to a single group. It configures the join result by performing the generated query including the outer join in a single group as follows.

```
gSQL> \EXPLAIN PLAN
      SELECT s_suppkey, n_name
        FROM supplier@g2
             LEFT OUTER JOIN
             nation@g2
             ON s_nationkey = n_nationkey;
S_SUPPKEY N_NAME
--------- -------------------------
        4 CANADA
        1 FRANCE
        3 GERMANY
        2 INDIA
        5 UNITED STATES
5 rows selected.
>>>  start print plan
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                          |           ROWS |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                          |              5 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")               |              5 |
|    2  |      SINGLE CLUSTER                        | REMOTE ONLY  5 |
|    3  |        HASH JOIN (INVERTED LEFT OUTER JOIN) |              0 |
|    4  |          TABLE ACCESS ("NATION")           |              0 |
|    5  |          HASH JOIN INSTANT                 |              0 |
|    6  |            TABLE ACCESS ("SUPPLIER")       |              0 |
=============================================================================
     1  -  TARGET : SUPPLIER.S_SUPPKEY, NATION.N_NAME
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE USE_HASH_IN( _A1, 5 ) FULL( _A2 ) FULL( _A1 ) */
"_A1"."S_SUPPKEY", "_A2"."N_NAME" FROM ( "PUBLIC"."SUPPLIER"@"G2N1"|"G2N2" AS "_A1" LEFT OUTER
JOIN "PUBLIC"."NATION"@"G2N1"|"G2N2" AS "_A2" ON "_A1"."S_NATIONKEY" = "_A2"."N_NATIONKEY")
ALIAS "_A3"
           TARGET DOMAIN : G2(G2N1,G2N2) 5 rows
```

```
    3  -  JOINED COLUMN : SUPPLIER.S_SUPPKEY, NATION.N_NAME
    4  -  CLONED
          READ COLUMN : NATION.N_NATIONKEY, NATION.N_NAME
    5  -  HASH KEY : SUPPLIER.S_NATIONKEY
          RECORD COLUMN : SUPPLIER.S_SUPPKEY
          READ KEY COLUMN : SUPPLIER.S_NATIONKEY, SUPPLIER.S_SUPPKEY
            HASH FILTER : SUPPLIER.S_NATIONKEY = NATION.N_NATIONKEY
    6  -  CLONED
          READ COLUMN : SUPPLIER.S_SUPPKEY, SUPPLIER.S_NATIONKEY
<<<  end print plan
```

Outer join which collects data from multiple groups is applied to following three cases.

- Case 1: Joining sharded table and cloned table
  - The cloned table data is distributed to all groups where sharded table data were distributed.
- Case 2: Joining sharded table and sharded table
  - Equi-join condition between sharding keys of two sharded tables exists.
- Case 3: It includes one or more sharded tables and equi-join condition does not exists between shardi ng keys.

In case 1, it configures the generated query including the outer join.
The total sum of outer join results from each group becomes the entire join result as follows.

```
gSQL> \EXPLAIN PLAN
      SELECT p_name, ps_suppkey
        FROM part
             LEFT OUTER JOIN
             partsupp
             ON p_partkey = ps_partkey;
P_NAME PS_SUPPKEY
------ ----------
Part#3          4
Part#3          1
Part#1          3
Part#1          2
Part#4          5
Part#4          3
Part#2          5
Part#2          2
Part#5          4
Part#5          1
10 rows selected.
```

```
>>>  start print plan
< Execution Plan >
=============================================================================
¦IDX¦ NODE DESCRIPTION                                ¦            ROWS ¦
-----------------------------------------------------------------------------
¦ 0 ¦ SELECT STATEMENT                                ¦              10 ¦
¦ 1 ¦  QUERY BLOCK ("$QB_IDX_2")                      ¦              10 ¦
¦ 2 ¦   SINGLE CLUSTER                                ¦ LOCAL/REMOTE 10 ¦
¦ 3 ¦    SELECT STATEMENT                             ¦               2 ¦
¦ 4 ¦     QUERY BLOCK ("$QB_IDX_2")                   ¦               2 ¦
¦ 5 ¦      HASH JOIN (LEFT OUTER JOIN)                ¦               2 ¦
¦ 6 ¦       TABLE ACCESS ("PART" AS _A2)              ¦               1 ¦
¦ 7 ¦       HASH JOIN INSTANT                         ¦               2 ¦
¦ 8 ¦        INDEX ACCESS ("PARTSUPP" AS _A1, ... )   ¦ (     2)      2 ¦
=============================================================================
     1  -  TARGET : PART.P_NAME, PARTSUPP.PS_SUPPKEY
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE USE_HASH_IN( _A1, 10 ) FULL( _A2 ) INDEX( _A1,
"PUBLIC"."PARTSUPP_PK_INDEX" ) */ "_A2"."P_NAME", "_A1"."PS_SUPPKEY" FROM (
"PUBLIC"."PART"@"G1N1"¦"G1N2"¦"G2N1"¦"G2N2"¦"G3N1"¦"G3N2" AS "_A2" LEFT OUTER JOIN
"PUBLIC"."PARTSUPP"@"G1N1"¦"G1N2"¦"G2N1"¦"G2N2"¦"G3N1"¦"G3N2" AS "_A1" ON "_A1"."PS_PARTKEY" =
"_A2"."P_PARTKEY") ALIAS "_A3"
           TARGET DOMAIN : G1(G1N1,G1N2) 2 rows, G2(G2N1,G2N2) 4 rows, G3(G3N1,G3N2) 4 rows
     4  -  TARGET : _A2.P_NAME, _A1.PS_SUPPKEY
     5  -  JOINED COLUMN : _A2.P_NAME, _A1.PS_SUPPKEY
     6  -  HASH SHARD ( # 3 )
           READ COLUMN : _A2.P_PARTKEY, _A2.P_NAME
     7  -  HASH KEY : _A1.PS_PARTKEY
           RECORD COLUMN : _A1.PS_SUPPKEY
           READ KEY COLUMN : _A1.PS_PARTKEY, _A1.PS_SUPPKEY
             HASH FILTER : _A1.PS_PARTKEY = _A2.P_PARTKEY
     8  -  HASH SHARD ( # 3 )
           READ INDEX COLUMN : _A1.PS_PARTKEY, _A1.PS_SUPPKEY
<<<  end print plan
```

In case 2, it configures the generated query including the outer join like as case 1.
The total sum of outer join results from each group becomes the entire join result as follows.

```
gSQL> \EXPLAIN PLAN
      SELECT p_name, ps_suppkey
        FROM part
             LEFT OUTER JOIN
```

```
            partsupp
            ON p_partkey = ps_partkey;
P_NAME PS_SUPPKEY
------ ----------
Part#3          4
Part#3          1
Part#1          3
Part#1          2
Part#4          5
Part#4          3
Part#2          5
Part#2          2
Part#5          4
Part#5          1
10 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
¦IDX¦ NODE DESCRIPTION                                ¦            ROWS ¦
--------------------------------------------------------------------------------
¦ 0 ¦ SELECT STATEMENT                                ¦              10 ¦
¦ 1 ¦  QUERY BLOCK ("$QB_IDX_2")                      ¦              10 ¦
¦ 2 ¦   SINGLE CLUSTER                                ¦ LOCAL/REMOTE 10 ¦
¦ 3 ¦    SELECT STATEMENT                             ¦               2 ¦
¦ 4 ¦     QUERY BLOCK ("$QB_IDX_2")                   ¦               2 ¦
¦ 5 ¦      HASH JOIN (LEFT OUTER JOIN)                ¦               2 ¦
¦ 6 ¦       TABLE ACCESS ("PART" AS _A2)              ¦               1 ¦
¦ 7 ¦       HASH JOIN INSTANT                         ¦               2 ¦
¦ 8 ¦        INDEX ACCESS ("PARTSUPP" AS _A1, ... )   ¦ (     2)      2 ¦
================================================================================
     1  -  TARGET : PART.P_NAME, PARTSUPP.PS_SUPPKEY
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE USE_HASH_IN( _A1, 10 ) FULL( _A2 ) INDEX( _A1,
"PUBLIC"."PARTSUPP_PK_INDEX" ) */ "_A2"."P_NAME", "_A1"."PS_SUPPKEY" FROM (
"PUBLIC"."PART"@"G1N1"¦"G1N2"¦"G2N1"¦"G2N2"¦"G3N1"¦"G3N2" AS "_A2" LEFT OUTER JOIN
"PUBLIC"."PARTSUPP"@"G1N1"¦"G1N2"¦"G2N1"¦"G2N2"¦"G3N1"¦"G3N2" AS "_A1" ON "_A1"."PS_PARTKEY" =
"_A2"."P_PARTKEY") ALIAS "_A3"
            TARGET DOMAIN : G1(G1N1,G1N2) 2 rows, G2(G2N1,G2N2) 4 rows, G3(G3N1,G3N2) 4 rows
     4  -  TARGET : _A2.P_NAME, _A1.PS_SUPPKEY
     5  -  JOINED COLUMN : _A2.P_NAME, _A1.PS_SUPPKEY
     6  -  HASH SHARD ( # 3 )
            READ COLUMN : _A2.P_PARTKEY, _A2.P_NAME
```

```
     7  -  HASH KEY : _A1.PS_PARTKEY
           RECORD COLUMN : _A1.PS_SUPPKEY
           READ KEY COLUMN : _A1.PS_PARTKEY, _A1.PS_SUPPKEY
             HASH FILTER : _A1.PS_PARTKEY = _A2.P_PARTKEY
     8  -  HASH SHARD ( # 3 )
           READ INDEX COLUMN : _A1.PS_PARTKEY, _A1.PS_SUPPKEY
<<<  end print plan
```

In case 3, the total sum of the results of the generated query including the outer join from each group be comes the duplicate entire anti join result like as case 1. It manipulates the data in intersect key group me thod to delete the duplicate anti join results. For more information about the intersect key group, refer to **Generated Query for Intersect Key Group**.

The following is an example of processing the intersect key group for the outer join.

```
gSQL> \EXPLAIN PLAN
      SELECT /*+ REMOTE_JOIN( supplier ) */ ps_partkey, s_name
        FROM supplier
             LEFT OUTER JOIN
             partsupp
             ON ps_suppkey = s_suppkey;
PS_PARTKEY S_NAME
---------- -------------------------
         3 Supplier#1
         5 Supplier#1
         2 Supplier#2
         1 Supplier#2
         1 Supplier#3
         4 Supplier#3
         3 Supplier#4
         5 Supplier#4
         2 Supplier#5
         4 Supplier#5
10 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
|IDX| NODE DESCRIPTION                                 |            ROWS |
--------------------------------------------------------------------------------
| 0 |SELECT STATEMENT                                 |              10 |
| 1 |  QUERY BLOCK ("$QB_IDX_2")                      |              10 |
| 2 |    MULTIPLE CLUSTER                             | LOCAL/REMOTE 10 |
```

```
| 3 |      SELECT STATEMENT                              |              5 |
| 4 |        QUERY BLOCK ("$QB_IDX_2")                   |              5 |
| 5 |          SORT INSTANT                              |              5 |
| 6 |            HASH JOIN (INVERTED LEFT OUTER JOIN)    |              5 |
| 7 |              INDEX ACCESS ("PARTSUPP" AS _A2, ... | (     2)     2 |
| 8 |              HASH JOIN INSTANT                     |              5 |
| 9 |                TABLE ACCESS ("SUPPLIER" AS _A1)    |              5 |
=============================================================================
     1  -  TARGET : PARTSUPP.PS_PARTKEY, SUPPLIER.S_NAME
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE USE_HASH_IN( _A1, 5 ) INDEX( _A2,
"PUBLIC"."PARTSUPP_PK_INDEX" ) FULL( _A1 ) */ "_A1"."S_SUPPKEY", "_A2"."PS_SUPPKEY",
"_A2"."PS_PARTKEY", "_A1"."S_NAME" FROM (
"PUBLIC"."SUPPLIER"@"G1N1"¦"G1N2"¦"G2N1"¦"G2N2"¦"G3N1"¦"G3N2" AS "_A1" LEFT OUTER JOIN
"PUBLIC"."PARTSUPP"@"G1N1"¦"G1N2"¦"G2N1"¦"G2N2"¦"G3N1"¦"G3N2" AS "_A2" ON "_A1"."S_SUPPKEY" =
"_A2"."PS_SUPPKEY") ALIAS "_A3" ORDER BY "_A1"."S_SUPPKEY" ASC NULLS LAST
           TARGET DOMAIN : G1(G1N1,G1N2) 5 rows, G2(G2N1,G2N2) 5 rows, G3(G3N1,G3N2) 6 rows
           INTERSECT KEY GROUP
             KEY GROUP : SUPPLIER.S_SUPPKEY
             Nil Expression : PARTSUPP.PS_SUPPKEY
     4  -  TARGET : _A1.S_SUPPKEY, _A2.PS_SUPPKEY, _A2.PS_PARTKEY, _A1.S_NAME
     5  -  SORT KEY : "_A1.S_SUPPKEY ASC NULLS LAST"
           RECORD COLUMN : _A2.PS_SUPPKEY, _A2.PS_PARTKEY, _A1.S_NAME
           READ KEY COLUMN : _A1.S_SUPPKEY
           READ RECORD COLUMN : _A2.PS_SUPPKEY, _A2.PS_PARTKEY, _A1.S_NAME
     6  -  JOINED COLUMN : _A1.S_SUPPKEY, _A2.PS_SUPPKEY, _A2.PS_PARTKEY, _A1.S_NAME
     7  -  HASH SHARD ( # 3 )
           READ INDEX COLUMN : _A2.PS_PARTKEY, _A2.PS_SUPPKEY
     8  -  HASH KEY : _A1.S_SUPPKEY
           RECORD COLUMN : _A1.S_NAME
           READ KEY COLUMN : _A1.S_SUPPKEY, _A1.S_NAME
             HASH FILTER : _A1.S_SUPPKEY = _A2.PS_SUPPKEY
     9  -  CLONED
           READ COLUMN : _A1.S_SUPPKEY, _A1.S_NAME
<<<  end print plan
```

The generated query includes the ordering for columns included in an equi-join condition as above. It collects data collected from each group in an order of the ordering, then applies the intersect key group only to the case whose nil expression value is null.

FROM Statement (Join Including Subquery)

The join including the subquery is classified according to whether the subquery is used as a join condition. When processing the cluster of the join which does not use the subquery as a join condition, it collects an d manipulates the data through the cluster puller, then applies the filter related to the subquery. The join which uses the subquery as a join condition processes the join according to the join operation.

The following is an example of processing the subquery which is not used as the join condition.

```
gSQL> \EXPLAIN PLAN
      SELECT p_name, ps_suppkey
        FROM part, partsupp
       WHERE p_partkey = ps_partkey
             AND ps_suppkey IN ( SELECT /*+ NO_UNNEST */ s_suppkey FROM supplier );
P_NAME PS_SUPPKEY
------ ----------
Part#3          4
Part#3          1
Part#2          5
Part#2          2
Part#5          4
Part#5          1
Part#1          3
Part#1          2
Part#4          5
Part#4          3
10 rows selected.
>>>  start print plan
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                          |             ROWS |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                          |               10 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")               |               10 |
|    2  |      PLAN BASED CLUSTER                    | LOCAL/REMOTE  10 |
|    3  |        HASH JOIN (INNER JOIN)              |                2 |
|    4  |          TABLE ACCESS ("PART")             |                1 |
|    5  |          HASH JOIN INSTANT                 |                2 |
|    6  |            INDEX ACCESS ("PARTSUPP", ...)  | (      2)      2 |
|    7  |  SUB QUERY LIST                            |                  |
|    8  |    INLINE_VIEW ("$V8") (MATERIALIZED)      |               10 |
|    9  |      QUERY BLOCK ("$QB_IDX_8")             |                5 |
```

```
|   10  |        INDEX ACCESS ("SUPPLIER", ...)     | (      5)      5 |
=========================================================================
     1  -  TARGET : PART.P_NAME, PARTSUPP.PS_SUPPKEY
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE USE_HASH_IN( _A1, 10 ) FULL( _A2 ) INDEX( _A1,
"PUBLIC"."PARTSUPP_PK_INDEX" ) */ "_A1"."PS_SUPPKEY", "_A2"."P_NAME" FROM (
"PUBLIC"."PART"@LOCAL AS "_A2" INNER JOIN "PUBLIC"."PARTSUPP"@LOCAL AS "_A1" ON
"_A1"."PS_PARTKEY" = "_A2"."P_PARTKEY") ALIAS "_A3"
           TARGET DOMAIN : G1(G1N1,G1N2) 2 rows, G2(G2N1,G2N2) 4 rows, G3(G3N1,G3N2) 4 rows
             POST FILTER : ( PARTSUPP.PS_SUPPKEY ) IN ( $V8.S_SUPPKEY )
     3  -  JOINED COLUMN : PARTSUPP.PS_SUPPKEY, PART.P_NAME
     4  -  HASH SHARD ( # 3 )
           READ COLUMN : PART.P_PARTKEY, PART.P_NAME
     5  -  HASH KEY : PARTSUPP.PS_PARTKEY
           RECORD COLUMN : PARTSUPP.PS_SUPPKEY
           READ KEY COLUMN : PARTSUPP.PS_PARTKEY, PARTSUPP.PS_SUPPKEY
             HASH FILTER : PARTSUPP.PS_PARTKEY = PART.P_PARTKEY
     6  -  HASH SHARD ( # 3 )
           READ INDEX COLUMN : PARTSUPP.PS_PARTKEY, PARTSUPP.PS_SUPPKEY
     8  -  COLUMN : SUPPLIER.S_SUPPKEY AS S_SUPPKEY
     9  -  TARGET : SUPPLIER.S_SUPPKEY
    10  -  CLONED
           READ INDEX COLUMN : SUPPLIER.S_SUPPKEY
<<<  end print plan
```

Joins which use the subquery as a join condition are classified according to the operator including the sub query as follows.

Table 12-11 Join including subquery

Join operation	Operator including the subquery
INNER JOIN	All operators except for **〈Group Comparison Conditions〉**
OUTER JOIN	All operators except for **〈Group Comparison Conditions〉**
SEMI JOIN	**〈Group Comparison Conditions〉** having EXISTS, IN, ANY quantifier
ANTI-SEMI JOIN	**〈Group Comparison Conditions〉** having NOT EXISTS, NOT IN, ALL quantifier

For more information about the join operation, refer to **Join**.

If a join condition or a filter, which includes a subquery is used in an inner join, then it collects the data an d applies the filter related to the subquery as follows.

```
gSQL> \EXPLAIN PLAN
      SELECT ps_partkey, s_name
        FROM supplier
```

```
            INNER JOIN
            partsupp
            ON ps_suppkey = s_suppkey
               AND ps_suppkey = ( SELECT s_suppkey FROM DUAL );
PS_PARTKEY S_NAME
---------- -------------------------
         3 Supplier#1
         3 Supplier#4
         1 Supplier#2
         1 Supplier#3
         4 Supplier#3
         4 Supplier#5
         2 Supplier#2
         2 Supplier#5
         5 Supplier#1
         5 Supplier#4
10 rows selected.
>>>  start print plan
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                        |             ROWS |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                        |               10 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")             |               10 |
|    2  |      PLAN BASED CLUSTER                  | LOCAL/REMOTE  10 |
|    3  |        HASH JOIN (INNER JOIN)            |                2 |
|    4  |          INDEX ACCESS ("PARTSUPP", ...)  | (      2)      2 |
|    5  |          HASH JOIN INSTANT               |                2 |
|    6  |            TABLE ACCESS ("SUPPLIER")     |                5 |
|    7  |      SUB QUERY LIST                      |                  |
|    8  |        INLINE_VIEW ("$V8")               |               10 |
|    9  |          QUERY BLOCK ("$QB_IDX_8")       |               10 |
|   10  |            FAST DUAL ACCESS ("DUAL")     |               10 |
=============================================================================
     1  -  TARGET : PARTSUPP.PS_PARTKEY, SUPPLIER.S_NAME
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE USE_HASH_IN( _A1, 5 ) INDEX( _A2,
"PUBLIC"."PARTSUPP_PK_INDEX" ) FULL( _A1 ) */ "_A2"."PS_SUPPKEY", "_A1"."S_SUPPKEY",
"_A2"."PS_PARTKEY", "_A1"."S_NAME" FROM ( "PUBLIC"."PARTSUPP"@LOCAL AS "_A2" INNER JOIN
"PUBLIC"."SUPPLIER"@LOCAL AS "_A1" ON "_A1"."S_SUPPKEY" = "_A2"."PS_SUPPKEY") ALIAS "_A3"
           TARGET DOMAIN : G1(G1N1,G1N2) 2 rows, G2(G2N1,G2N2) 4 rows, G3(G3N1,G3N2) 4 rows
             POST FILTER : PARTSUPP.PS_SUPPKEY = $V8.S_SUPPKEY
```

```
     3  -  JOINED COLUMN : PARTSUPP.PS_SUPPKEY, SUPPLIER.S_SUPPKEY, PARTSUPP.PS_PARTKEY,
SUPPLIER.S_NAME
     4  -  HASH SHARD ( # 3 )
           READ INDEX COLUMN : PARTSUPP.PS_PARTKEY, PARTSUPP.PS_SUPPKEY
     5  -  HASH KEY : SUPPLIER.S_SUPPKEY
           RECORD COLUMN : SUPPLIER.S_NAME
           READ KEY COLUMN : SUPPLIER.S_SUPPKEY, SUPPLIER.S_NAME
             HASH FILTER : SUPPLIER.S_SUPPKEY = PARTSUPP.PS_SUPPKEY
           FETCH ONE ROW
     6  -  CLONED
           READ COLUMN : SUPPLIER.S_SUPPKEY, SUPPLIER.S_NAME
     8  -  COLUMN : {SUPPLIER.S_SUPPKEY} AS S_SUPPKEY
     9  -  TARGET : {SUPPLIER.S_SUPPKEY}
    10  -  READ COLUMN : NOTHING
<<<  end print plan
```

If a join condition including a subquery exists in an outer join, then it can not configure the generated qu ery including the outer join.
In this case, perform the join by configuring the cluster puller per each join target table as follows.

```
gSQL> \EXPLAIN PLAN
      SELECT ps_partkey, s_name
        FROM supplier
             LEFT OUTER JOIN
             partsupp
             ON ps_suppkey = ( SELECT s_suppkey FROM DUAL );
PS_PARTKEY S_NAME
---------- -------------------------
         3 Supplier#1
         5 Supplier#1
         2 Supplier#2
         1 Supplier#2
         1 Supplier#3
         4 Supplier#3
         3 Supplier#4
         5 Supplier#4
         4 Supplier#5
         2 Supplier#5
10 rows selected.
>>>  start print plan
< Execution Plan >
```

```
=============================================================================
|  IDX  |  NODE DESCRIPTION                        |              ROWS |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                        |                10 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")             |                10 |
|    2  |      NESTED JOIN (LEFT OUTER JOIN)       |                10 |
|    3  |        TABLE ACCESS ("SUPPLIER")         |                 5 |
|    4  |        PLAN BASED CLUSTER                | LOCAL/REMOTE   50 |
|    5  |          INDEX ACCESS ("PARTSUPP", ...)  | (        10)   10 |
|    6  |      SUB QUERY LIST                      |                    |
|    7  |        INLINE_VIEW ("$V7")               |                50 |
|    8  |          QUERY BLOCK ("$QB_IDX_8")       |                50 |
|    9  |            FAST DUAL ACCESS ("DUAL")     |                50 |
=============================================================================
     1  -  TARGET : PARTSUPP.PS_PARTKEY, SUPPLIER.S_NAME
     2  -  JOINED COLUMN : PARTSUPP.PS_SUPPKEY, SUPPLIER.S_SUPPKEY, PARTSUPP.PS_PARTKEY,
SUPPLIER.S_NAME
              POST ON FILTER : PARTSUPP.PS_SUPPKEY = $V7.S_SUPPKEY
     3  -  CLONED
           READ COLUMN : SUPPLIER.S_SUPPKEY, SUPPLIER.S_NAME
     4  -  SQL : SELECT /*+ INDEX( _A1, "PUBLIC"."PARTSUPP_PK_INDEX" ) */ "_A1"."PS_PARTKEY",
"_A1"."PS_SUPPKEY" FROM "PUBLIC"."PARTSUPP"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 10 rows, G2(G2N1,G2N2) 20 rows, G3(G3N1,G3N2) 20 rows
     5  -  HASH SHARD ( # 3 )
           READ INDEX COLUMN : PARTSUPP.PS_PARTKEY, PARTSUPP.PS_SUPPKEY
     7  -  COLUMN : {SUPPLIER.S_SUPPKEY} AS S_SUPPKEY
     8  -  TARGET : {SUPPLIER.S_SUPPKEY}
     9  -  READ COLUMN : NOTHING
<<<  end print plan
```

The filter including a subquery in an outer join is not included in the generated query. The cluster puller f or an outer join collects and manipulates the data, then perform filters which are not included in the gen erated query.

The following is an example of processing a filter including a subquery in an outer join.

```
gSQL> \EXPLAIN PLAN
      SELECT p_name, ps_suppkey
        FROM partsupp
             LEFT OUTER JOIN
             part
             ON p_partkey = ps_partkey
```

```
      WHERE ps_supplycost > ( SELECT p_retailprice FROM supplier WHERE s_suppkey = ps_suppkey
);
P_NAME PS_SUPPKEY
------ ----------
Part#3          4
Part#1          2
Part#5          1
3 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                              |            ROWS |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                              |               3 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                   |               3 |
|    2  |      SINGLE CLUSTER                            | LOCAL/REMOTE  3 |
|    3  |        SELECT STATEMENT                        |               2 |
|    4  |          QUERY BLOCK ("$QB_IDX_2")             |               2 |
|    5  |            HASH JOIN (LEFT OUTER JOIN)         |               2 |
|    6  |              TABLE ACCESS ("PARTSUPP" AS _A2)  |               2 |
|    7  |              HASH JOIN INSTANT                 |               2 |
|    8  |                TABLE ACCESS ("PART" AS _A1)    |               1 |
|    9  |      SUB QUERY LIST                            |                 |
|   10  |        INLINE_VIEW ("$V8")                     |              10 |
|   11  |          QUERY BLOCK ("$QB_IDX_8")             |              10 |
|   12  |            INDEX ACCESS ("SUPPLIER", ...)      | (     10)    10 |
================================================================================
     1  -  TARGET : PART.P_NAME, PARTSUPP.PS_SUPPKEY
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE USE_HASH_IN( _A1, 500 ) FULL( _A2 ) FULL( _A1 )
*/ "_A2"."PS_SUPPLYCOST", "_A2"."PS_SUPPKEY", "_A1"."P_RETAILPRICE", "_A1"."P_NAME" FROM (
"PUBLIC"."PARTSUPP"@"G1N1"¦"G1N2"¦"G2N1"¦"G2N2"¦"G3N1"¦"G3N2" AS "_A2" LEFT OUTER JOIN
"PUBLIC"."PART"@"G1N1"¦"G1N2"¦"G2N1"¦"G2N2"¦"G3N1"¦"G3N2" AS "_A1" ON "_A1"."P_PARTKEY" =
"_A2"."PS_PARTKEY") ALIAS "_A3"
           TARGET DOMAIN : G1(G1N1,G1N2) 2 rows, G2(G2N1,G2N2) 4 rows, G3(G3N1,G3N2) 4 rows
             POST FILTER : PARTSUPP.PS_SUPPLYCOST > $V8.P_RETAILPRICE
     4  -  TARGET : _A2.PS_SUPPLYCOST, _A2.PS_SUPPKEY, _A1.P_RETAILPRICE, _A1.P_NAME
     5  -  JOINED COLUMN : _A2.PS_SUPPLYCOST, _A2.PS_SUPPKEY, _A1.P_RETAILPRICE, _A1.P_NAME
     6  -  HASH SHARD ( # 3 )
           READ COLUMN : _A2.PS_PARTKEY, _A2.PS_SUPPKEY, _A2.PS_SUPPLYCOST
     7  -  HASH KEY : _A1.P_PARTKEY
           RECORD COLUMN : _A1.P_RETAILPRICE, _A1.P_NAME
```

```
           READ KEY COLUMN : _A1.P_PARTKEY, _A1.P_RETAILPRICE, _A1.P_NAME
             HASH FILTER : _A1.P_PARTKEY = _A2.PS_PARTKEY
           FETCH ONE ROW
     8  -  HASH SHARD ( # 3 )
           READ COLUMN : _A1.P_PARTKEY, _A1.P_NAME, _A1.P_RETAILPRICE
    10  -  COLUMN : {PART.P_RETAILPRICE} AS P_RETAILPRICE
    11  -  TARGET : {PART.P_RETAILPRICE}
    12  -  CLONED
           READ INDEX COLUMN : SUPPLIER.S_SUPPKEY
             MIN RANGE : SUPPLIER.S_SUPPKEY = {PARTSUPP.PS_SUPPKEY}
             MAX RANGE : SUPPLIER.S_SUPPKEY = {PARTSUPP.PS_SUPPKEY}
           FETCH ONE ROW
<<<  end print plan
```

When a subquery specified in where statement is converted into a semi join by performing <subquery unnest>, then the generated query to process the semi join includes the semi join statement. The total sum of generated query results from multiple groups becomes the duplicate semi join result. Then, it manipulates the data in distinct key group method to delete the duplicate semi join results. For more information about distinct key group, refer to **Generated Query for Distinct Key Group**.

The following is an example of processing the distinct key group for the semi join.

```
gSQL> \EXPLAIN PLAN
      SELECT s_name
        FROM supplier
       WHERE s_suppkey IN ( SELECT /*+ REMOTE_UNNEST */ ps_suppkey FROM partsupp );
S_NAME
-------------------------
Supplier#1
Supplier#2
Supplier#3
Supplier#4
Supplier#5
5 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
|IDX|  NODE DESCRIPTION                                |           ROWS |
--------------------------------------------------------------------------------
| 0 |SELECT STATEMENT                                  |              5 |
| 1 |  QUERY BLOCK ("$QB_IDX_2")                       |              5 |
| 2 |    MULTIPLE CLUSTER                              | LOCAL/REMOTE 5 |
```

```
| 3 |      SELECT STATEMENT                                |             2 |
| 4 |        QUERY BLOCK ("$QB_IDX_2")                     |             2 |
| 5 |          SORT INSTANT                                |             2 |
| 6 |            HASH JOIN (INVERTED SEMI)                 |             2 |
| 7 |              INDEX ACCESS ("PARTSUPP" AS _A2, ...) | (    2)     2 |
| 8 |              HASH JOIN INSTANT                       |             2 |
| 9 |                TABLE ACCESS ("SUPPLIER" AS _A1)      |             5 |
=========================================================================
     1  -  TARGET : SUPPLIER.S_NAME
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE USE_HASH_IN( _A1, 5 ) INDEX( _A2,
"PUBLIC"."PARTSUPP_PK_INDEX" ) FULL( _A1 ) */ "_A1"."S_SUPPKEY", "_A1"."S_NAME" FROM (
"PUBLIC"."SUPPLIER"@"G1N1"¦"G1N2"¦"G2N1"¦"G2N2"¦"G3N1"¦"G3N2" AS "_A1" SEMI JOIN
"PUBLIC"."PARTSUPP"@"G1N1"¦"G1N2"¦"G2N1"¦"G2N2"¦"G3N1"¦"G3N2" AS "_A2" ON "_A1"."S_SUPPKEY" =
"_A2"."PS_SUPPKEY") ALIAS "_A3" ORDER BY "_A1"."S_SUPPKEY" ASC NULLS LAST
           TARGET DOMAIN : G1(G1N1,G1N2) 2 rows, G2(G2N1,G2N2) 4 rows, G3(G3N1,G3N2) 3 rows
           DISTINCT KEY GROUP
             KEY GROUP : SUPPLIER.S_SUPPKEY
     4  -  TARGET : _A1.S_SUPPKEY, _A1.S_NAME
     5  -  SORT KEY : "_A1.S_SUPPKEY ASC NULLS LAST"
           RECORD COLUMN : _A1.S_NAME
           READ KEY COLUMN : _A1.S_SUPPKEY
           READ RECORD COLUMN : _A1.S_NAME
     6  -  JOINED COLUMN : _A1.S_SUPPKEY, _A1.S_NAME
     7  -  HASH SHARD ( # 3 )
           READ INDEX COLUMN : _A2.PS_SUPPKEY
     8  -  HASH KEY : _A1.S_SUPPKEY
           RECORD COLUMN : _A1.S_NAME
           READ KEY COLUMN : _A1.S_SUPPKEY, _A1.S_NAME
             HASH FILTER : _A1.S_SUPPKEY = _A2.PS_SUPPKEY
     9  -  CLONED
           READ COLUMN : _A1.S_SUPPKEY, _A1.S_NAME
<<<  end print plan
```

When a subquery specified in where statement is converted into an anti-semi join by performing <subquery unnest>, then the generated query to process the anti-semi join includes the anti-semi join statement. The total sum of generated query results from multiple groups becomes the duplicate anti-semi join result. Then, it manipulates the data in distinct key group method to delete the duplicate anti-semi join results. For more information about intersect key group, refer to **Generated Query for Intersect Key Group**.

The following is an example of processing the intersect key group for the anti-semi join.

```
gSQL> \EXPLAIN PLAN
      SELECT s_name
        FROM supplier
       WHERE s_suppkey NOT IN ( SELECT /*+ REMOTE_UNNEST */ ps_suppkey FROM partsupp WHERE
ps_supplycost > 900 );
S_NAME
-------------------------
Supplier#3
Supplier#5
2 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
¦IDX¦  NODE DESCRIPTION                                ¦           ROWS ¦
--------------------------------------------------------------------------------
¦ 0 ¦  SELECT STATEMENT                                ¦              2 ¦
¦ 1 ¦    QUERY BLOCK ("$QB_IDX_2")                     ¦              2 ¦
¦ 2 ¦      MULTIPLE CLUSTER                            ¦ LOCAL/REMOTE 2 ¦
¦ 3 ¦        SELECT STATEMENT                          ¦              4 ¦
¦ 4 ¦          QUERY BLOCK ("$QB_IDX_2")               ¦              4 ¦
¦ 5 ¦            SORT INSTANT                          ¦              4 ¦
¦ 6 ¦              HASH JOIN (ANTI SEMI)               ¦              4 ¦
¦ 7 ¦                TABLE ACCESS ("SUPPLIER" AS _A2)  ¦              5 ¦
¦ 8 ¦                HASH JOIN INSTANT (UNIQUE)        ¦              4 ¦
¦ 9 ¦                  TABLE ACCESS ("PARTSUPP" AS _A1)¦              1 ¦
================================================================================
     1  -  TARGET : SUPPLIER.S_NAME
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE USE_HASH_IN( _A1, 10 ) FULL( _A2 ) FULL( _A1 )
*/ "_A2"."S_SUPPKEY", "_A2"."S_NAME" FROM (
"PUBLIC"."SUPPLIER"@"G1N1"¦"G1N2"¦"G2N1"¦"G2N2"¦"G3N1"¦"G3N2" AS "_A2" ANTI SEMI JOIN
"PUBLIC"."PARTSUPP"@"G1N1"¦"G1N2"¦"G2N1"¦"G2N2"¦"G3N1"¦"G3N2" AS "_A1" ON "_A1"."PS_SUPPKEY" =
"_A2"."S_SUPPKEY" AND "_A1"."PS_SUPPLYCOST" > :_V0) ALIAS "_A3" ORDER BY "_A2"."S_SUPPKEY" ASC
NULLS LAST
           TARGET DOMAIN : G1(G1N1,G1N2) 4 rows, G2(G2N1,G2N2) 4 rows, G3(G3N1,G3N2) 4 rows
           INTERSECT KEY GROUP
             KEY GROUP : SUPPLIER.S_SUPPKEY
     4  -  TARGET : _A2.S_SUPPKEY, _A2.S_NAME
     5  -  SORT KEY : "_A2.S_SUPPKEY ASC NULLS LAST"
           RECORD COLUMN : _A2.S_NAME
           READ KEY COLUMN : _A2.S_SUPPKEY
           READ RECORD COLUMN : _A2.S_NAME
```

```
    6  -  JOINED COLUMN : _A2.S_SUPPKEY, _A2.S_NAME
    7  -  CLONED
          READ COLUMN : _A2.S_SUPPKEY, _A2.S_NAME
    8  -  HASH KEY : _A1.PS_SUPPKEY
          READ KEY COLUMN : _A1.PS_SUPPKEY
            HASH FILTER : _A1.PS_SUPPKEY = _A2.S_SUPPKEY
          FETCH ONE ROW
    9  -  HASH SHARD ( # 3 )
          READ COLUMN : _A1.PS_SUPPKEY, _A1.PS_SUPPLYCOST
            PHYSICAL FILTER : _A1.PS_SUPPLYCOST > :_V0
<<<  end print plan
```

WHERE Statement

Filter configured on plan node are classified as follows.

- Constant filter: This filter is processed by being made into a constant in plan node unit.
- Post filter: This filter includes a subquery or it consists of non-deterministic expressions.
- Filter: This filter is the rest of what is not classified as a constant filter or a post filter.

The generated query configures the cluster puller and the filter belonging to the subordinate node into a query. It configures the query in a bind parameter form by making a constant filter into a constant, and c onfigures the filter into a query without modification. However, it does not configure the generated quer y for a post filter.

The following is an example of the generated query including a constant filter.

```
gSQL> \VAR v1 INTEGER
gSQL> \EXEC :v1 := 1
gSQL> \EXPLAIN PLAN
      SELECT p_name, p_brand, p_type, cluster_group_id
        FROM part
       WHERE :v1 = 1;

P_NAME P_BRAND    P_TYPE CLUSTER_GROUP_ID
------ ---------- ------ ----------------
Part#3 Brand#2    STEEL                 1
Part#2 Brand#1    NICKEL                2
Part#5 Brand#3    STEEL                 2
Part#1 Brand#1    COPPER                3
Part#4 Brand#3    NICKEL                3
5 rows selected.
>>>  start print plan
```

```
< Execution Plan >
=================================================================
|  IDX  |  NODE DESCRIPTION                   |              ROWS |
-----------------------------------------------------------------
|    0  |  SELECT STATEMENT                   |                 5 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")        |                 5 |
|    2  |      PLAN BASED CLUSTER             | LOCAL/REMOTE    5 |
|    3  |        TABLE ACCESS ("PART")        |                 1 |
=================================================================
     1  -  TARGET : PART.P_NAME, PART.P_BRAND, PART.P_TYPE, PART.CLUSTER_GROUP_ID
     2  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."CLUSTER_GROUP_ID", "_A1"."P_NAME",
"_A1"."P_BRAND", "_A1"."P_TYPE" FROM "PUBLIC"."PART"@LOCAL AS "_A1" WHERE :_V0
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 2 rows, G3(G3N1,G3N2) 2 rows
             CONSTANT FILTER : :V1 = 1
     3  -  HASH SHARD ( # 3 )
           READ COLUMN : PART.P_NAME, PART.P_BRAND, PART.P_TYPE
             CONSTANT FILTER : :V1 = 1
<<<  end print plan
```

The following is an example of the generated query including a subordinate filter of the cluster puller.

```
gSQL> \EXPLAIN PLAN
      SELECT p_name, p_brand, p_type, cluster_group_id
        FROM part
       WHERE p_partkey = 1;
P_NAME P_BRAND    P_TYPE CLUSTER_GROUP_ID
------ ---------- ------ ----------------
Part#1 Brand#1    COPPER                3
1 row selected.
>>>  start print plan
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                             |           ROWS |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                             |              1 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                  |              1 |
|    2  |      PLAN BASED CLUSTER                       | REMOTE ONLY  1 |
|    3  |        INDEX ACCESS ("PART", "PART_PK_INDEX") | (     0)     0 |
=============================================================================
     1  -  TARGET : PART.P_NAME, PART.P_BRAND, PART.P_TYPE, PART.CLUSTER_GROUP_ID
     2  -  SQL : SELECT /*+ INDEX( _A1, "PUBLIC"."PART_PK_INDEX" ) */
```

```
"_A1"."CLUSTER_GROUP_ID", "_A1"."P_NAME", "_A1"."P_BRAND", "_A1"."P_TYPE" FROM
"PUBLIC"."PART"@LOCAL AS "_A1" WHERE "_A1"."P_PARTKEY" = :_V0
           TARGET DOMAIN : G3(G3N1,G3N2) 1 rows
     3  -  HASH SHARD ( # 3 )
           READ INDEX COLUMN : PART.P_PARTKEY
           READ TABLE COLUMN : PART.P_NAME, PART.P_BRAND, PART.P_TYPE
             MIN RANGE : PART.P_PARTKEY = 1
             MAX RANGE : PART.P_PARTKEY = 1
           FETCH ONE ROW
<<<  end print plan
```

The following is an example of the generated query of when the cluster puller has a post filter.

```
gSQL> \EXPLAIN PLAN
      SELECT p_name, p_brand, p_type, cluster_group_id
        FROM part
       WHERE p_name = 'Part#5' AND p_partkey IN ( SELECT /*+ NO_UNNEST */ p_partkey FROM DUAL
);
P_NAME P_BRAND    P_TYPE CLUSTER_GROUP_ID
------ ---------- ------ ----------------
Part#5 Brand#3    STEEL                 2
1 row selected.
>>>  start print plan
< Execution Plan >
=========================================================================
|  IDX  |  NODE DESCRIPTION                     |             ROWS |
-------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                     |                1 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")          |                1 |
|    2  |      PLAN BASED CLUSTER               | LOCAL/REMOTE   1 |
|    3  |        TABLE ACCESS ("PART")          |                0 |
|    4  |      SUB QUERY LIST                   |                  |
|    5  |        INLINE_VIEW ("$V5")            |                1 |
|    6  |          QUERY BLOCK ("$QB_IDX_6")    |                1 |
|    7  |            FAST DUAL ACCESS ("DUAL")  |                1 |
=========================================================================
     1  -  TARGET : PART.P_NAME, PART.P_BRAND, PART.P_TYPE, PART.CLUSTER_GROUP_ID
     2  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."CLUSTER_GROUP_ID", "_A1"."P_PARTKEY",
"_A1"."P_NAME", "_A1"."P_BRAND", "_A1"."P_TYPE" FROM "PUBLIC"."PART"@LOCAL AS "_A1" WHERE
"_A1"."P_NAME" = :_V0
           TARGET DOMAIN : G1(G1N1,G1N2) 0 rows, G2(G2N1,G2N2) 1 rows, G3(G3N1,G3N2) 0 rows
```

```
            POST FILTER : ( PART.P_PARTKEY ) IN ( $V5.P_PARTKEY )
     3  -  HASH SHARD ( # 3 )
           READ COLUMN : PART.P_PARTKEY, PART.P_NAME, PART.P_BRAND, PART.P_TYPE
             PHYSICAL FILTER : PART.P_NAME = 'Part#5'
     5  -  COLUMN : {PART.P_PARTKEY} AS P_PARTKEY
     6  -  TARGET : {PART.P_PARTKEY}
     7  -  READ COLUMN : NOTHING
<<<  end print plan
```

Using ROWNUM

The generated query can not include a non-deterministic statement, so it can not include rownum. When rownum is used, it configures COUNT plan node. In conclusion, the cluster puller can not be configured o n the top of COUNT plan.

The following is an example of the generated query of when rownum is used.

```
gSQL> \EXPLAIN PLAN
      SELECT rownum, p_name, p_brand, p_type, cluster_group_id
        FROM part;
ROWNUM P_NAME P_BRAND    P_TYPE CLUSTER_GROUP_ID
------ ------ ---------- ------ ----------------
     1 Part#3 Brand#2    STEEL                 1
     2 Part#1 Brand#1    COPPER                3
     3 Part#4 Brand#3    NICKEL                3
     4 Part#2 Brand#1    NICKEL                2
     5 Part#5 Brand#3    STEEL                 2
5 rows selected.
>>>  start print plan
< Execution Plan >
=================================================================
|  IDX  |  NODE DESCRIPTION                |              ROWS |
-----------------------------------------------------------------
|    0  |  SELECT STATEMENT                |                 5 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")     |                 5 |
|    2  |      COUNT                       |                 5 |
|    3  |        PLAN BASED CLUSTER        | LOCAL/REMOTE    5 |
|    4  |          TABLE ACCESS ("PART")   |                 1 |
=================================================================
     1  -  TARGET : ROWNUM, PART.P_NAME, PART.P_BRAND, PART.P_TYPE, PART.CLUSTER_GROUP_ID
     3  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."CLUSTER_GROUP_ID", "_A1"."P_NAME",
"_A1"."P_BRAND", "_A1"."P_TYPE" FROM "PUBLIC"."PART"@LOCAL AS "_A1"
```

```
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 2 rows, G3(G3N1,G3N2) 2 rows
     4  -  HASH SHARD ( # 3 )
           READ COLUMN : PART.P_NAME, PART.P_BRAND, PART.P_TYPE
<<<  end print plan
```

The following is an example of the generated query of when rownum filter is used.

```
gSQL> \EXPLAIN PLAN
      SELECT p_name, p_brand, p_type, cluster_group_id
        FROM part
       WHERE rownum < 3;
P_NAME P_BRAND    P_TYPE CLUSTER_GROUP_ID
------ ---------- ------ ----------------
Part#3 Brand#2    STEEL                1
Part#1 Brand#1    COPPER               3
2 rows selected.
>>>  start print plan
< Execution Plan >
=====================================================================
|  IDX  |  NODE DESCRIPTION                |                ROWS |
---------------------------------------------------------------------
|    0  |  SELECT STATEMENT                |                   2 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")     |                   2 |
|    2  |      COUNT                       |                   2 |
|    3  |        PLAN BASED CLUSTER        | LOCAL/REMOTE      3 |
|    4  |          TABLE ACCESS ("PART")   |                   1 |
=====================================================================
     1  -  TARGET : PART.P_NAME, PART.P_BRAND, PART.P_TYPE, PART.CLUSTER_GROUP_ID
     2  -  STOP KEY FILTER : ROWNUM < 3
     3  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."CLUSTER_GROUP_ID", "_A1"."P_NAME",
"_A1"."P_BRAND", "_A1"."P_TYPE" FROM "PUBLIC"."PART"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 0 rows, G3(G3N1,G3N2) 2 rows
     4  -  HASH SHARD ( # 3 )
           READ COLUMN : PART.P_NAME, PART.P_BRAND, PART.P_TYPE
<<<  end print plan
```

GROUP BY Statement

The query process for the group by statement varies upon the sharding strategy which can be used by the subordinate plan node, and grouping configuration.

Table 12-12 Grouping according to the sharding strategy of subordinate node

Sharding strategy of subordinate node, and grouping	Data collection	Data manipulation	Processing *having* clause
When all sharding keys of subordinate node are included in the grouping key	It performs the generated query including *group by* statement in all groups.	no manipulation	It includes *having* clause in the generated query.
When subordinate node is a cloned node	It performs the generated query including *group by* statement in a single group.	no manipulation	It includes *having* clause in the generated query.
When it is unable to use the sharding strategy of the subordinate node	It performs the generated query excluding *group by* statement in all groups.	grouping	It applies *having* clause after manipulating the data.

If having clause includes the non-deterministic information, except when the subordinate node is a cloned node, then the generated query including grouping can not be configured. In conclusion, the cluster puller plan node is configured below *group by*.

The following is an example of grouping when all sharding keys on the subordinate node are included in the grouping key.

```
gSQL> \EXPLAIN PLAN
      SELECT p_partkey
        FROM part
       GROUP BY p_partkey
      HAVING SUM( p_size ) > 0;
P_PARTKEY
---------
        3
        1
        4
        2
        5
5 rows selected.
>>>  start print plan
< Execution Plan >
=========================================================================
|  IDX  |  NODE DESCRIPTION                  |                ROWS |
-------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                  |                   5 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")       |                   5 |
|    2  |      PLAN BASED CLUSTER            | LOCAL/REMOTE      5 |
```

```
|    3  |        GROUP HASH INSTANT           |                  1 |
|    4  |          TABLE ACCESS ("PART")      |                  1 |
=========================================================================
     1  -  TARGET : PART.P_PARTKEY
     2  -  SQL : SELECT /*+ USE_GROUP_HASH(500) FULL( _A1 ) */ "_A1"."P_PARTKEY" FROM
"PUBLIC"."PART"@LOCAL AS "_A1" GROUP BY "_A1"."P_PARTKEY" HAVING SUM( "_A1"."P_SIZE" ) > :_V0
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 2 rows, G3(G3N1,G3N2) 2 rows
     3  -  GROUP KEY : PART.P_PARTKEY
           RECORD COLUMN : SUM( PART.P_SIZE )
           READ KEY COLUMN : PART.P_PARTKEY
           READ RECORD COLUMN : SUM( PART.P_SIZE )
             PHYSICAL FILTER : SUM( PART.P_SIZE ) > 0
     4  -  HASH SHARD ( # 3 )
           READ COLUMN : PART.P_PARTKEY, PART.P_SIZE
<<<  end print plan
```

The following is an example of grouping when the subordinate node is a cloned node.

```
gSQL> \EXPLAIN PLAN
      SELECT s_nationkey
        FROM supplier
       GROUP BY s_nationkey
      HAVING COUNT( DISTINCT s_name ) > 0;
S_NATIONKEY
---------------
CANADA
UNITED STATES
GERMANY
KOREA
FRANCE
5 rows selected.
>>>  start print plan
< Execution Plan >
=========================================================================
|  IDX  |  NODE DESCRIPTION                          |             ROWS |
-------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                          |                5 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")               |                5 |
|    2  |      GROUP HASH INSTANT                    |                5 |
|    3  |        TABLE ACCESS ("SUPPLIER")           |                5 |
=========================================================================
```

```
     1  -  TARGET : SUPPLIER.S_NATIONKEY
     2  -  GROUP KEY : SUPPLIER.S_NATIONKEY
           RECORD COLUMN : COUNT( DISTINCT SUPPLIER.S_NAME )
           READ KEY COLUMN : SUPPLIER.S_NATIONKEY
           READ RECORD COLUMN : COUNT( DISTINCT SUPPLIER.S_NAME )
             PHYSICAL FILTER : COUNT( DISTINCT SUPPLIER.S_NAME ) > 0
     3  -  CLONED
           READ COLUMN : SUPPLIER.S_NAME, SUPPLIER.S_NATIONKEY
<<<  end print plan
```

In the result above, it accessed only to a local server (a single group) when grouping the cloned table. The following is an example of grouping when it is required to access the data in the remote server.

```
gSQL> \EXPLAIN PLAN
      SELECT s_nationkey
        FROM supplier@g2
       GROUP BY s_nationkey
      HAVING COUNT( DISTINCT s_name ) > 0;
S_NATIONKEY
---------------
CANADA
UNITED STATES
GERMANY
KOREA
FRANCE
5 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                      |                 ROWS |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                      |                    5 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")           |                    5 |
|    2  |      PLAN BASED CLUSTER                | REMOTE ONLY        5 |
|    3  |        GROUP HASH INSTANT              |                    0 |
|    4  |          TABLE ACCESS ("SUPPLIER")     |                    0 |
================================================================================
     1  -  TARGET : SUPPLIER.S_NATIONKEY
     2  -  SQL : SELECT /*+ USE_GROUP_HASH(10) FULL( _A1 ) */ "_A1"."S_NATIONKEY" FROM
"PUBLIC"."SUPPLIER"@LOCAL AS "_A1" GROUP BY "_A1"."S_NATIONKEY" HAVING COUNT( DISTINCT
"_A1"."S_NAME" ) > :_V0
```

```
           TARGET DOMAIN : G2(G2N1,G2N2) 5 rows, G3(G3N1,G3N2) 0 rows
     3  -  GROUP KEY : SUPPLIER.S_NATIONKEY
           RECORD COLUMN : COUNT( DISTINCT SUPPLIER.S_NAME )
           READ KEY COLUMN : SUPPLIER.S_NATIONKEY
           READ RECORD COLUMN : COUNT( DISTINCT SUPPLIER.S_NAME )
             PHYSICAL FILTER : COUNT( DISTINCT SUPPLIER.S_NAME ) > 0
     4  -  CLONED
           READ COLUMN : SUPPLIER.S_NAME, SUPPLIER.S_NATIONKEY
<<<  end print plan
```

When sharding strategy of the subordinate node is unable to use for grouping, it performs the grouping per group through the generated query. It collects the data, then performs the grouping again to creates the grouping result. The generated query configured at that moment does not include *having* clause. *having* clause is evaluated after creating the grouping result.

The following is an example of grouping which can not use the sharding strategy of the subordinate node.

```
gSQL> \EXPLAIN PLAN
      SELECT p_type
        FROM part
       GROUP BY p_type
      HAVING SUM( p_size ) > 0;
P_TYPE
------
STEEL
NICKEL
COPPER
3 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                          |              ROWS |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                          |                 3 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")               |                 3 |
|    2  |      SINGLE CLUSTER                        | LOCAL/REMOTE    3 |
|    3  |        SELECT STATEMENT                    |                 1 |
|    4  |          QUERY BLOCK ("$QB_IDX_2")         |                 1 |
|    5  |            GROUP HASH INSTANT              |                 1 |
|    6  |              TABLE ACCESS ("PART" AS _A1)   |                 1 |
================================================================================
```

```
     1  -  TARGET : PART.P_TYPE
     2  -  SQL : SELECT /*+ USE_GROUP_HASH(10) FULL( _A1 ) */ "_A1"."P_TYPE", SUM(
"_A1"."P_SIZE" ) FROM "PUBLIC"."PART"@LOCAL AS "_A1" GROUP BY "_A1"."P_TYPE"
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 2 rows, G3(G3N1,G3N2) 2 rows
           RE-GROUPING
             GROUP KEY : PART.P_TYPE
             AGGREGATION : SUM( SUM( PART.P_SIZE ) )
             PHYSICAL FILTER : SUM( PART.P_SIZE ) > 0
     4  -  TARGET : _A1.P_TYPE, SUM( _A1.P_SIZE )
     5  -  GROUP KEY : _A1.P_TYPE
           RECORD COLUMN : SUM( _A1.P_SIZE )
           READ KEY COLUMN : _A1.P_TYPE
           READ RECORD COLUMN : SUM( _A1.P_SIZE )
     6  -  HASH SHARD ( # 3 )
           READ COLUMN : _A1.P_TYPE, _A1.P_SIZE
<<<  end print plan
```

In the result above, it collects the data by using the single cluster, then performs grouping.

When sharding strategy of the subordinate node is unable to use for grouping, it can be processed by usi ng ordering for the grouping key in the generated query. The data collected by processing the generated query from each group are sorted per grouping key by merge sorting. Then, it performs the grouping aga in based on the sorted data, and it is called as merge-grouping.

The following is an example of grouping by using the merge-grouping.

```
gSQL> \EXPLAIN PLAN
      SELECT /*+ MERGE_GROUP */ p_brand
        FROM part
       GROUP BY p_brand
      HAVING SUM( p_size ) > 0;
P_BRAND
----------
Brand#1
Brand#2
Brand#3
3 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
¦ IDX ¦ NODE DESCRIPTION                                ¦          ROWS ¦
--------------------------------------------------------------------------------
```

```
┆   0 ┆ SELECT STATEMENT                                  ┆             3 ┆
┆   1 ┆  QUERY BLOCK ("$QB_IDX_2")                        ┆             3 ┆
┆   2 ┆   MULTIPLE CLUSTER                                ┆ LOCAL/REMOTE 3 ┆
┆   3 ┆    SELECT STATEMENT                               ┆             1 ┆
┆   4 ┆     QUERY BLOCK ("$QB_IDX_2")                     ┆             1 ┆
┆   5 ┆      GROUP                                        ┆             1 ┆
┆   6 ┆       INDEX ACCESS ("PART" AS _A1, "IDX_P_BRAND") ┆ (    1)       1 ┆
=========================================================================
     1  -  TARGET : PART.P_BRAND
     2  -  SQL : SELECT /*+ INDEX( _A1, "PUBLIC"."IDX_P_BRAND" ) */ "_A1"."P_BRAND", SUM(
"_A1"."P_SIZE" ) FROM "PUBLIC"."PART"@LOCAL AS "_A1" GROUP BY "_A1"."P_BRAND" ORDER BY
"_A1"."P_BRAND" ASC NULLS LAST
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 2 rows, G3(G3N1,G3N2) 2 rows
           MERGE GROUPING
             SORT KEY : PART.P_BRAND
             GROUP KEY : PART.P_BRAND
             AGGREGATION : SUM( SUM( PART.P_SIZE ) )
             LOGICAL FILTER : SUM( PART.P_SIZE ) > 0
     4  -  TARGET : _A1.P_BRAND, SUM( _A1.P_SIZE )
     5  -  GROUP KEY : _A1.P_BRAND
           RECORD COLUMN : SUM( _A1.P_SIZE )
     6  -  HASH SHARD ( # 3 )
           READ INDEX COLUMN : _A1.P_BRAND
           READ TABLE COLUMN : _A1.P_SIZE
<<<  end print plan
```

ORDER BY Statement

The cluster puller uses merge sorting to manipulate the data for *order by* statement. It configures the gen erated query including ordering, then collects the data per group. It merges the data while sorting the col lected data in an order of the ordering key. *Order by* node can not have a filter, so it transfers the merge s orting result to the superordinate node without modification.

It uses multiple clusters to perform merge sorting.

The following is an example of processing *order by* statement for the sharded table by using merge sortin g.

```
gSQL> \EXPLAIN PLAN
      SELECT p_type
        FROM part
       ORDER BY p_size;
P_TYPE
```

```
------
NICKEL
COPPER
NICKEL
STEEL
STEEL
5 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                          |             ROWS |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                          |                5 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")               |                5 |
|    2  |      MULTIPLE CLUSTER                      | LOCAL/REMOTE   5 |
|    3  |        SELECT STATEMENT                    |                1 |
|    4  |          QUERY BLOCK ("$QB_IDX_2")         |                1 |
|    5  |            SORT INSTANT                    |                1 |
|    6  |              TABLE ACCESS ("PART" AS _A1)  |                1 |
================================================================================
     1  -  TARGET : PART.P_TYPE
     2  -  SQL : SELECT /*+ USE_ORDER_SORT FULL( _A1 ) */ "_A1"."P_SIZE", "_A1"."P_TYPE" FROM
"PUBLIC"."PART"@LOCAL AS "_A1" ORDER BY "_A1"."P_SIZE" ASC NULLS LAST
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 2 rows, G3(G3N1,G3N2) 2 rows
           MERGE SORTING
             SORT KEY : PART.P_SIZE
     4  -  TARGET : _A1.P_SIZE, _A1.P_TYPE
     5  -  SORT KEY : "_A1.P_SIZE ASC NULLS LAST"
           RECORD COLUMN : _A1.P_TYPE
           READ KEY COLUMN : _A1.P_SIZE
           READ RECORD COLUMN : _A1.P_TYPE
     6  -  HASH SHARD ( # 3 )
           READ COLUMN : _A1.P_TYPE, _A1.P_SIZE
<<<  end print plan
```

If non-deterministic information is included in the ordering key, then the generated query can not include the ordering information. In this case, the cluster puller plan node is configured below the *order by* node.

The following is an example of an ordering including the non-deterministic information.

```
gSQL> \EXPLAIN PLAN
      SELECT p_type
        FROM part
       ORDER BY p_size, RANDOM( 1, 1 );
P_TYPE
------
NICKEL
COPPER
NICKEL
STEEL
STEEL
5 rows selected.
>>>  start print plan
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                    |                ROWS |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                    |                   5 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")         |                   5 |
|    2  |      SORT INSTANT                    |                   5 |
|    3  |        PLAN BASED CLUSTER            | LOCAL/REMOTE      5 |
|    4  |          TABLE ACCESS ("PART")       |                   1 |
=============================================================================
     1  -  TARGET : PART.P_TYPE
     2  -  SORT KEY : "PART.P_SIZE ASC NULLS LAST", "RANDOM(1,1) ASC NULLS LAST"
           RECORD COLUMN : PART.P_TYPE
           READ RECORD COLUMN : PART.P_TYPE
     3  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."P_TYPE", "_A1"."P_SIZE" FROM
"PUBLIC"."PART"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 2 rows, G3(G3N1,G3N2) 2 rows
     4  -  HASH SHARD ( # 3 )
           READ COLUMN : PART.P_TYPE, PART.P_SIZE
<<<  end print plan
```

order by statement and *group by* statement can be processed by using a single cluster puller in the follow ing cases.

- When all ordering keys are included in the grouping key
- When the subordinate node of grouping is a cloned node
- When all sharding keys on the subordinate node of grouping are used as a grouping key

The following is an example of when all ordering keys are included in the grouping key.

```
gSQL> \EXPLAIN PLAN
      SELECT p_partkey, COUNT( p_type )
        FROM part
       GROUP BY p_partkey
       ORDER BY p_partkey;
P_PARTKEY COUNT( P_TYPE )
--------- ---------------
        1               1
        2               1
        3               1
        4               1
        5               1
5 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                          |             ROWS |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                          |                5 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")               |                5 |
|    2  |      MULTIPLE CLUSTER                      | LOCAL/REMOTE   5 |
|    3  |        SELECT STATEMENT                    |                1 |
|    4  |          QUERY BLOCK ("$QB_IDX_2")         |                1 |
|    5  |            SORT INSTANT                    |                1 |
|    6  |              GROUP HASH INSTANT            |                1 |
|    7  |                TABLE ACCESS ("PART" AS _A1) |                1 |
================================================================================
     1  -  TARGET : PART.P_PARTKEY, COUNT( PART.P_TYPE )
     2  -  SQL : SELECT /*+ USE_ORDER_SORT USE_GROUP_HASH(500) FULL( _A1 ) */
"_A1"."P_PARTKEY", COUNT( "_A1"."P_TYPE" ) FROM "PUBLIC"."PART"@LOCAL AS "_A1" GROUP BY
"_A1"."P_PARTKEY" ORDER BY "_A1"."P_PARTKEY" ASC NULLS LAST
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 2 rows, G3(G3N1,G3N2) 2 rows
           MERGE SORTING
             SORT KEY : PART.P_PARTKEY
     4  -  TARGET : _A1.P_PARTKEY, COUNT( _A1.P_TYPE )
     5  -  SORT KEY : "_A1.P_PARTKEY ASC NULLS LAST"
           RECORD COLUMN : COUNT( _A1.P_TYPE )
           READ KEY COLUMN : _A1.P_PARTKEY
           READ RECORD COLUMN : COUNT( _A1.P_TYPE )
     6  -  GROUP KEY : _A1.P_PARTKEY
           RECORD COLUMN : COUNT( _A1.P_TYPE )
```

```
           READ KEY COLUMN : _A1.P_PARTKEY
           READ RECORD COLUMN : COUNT( _A1.P_TYPE )
     7  -  HASH SHARD ( # 3 )
           READ COLUMN : _A1.P_PARTKEY, _A1.P_TYPE
<<<  end print plan
```

The following is an example of ordering and grouping by using a single cluster puller when the the subor dinate node of grouping is a cloned node.

```
gSQL> \EXPLAIN PLAN
      SELECT s_nationkey, COUNT( s_suppkey )
        FROM supplier@G2¦G3
       GROUP BY s_nationkey
       ORDER BY COUNT( s_suppkey );
S_NATIONKEY     COUNT( S_SUPPKEY )
--------------- ------------------
CANADA                          1
UNITED STATES                   1
GERMANY                         1
KOREA                           1
FRANCE                          1
5 rows selected.
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                     ¦              ROWS ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                     ¦                 5 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")         ¦                 5 ¦
¦    2  ¦      PLAN BASED CLUSTER              ¦ REMOTE ONLY     5 ¦
¦    3  ¦        SORT INSTANT                  ¦                 0 ¦
¦    4  ¦          GROUP HASH INSTANT          ¦                 0 ¦
¦    5  ¦            TABLE ACCESS ("SUPPLIER") ¦                 0 ¦
=============================================================================
     1  -  TARGET : SUPPLIER.S_NATIONKEY, COUNT( SUPPLIER.S_SUPPKEY )
     2  -  SQL : SELECT /*+ USE_ORDER_SORT USE_GROUP_HASH(10) FULL( _A1 ) */ COUNT(
"_A1"."S_SUPPKEY" ), "_A1"."S_NATIONKEY" FROM "PUBLIC"."SUPPLIER"@LOCAL AS "_A1" GROUP BY
"_A1"."S_NATIONKEY" ORDER BY COUNT( "_A1"."S_SUPPKEY" ) ASC NULLS LAST
           TARGET DOMAIN : G2(G2N1,G2N2) 5 rows, G3(G3N1,G3N2) 0 rows
     3  -  SORT KEY : "COUNT( SUPPLIER.S_SUPPKEY ) ASC NULLS LAST"
           RECORD COLUMN : SUPPLIER.S_NATIONKEY
```

```
           READ KEY COLUMN : COUNT( SUPPLIER.S_SUPPKEY )
           READ RECORD COLUMN : SUPPLIER.S_NATIONKEY
     4  -  GROUP KEY : SUPPLIER.S_NATIONKEY
           RECORD COLUMN : COUNT( SUPPLIER.S_SUPPKEY )
           READ KEY COLUMN : SUPPLIER.S_NATIONKEY
           READ RECORD COLUMN : COUNT( SUPPLIER.S_SUPPKEY )
     5  -  CLONED
           READ COLUMN : SUPPLIER.S_SUPPKEY, SUPPLIER.S_NATIONKEY
<<<  end print plan
```

The following is an example of ordering and grouping by using a single cluster puller when all sharding k eys on the subordinate node of grouping are used as a grouping key.

```
gSQL> \EXPLAIN PLAN
      SELECT p_partkey, COUNT( p_type )
        FROM part
       GROUP BY p_partkey
       ORDER BY COUNT( p_type );
P_PARTKEY COUNT( P_TYPE )
--------- ---------------
        3               1
        2               1
        5               1
        1               1
        4               1
5 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                          |            ROWS |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                          |               5 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")               |               5 |
|    2  |      MULTIPLE CLUSTER                      | LOCAL/REMOTE  5 |
|    3  |        SELECT STATEMENT                    |               1 |
|    4  |          QUERY BLOCK ("$QB_IDX_2")         |               1 |
|    5  |            SORT INSTANT                    |               1 |
|    6  |              GROUP HASH INSTANT            |               1 |
|    7  |                TABLE ACCESS ("PART" AS _A1)|               1 |
================================================================================
     1  -  TARGET : PART.P_PARTKEY, COUNT( PART.P_TYPE )
```

```
     2  -  SQL : SELECT /*+ USE_ORDER_SORT USE_GROUP_HASH(500) FULL( _A1 ) */ COUNT(
"_A1"."P_TYPE" ), "_A1"."P_PARTKEY" FROM "PUBLIC"."PART"@LOCAL AS "_A1" GROUP BY
"_A1"."P_PARTKEY" ORDER BY COUNT( "_A1"."P_TYPE" ) ASC NULLS LAST
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 2 rows, G3(G3N1,G3N2) 2 rows
           MERGE SORTING
             SORT KEY : COUNT( PART.P_TYPE )
     4  -  TARGET : COUNT( _A1.P_TYPE ), _A1.P_PARTKEY
     5  -  SORT KEY : "COUNT( _A1.P_TYPE ) ASC NULLS LAST"
           RECORD COLUMN : _A1.P_PARTKEY
           READ KEY COLUMN : COUNT( _A1.P_TYPE )
           READ RECORD COLUMN : _A1.P_PARTKEY
     6  -  GROUP KEY : _A1.P_PARTKEY
           RECORD COLUMN : COUNT( _A1.P_TYPE )
           READ KEY COLUMN : _A1.P_PARTKEY
           READ RECORD COLUMN : COUNT( _A1.P_TYPE )
     7  -  HASH SHARD ( # 3 )
           READ COLUMN : _A1.P_PARTKEY, _A1.P_TYPE
<<<  end print plan
```

DISTINCT Statement

The generated query of the cluster puller to perform distinct statement includes distinct statement. When performing distinct for the sharded table, then it collects the data and configures the result through grou ping. Performing distinct for a cloned table configures the result without modifying the collected data.

The following is an example of processing distinct statement for the sharded table.

```
gSQL> \EXPLAIN PLAN
      SELECT DISTINCT p_name
        FROM part;
P_NAME
------
Part#2
Part#4
Part#3
Part#1
Part#5
5 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
¦  IDX  ¦  NODE DESCRIPTION                          ¦             ROWS ¦
```

```
-------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                          |                5 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")               |                5 |
|    2  |      SINGLE CLUSTER                        | LOCAL/REMOTE   5 |
|    3  |        SELECT STATEMENT                    |                1 |
|    4  |          QUERY BLOCK ("$QB_IDX_2")         |                1 |
|    5  |            GROUP HASH INSTANT              |                1 |
|    6  |              TABLE ACCESS ("PART" AS _A1)  |                1 |
===============================================================================
     1  -  TARGET : PART.P_NAME
     2  -  SQL : SELECT /*+ USE_DISTINCT_HASH(10) FULL( _A1 ) */ DISTINCT "_A1"."P_NAME" FROM
"PUBLIC"."PART"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 2 rows, G3(G3N1,G3N2) 2 rows
           RE-GROUPING
             GROUP KEY : PART.P_NAME
     4  -  TARGET : _A1.P_NAME
     5  -  GROUP KEY : _A1.P_NAME
           READ KEY COLUMN : _A1.P_NAME
     6  -  HASH SHARD ( # 3 )
           READ COLUMN : _A1.P_NAME
<<<  end print plan
```

The following is an example of processing distinct statement for the cloned table in a remote server.

```
gSQL> \EXPLAIN PLAN
      SELECT DISTINCT s_name, s_nationkey
        FROM supplier@G2;
S_NAME                    S_NATIONKEY
------------------------- ---------------
Supplier#1                FRANCE
Supplier#5                CANADA
Supplier#4                UNITED STATES
Supplier#3                GERMANY
Supplier#2                KOREA
5 rows selected.
>>>  start print plan
< Execution Plan >
=========================================================================
|  IDX  |  NODE DESCRIPTION                    |              ROWS |
-------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                    |                 5 |
```

```
|    1  |    QUERY BLOCK ("$QB_IDX_2")          |                5 |
|    2  |      PLAN BASED CLUSTER               | REMOTE ONLY    5 |
|    3  |        GROUP HASH INSTANT             |                0 |
|    4  |          TABLE ACCESS ("SUPPLIER")    |                0 |
=========================================================================
     1  -  TARGET : SUPPLIER.S_NAME, SUPPLIER.S_NATIONKEY
     2  -  SQL : SELECT /*+ USE_DISTINCT_HASH(100) FULL( _A1 ) */ DISTINCT "_A1"."S_NAME",
"_A1"."S_NATIONKEY" FROM "PUBLIC"."SUPPLIER"@LOCAL AS "_A1"
           TARGET DOMAIN : G2(G2N1,G2N2) 5 rows
     3  -  GROUP KEY : SUPPLIER.S_NAME, SUPPLIER.S_NATIONKEY
           READ KEY COLUMN : SUPPLIER.S_NAME, SUPPLIER.S_NATIONKEY
     4  -  CLONED
           READ COLUMN : SUPPLIER.S_NAME, SUPPLIER.S_NATIONKEY
<<<  end print plan
```

Single Row Statement

Single row query processing in cluster is classified according to whether it is a cloned table or a sharded table. The generated query for process the single row query for the cloned table includes all aggregation functions. It configures the result by collecting the data performed the generated query only in a single group.

The following is an example of a single row statement for the cloned table in a remote server.

```
gSQL> \EXPLAIN PLAN
      SELECT COUNT( DISTINCT s_name ), SUM( s_suppkey )
        FROM supplier@G2;
COUNT( DISTINCT S_NAME ) SUM( S_SUPPKEY )
------------------------ ----------------
                       5               15
1 row selected.
>>>  start print plan
< Execution Plan >
=========================================================================
|  IDX  |  NODE DESCRIPTION                     |               ROWS |
-------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                     |                  1 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")          |                  1 |
|    2  |      PLAN BASED CLUSTER               | REMOTE ONLY      1 |
|    3  |        AGGREGATION BY HASH            |                  0 |
|    4  |          TABLE ACCESS ("SUPPLIER")    |                  0 |
=========================================================================
```

```
     1  -  TARGET : COUNT( DISTINCT SUPPLIER.S_NAME ), SUM( SUPPLIER.S_SUPPKEY )
     2  -  SQL : SELECT /*+ FULL( _A1 ) */ COUNT( DISTINCT "_A1"."S_NAME" ), SUM(
"_A1"."S_SUPPKEY" ) FROM "PUBLIC"."SUPPLIER"@LOCAL AS "_A1"
           TARGET DOMAIN : G2(G2N1,G2N2) 1 rows
     3  -  AGGREGATION : SUM( SUPPLIER.S_SUPPKEY )
           DISTINCT AGGREGATION : COUNT( DISTINCT SUPPLIER.S_NAME )
     4  -  CLONED
           READ COLUMN : SUPPLIER.S_SUPPKEY, SUPPLIER.S_NAME
<<<  end print plan
```

Single row query processing in a sharded table is classified according to whether the aggregation functio n includes distinct or not.

When one or more aggregation function including distinct exists

- The cluster puller plan node is configured below the plan for a single row processing.
- The generated query does not include an aggregation function.

The following is an example of processing a single row for a sharded table which has an aggregation incl uding distinct.

```
gSQL> \EXPLAIN PLAN
      SELECT COUNT( DISTINCT p_name ), SUM( p_size )
        FROM part;
COUNT( DISTINCT P_NAME ) SUM( P_SIZE )
------------------------ -------------
                       5            58
1 row selected.
>>>  start print plan
< Execution Plan >
=====================================================================
|  IDX  |  NODE DESCRIPTION                  |               ROWS |
---------------------------------------------------------------------
|    0  |  SELECT STATEMENT                  |                  1 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")       |                  1 |
|    2  |      AGGREGATION BY HASH           |                  1 |
|    3  |        PLAN BASED CLUSTER          | LOCAL/REMOTE     5 |
|    4  |          TABLE ACCESS ("PART")     |                  1 |
=====================================================================
     1  -  TARGET : COUNT( DISTINCT PART.P_NAME ), SUM( PART.P_SIZE )
     2  -  AGGREGATION : SUM( PART.P_SIZE )
           DISTINCT AGGREGATION : COUNT( DISTINCT PART.P_NAME )
```

```
     3  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."P_NAME", "_A1"."P_SIZE" FROM
"PUBLIC"."PART"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 2 rows, G3(G3N1,G3N2) 2 rows
     4  -  HASH SHARD ( # 3 )
           READ COLUMN : PART.P_NAME, PART.P_SIZE
<<<  end print plan
```

When all aggregations do not have distinct

- The plan to process a single row is not configured.
- The generated query includes an aggregation function.
- It configures the result by performing an aggregation after collecting the data.

The following is an example of processing a single row for a sharded table which does not have an aggre gation including distinct.

```
gSQL> \EXPLAIN PLAN
      SELECT COUNT( p_name ), SUM( p_size )
        FROM part;
COUNT( P_NAME ) SUM( P_SIZE )
--------------- -------------
              5            58
1 row selected.
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                          |               ROWS |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                          |                  1 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")               |                  1 |
|    2  |      SINGLE CLUSTER                        | LOCAL/REMOTE     1 |
|    3  |        SELECT STATEMENT                    |                  1 |
|    4  |          QUERY BLOCK ("$QB_IDX_2")         |                  1 |
|    5  |            TABLE ACCESS ("PART" AS _A1)    |                  1 |
================================================================================
     1  -  TARGET : COUNT( PART.P_NAME ), SUM( PART.P_SIZE )
     2  -  SQL : SELECT /*+ FULL( _A1 ) */ COUNT( "_A1"."P_NAME" ), SUM( "_A1"."P_SIZE" ) FROM
"PUBLIC"."PART"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 1 rows, G3(G3N1,G3N2) 1 rows
           RE-AGGREGATION
             AGGREGATION : SUM( COUNT( PART.P_NAME ) ), SUM( SUM( PART.P_SIZE ) )
     4  -  TARGET : COUNT( _A1.P_NAME ), SUM( _A1.P_SIZE )
```

```
    5  -  HASH SHARD ( # 3 )
          READ COLUMN : _A1.P_NAME, _A1.P_SIZE
          AGGREGATION : COUNT( _A1.P_NAME ), SUM( _A1.P_SIZE )
<<<  end print plan
```

Processing DML in Cluster

In GOLDILOCKS, a user can perform DML for all cluster members configuring a cluster system.

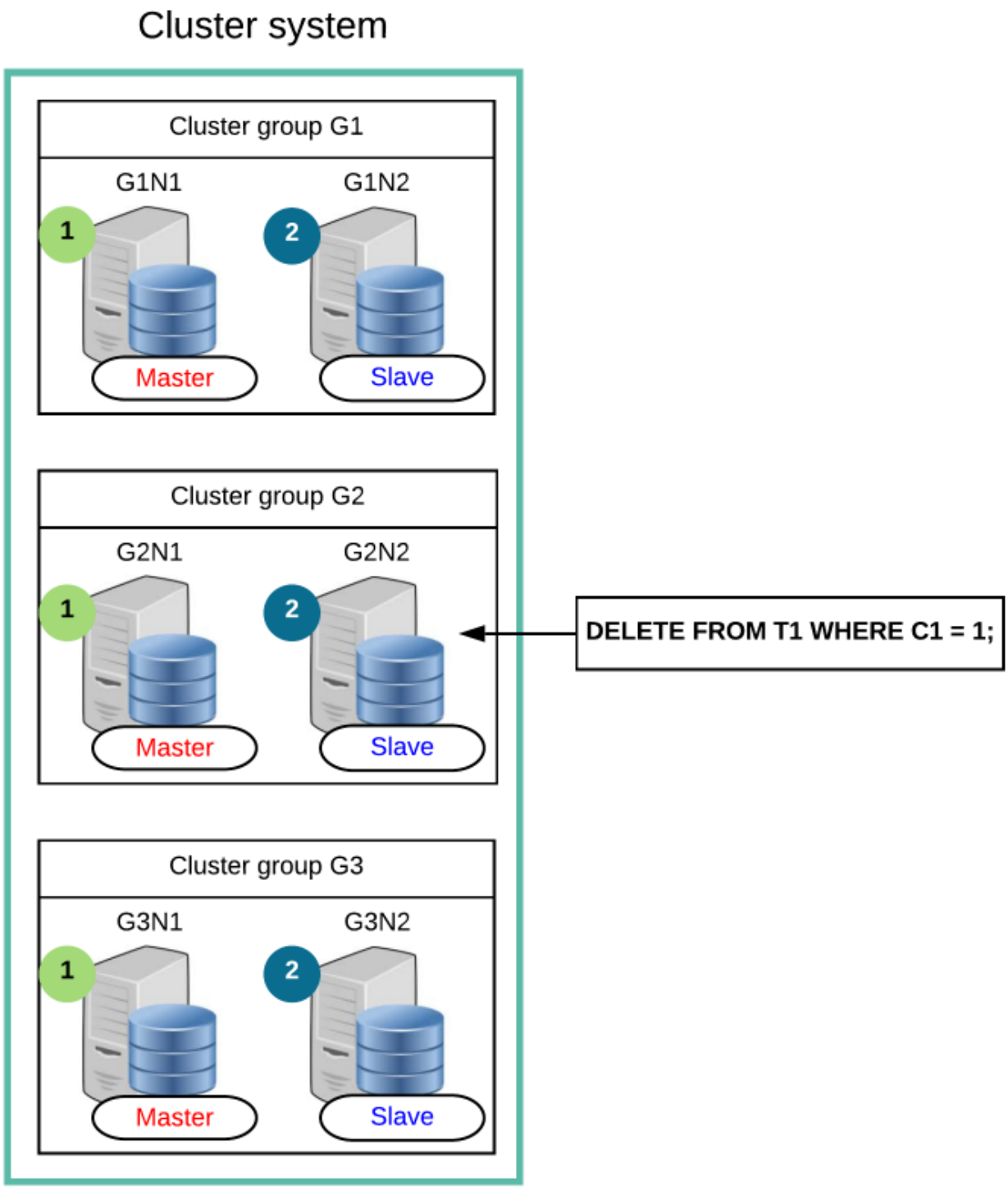
Data manipulation in cluster environment manipulates the data of the cluster member containing the sa me data replication same as well.

The following constructed table is used to describe the data manipulation.

```
CREATE TABLE t1( shard_key INTEGER, c1 INTEGER )
    SHARDING BY RANGE( shard_key )
        SHARD s1 VALUES LESS THAN ( 200 )       AT CLUSTER GROUP G1,
        SHARD s2 VALUES LESS THAN ( 400 )       AT CLUSTER GROUP G2,
        SHARD s3 VALUES LESS THAN ( MAXVALUE )  AT CLUSTER GROUP G3;
```

DML is performed as the figure below in cluster environment.

Figure 9 Processing DML in cluster



A master server and a slave server per each group are defined for the DML processing in cluster.

Selecting Master Server in Each Cluster Group

It selects a cluster member which was included first among cluster members accessible from each cluster group when manipulating data in cluster environment.

Selecting Slave Server in Each Cluster Group

It selects cluster members remained after excluding master servers from accessible cluster members per e ach cluster group when manipulating data in cluster environment.

For more information about cluster group and cluster member configuration, refer to **DBA_CLUSTER**.

```
gSQL> SELECT * FROM DBA_CLUSTER;
GROUP_ID GROUP_NAME MEMBER_ID MEMBER_NAME MEMBER_HOST MEMBER_PORT MEMBER_POSITION
-------- ---------- --------- ----------- ----------- ----------- ---------------
       1 G1                 1 G1N1        127.0.0.1         11150               0
       1 G1                 2 G1N2        127.0.0.1         11250               1
       2 G2                 3 G2N1        127.0.0.1         12150               2
       2 G2                 4 G2N2        127.0.0.1         12250               3
       3 G3                 5 G3N1        127.0.0.1         13150               4
       3 G3                 6 G3N2        127.0.0.1         13250               5
6 rows selected.
```

Performing DML

DML is sequentially performed by applying to a master server phase then applying to a slave server phase.

- Applying to a master server
  - It manipulates the data in the master server in each cluster group.
- Applying to a slave server
  - It manipulates the data in the slave server in the same way as they were applied in the master ser ver in each cluster group.

In GOLDILOCKS, the following two methods are used to manipulate data while synchronizing the master server and the slave server per each cluster group.

- **Query Based DML**
- **Global Rowid Based DML**

Query Based DML

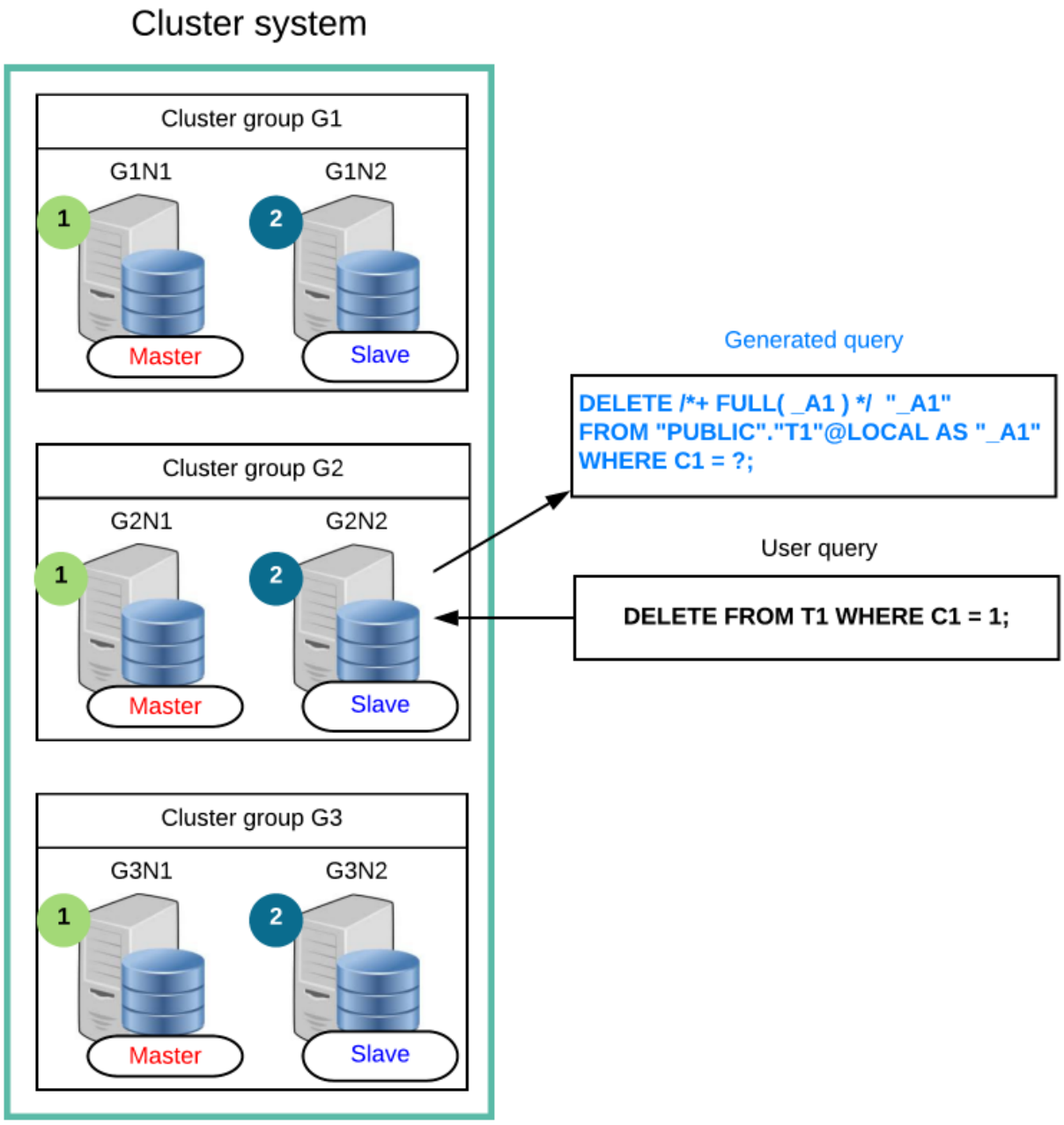
Query based DML is a method to manipulate records of each server by using the generated query. The generated query is internally generated by a server which was given a query from a user. It is supported only if the same result is guaranteed when performing DML with a generated query from each server.

For more information, refer to **Generated Query**.

The generated query for a master server and the generated query for a slave server may be different each other depending on whether the manipulated record returns the result.

The data manipulation using the generated query is performed as the figure below.

Figure 10 Updating data using generated query (Altering the entire cluster group target)



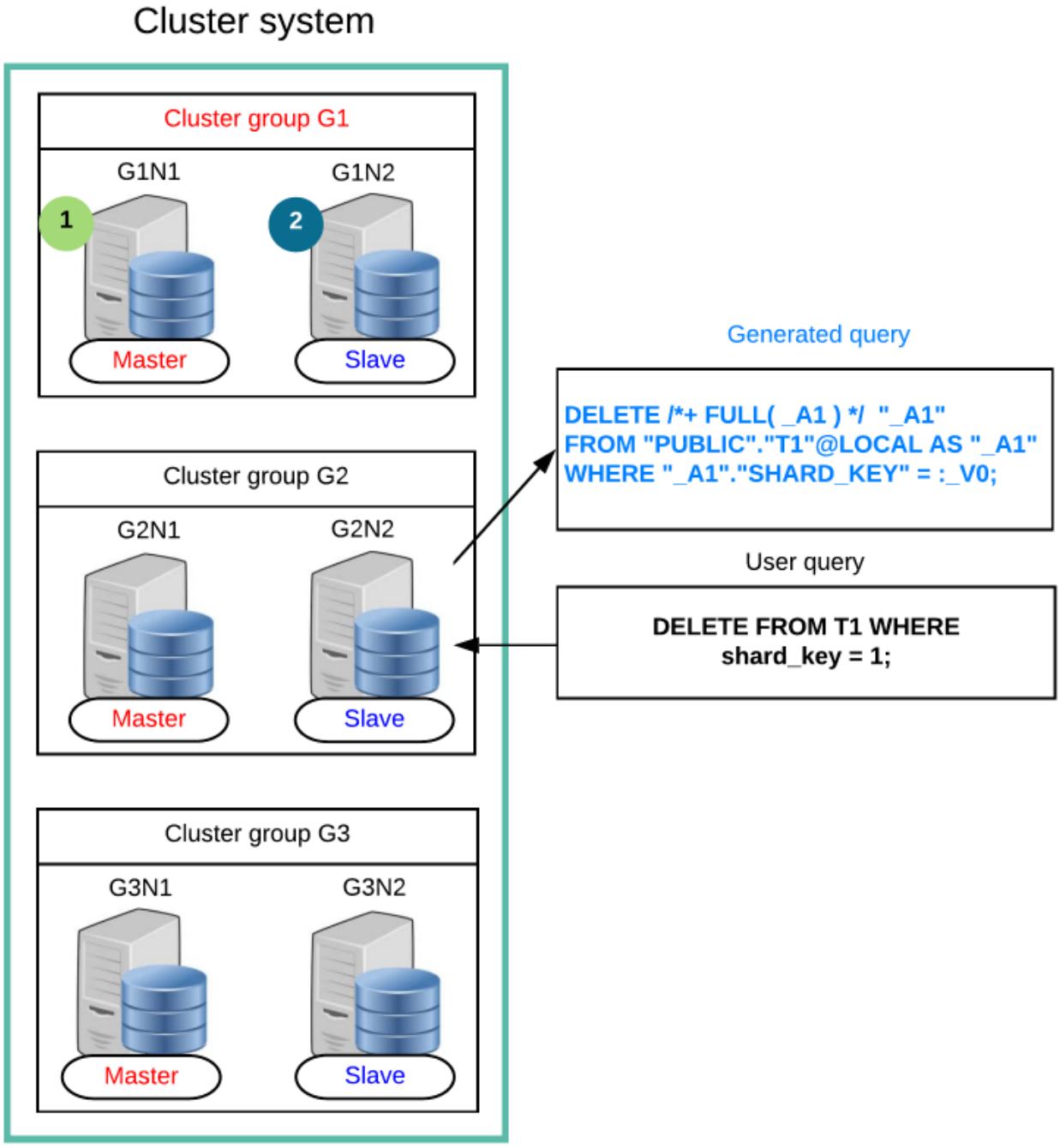
If there is not a condition to select an manipulation target cluster group in a conditional clause as given a bove, then all cluster groups become the data manipulation target of using the generated query.

The data manipulation using the generated query is performed as follows.

1. The generated query is performed on each master server
2. The generated query is performed on each slave server

If only a specific cluster group is selected as a data manipulation target by a conditional clause, then it is p erformed as follows.

Figure 11 Updating data using generated query (Altering a specific cluster group)



If the manipulation target is determined by using the search condition (shard_key = 1) as given above, it i s found that the record whose shard_key is 1 is in G1 cluster group by the sharding strategy. Therefore, o nly the record in G1 cluster group is deleted.

Manipulating the data in a specific cluster group is applied to a slave after it is applied to a master as like manipulating the data in the entire cluster group.

The plan node called as DML cluster performs the query based DML. For more information, refer to **DML Cluster**.

Data manipulation using the generated query is supported only in the following cases.

- If the generated query which guarantees the same result when performing the generated query on e ach cluster member of a cluster group can be generated
  - Refer to **Constraints of Generated Query Configuration**.
- If the data reference target server and the data manipulation target server can be restricted to a same server when configuring the generated query
  - This means that the server performing the generated query does not need to access another serv er while processing the query.

The following user queries support the data manipulation using the generated query.

- **SELECT .. FOR UPDATE**
- **SELECT .. INTO .. FOR UPDATE**
- **DELETE FROM**
- **DELETE FROM name RETURNING**
- **DELETE FROM name RETURNING .. INTO**
- **UPDATE**
- **UPDATE name RETURNING**
- **UPDATE name RETURNING .. INTO**

The data manipulation using the generated query is available even when the **Global Secondary Index** is n ot configured.

DML Cluster

DML cluster manipulates data from each server by using the generated query, and collects data when it is needed.

The following is an example of processing DELETE RETURN statement for a sharded table with the query based DML.

```
gSQL> \EXPLAIN PLAN DELETE FROM part WHERE p_partkey = 5 RETURN p_name;
P_NAME
------
```

```
Part#3
1 row deleted.
>>>  start print plan
< Execution Plan >
============================================================================
|  IDX  |  NODE DESCRIPTION                              |           ROWS |
----------------------------------------------------------------------------
|    0  |  DELETE STATEMENT ("PART")                     |              1 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                   |              0 |
|    2  |      DML CLUSTER                               | REMOTE ONLY  1 |
|    3  |        INDEX ACCESS ("PART", "PART_PK_INDEX") | (    0)      0 |
============================================================================
     1  -  TARGET : PART.P_NAME
     2  -  FETCH
           Fetch SQL : DELETE /*+ INDEX( _A1, "PUBLIC"."PART_PK_INDEX" ) */  "_A1" FROM
"PUBLIC"."PART"@LOCAL AS "_A1" WHERE "_A1"."P_PARTKEY" = :_V0 RETURN "_A1"."$PHYSICAL_ROWID",
"_A1"."P_PARTKEY", "_A1"."P_NAME", "_A1"."P_BRAND"
           Non-Fetch SQL : DELETE /*+ INDEX( _A1, "PUBLIC"."PART_PK_INDEX" ) */  "_A1" FROM
"PUBLIC"."PART"@LOCAL AS "_A1" WHERE "_A1"."P_PARTKEY" = :_V0
           TARGET DOMAIN : G2(G2N1,G2N2) 1 rows
     3  -  HASH SHARD ( # 3 )
           READ INDEX COLUMN : PART.P_PARTKEY
           READ TABLE COLUMN : PART.P_NAME, PART.P_BRAND
             MIN RANGE : PART.P_PARTKEY = 5
             MAX RANGE : PART.P_PARTKEY = 5
           FETCH ONE ROW
<<<  end print plan
```

DML cluster is used to perform DELETE statement in the query execution result. DML cluster is a plan wh ose idx of <Execution Plan> is 2 in the execution plan output above.

The detailed information about DML cluster is as follows.

- DML cluster usage type: FETCH, WITHOUT FETCH, SHARD KEY UPDATE
- Fetch SQL: The generated query for performing DML and collecting the data
- Non-fetch SQL: The generated query only for performing DML
- TARGET DOMAIN: The group and member to transfer the generated query and the number of data r eceived from the group
- Shard key update information: The generated query which divided a singe UPDATE statement into U PDATE, SELECT, and DELETE

DML cluster usage type is classified according to the user query.

- WITHOUT FETCH: DML which does not acquire the execution result data (DELETE, UPDATE)
- FETCH: DML to acquire the execution result data (SELECT FOR UPDATE, DELETE RETURN, UPDATE RETURN)
- SHARD KEY UPDATE: UPDATE which updates the sharding key (UPDATE, UPDATE RETURN)

DML Cluster (WITHOUT FETCH)

If DML cluster usage type is WITHOUT FETCH, then the detailed information about DML cluster is configured with non-fetch SQL and TARGET DOMAIN as follows.

```
gSQL> \EXPLAIN PLAN DELETE FROM supplier;
5 rows deleted.
>>>  start print plan
< Execution Plan >
=========================================================================
|  IDX  |  NODE DESCRIPTION                          |           ROWS |
-------------------------------------------------------------------------
|    0  |  DELETE STATEMENT ("SUPPLIER")             |              5 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")               |              0 |
|    2  |      DML CLUSTER                           |              5 |
|    3  |        INDEX ACCESS ("SUPPLIER", ...)      | (     5)     5 |
=========================================================================
     1  -  TARGET : NOTHING
     2  -  WITHOUT FETCH
           Non-Fetch SQL : DELETE /*+ INDEX( _A1, "PUBLIC"."SUPPLIER_PK_INDEX" ) */  "_A1"
FROM "PUBLIC"."SUPPLIER"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 5 rows, G2(G2N1,G2N2) 5 rows, G3(G3N1,G3N2) 5 rows
     3  -  CLONED
           READ INDEX COLUMN : SUPPLIER.S_SUPPKEY
<<<  end print plan
```

If sharded table data manipulation affects only a specific group as follows, then the group in which a non-fetch SQL is performed is restricted.

```
gSQL> \EXPLAIN PLAN DELETE FROM part WHERE p_partkey = 5;
1 row deleted.
>>>  start print plan
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                              |         ROWS |
-----------------------------------------------------------------------------
|    0  |  DELETE STATEMENT ("PART")                     |            1 |
```

```
┆    1  ┆    QUERY BLOCK ("$QB_IDX_2")                  ┆           0 ┆
┆    2  ┆      DML CLUSTER                              ┆           1 ┆
┆    3  ┆        INDEX ACCESS ("PART", "PART_PK_INDEX") ┆ (    0)   0 ┆
=========================================================================
     1  -  TARGET : NOTHING
     2  -  WITHOUT FETCH
           Non-Fetch SQL : DELETE /*+ INDEX( _A1, "PUBLIC"."PART_PK_INDEX" ) */  "_A1" FROM
"PUBLIC"."PART"@LOCAL AS "_A1" WHERE "_A1"."P_PARTKEY" = :_V0
           TARGET DOMAIN : G2(G2N1,G2N2) 1 rows
     3  -  HASH SHARD ( # 3 )
           READ INDEX COLUMN : PART.P_PARTKEY
           READ TABLE COLUMN : PART.P_BRAND
             MIN RANGE : PART.P_PARTKEY = 5
             MAX RANGE : PART.P_PARTKEY = 5
           FETCH ONE ROW
<<<  end print plan
```

A non-fetch SQL of DML cluster which consists of WITHOUT FETCH is performed equally in all master servers and slave servers without distinguishing a cloned table and a sharded table.

DML Cluster (FETCH)

If DML cluster usage type is FETCH, then the detailed information about DML cluster is configured with fetch SQL, non-fetch SQL and TARGET DOMAIN as follows.

```
gSQL> \EXPLAIN PLAN SELECT p_name FROM part FOR UPDATE;
P_NAME
------
Part#1
Part#4
Part#3
Part#2
Part#5
5 rows selected.
>>>  start print plan
< Execution Plan >
=================================================================
┆  IDX  ┆  NODE DESCRIPTION               ┆              ROWS ┆
-----------------------------------------------------------------
┆    0  ┆  SELECT FOR UPDATE STATEMENT    ┆                 5 ┆
┆    1  ┆    QUERY BLOCK ("$QB_IDX_2")    ┆                 0 ┆
┆    2  ┆      DML CLUSTER                ┆ REMOTE ONLY     5 ┆
```

```
¦    3  ¦        TABLE ACCESS ("PART")     ¦                 2 ¦
=============================================================
     1  -  TARGET : PART.P_NAME
     2  -  FETCH
           Fetch SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."$PHYSICAL_ROWID", "_A1"."P_NAME" FROM
"PUBLIC"."PART"@LOCAL AS "_A1" FOR UPDATE OF "_A1"."P_PARTKEY"
           Non-Fetch SQL : SELECT /*+ FULL( _A1 ) */  NULL FROM "PUBLIC"."PART"@LOCAL AS "_A1"
FOR UPDATE OF "_A1"."P_PARTKEY" WITHOUT FETCH
           TARGET DOMAIN : G1(G1N1,G1N2) 2 rows, G2(G2N1,G2N2) 1 rows, G3(G3N1,G3N2) 2 rows
     3  -  HASH SHARD ( # 3 )
           READ COLUMN : PART.P_NAME
<<<  end print plan
```

A non-fetch SQL consists of SELECT FOR UPDATE statement in the result above. The non-fetch SQL confi gured like this is performed in a local server and a remote server, but it does not collect the data through SELECT FOR UPDATE statement.

The fetch SQL of query based DML for a cloned table is performed on a member among the entire group. If a local server has a replication of a cloned table, then the fetch SQL is performed in a local server. If a lo cal server does not have a replication of a cloned table, then the fetch SQL is performed in an arbitrary ser ver. All servers having a replication of a cloned table except for the server performing fetch SQL performs non-fetch SQLs.

The fetch SQL of query based DML for a sharded table is performed on master servers of each group. All of slave servers perform non-fetch SQL.

If sharded table data manipulation affects only a specific group as follows, then the group in which a fetc h SQL and a non-fetch SQL is performed is restricted.

```
gSQL> \EXPLAIN PLAN SELECT p_name FROM part WHERE p_partkey = 5 FOR UPDATE;
P_NAME
------
Part#3
1 row selected.
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                              ¦           ROWS ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT FOR UPDATE STATEMENT                   ¦              1 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                  ¦              0 ¦
¦    2  ¦      DML CLUSTER                              ¦ REMOTE ONLY  1 ¦
¦    3  ¦        INDEX ACCESS ("PART", "PART_PK_INDEX") ¦ (    0)      0 ¦
```

```
================================================================================
     1  -  TARGET : PART.P_NAME
     2  -  FETCH
           Fetch SQL : SELECT /*+ INDEX( _A1, "PUBLIC"."PART_PK_INDEX" ) */
"_A1"."$PHYSICAL_ROWID", "_A1"."P_NAME" FROM "PUBLIC"."PART"@LOCAL AS "_A1" WHERE
"_A1"."P_PARTKEY" = :_V0 FOR UPDATE OF "_A1"."P_PARTKEY"
           Non-Fetch SQL : SELECT /*+ INDEX( _A1, "PUBLIC"."PART_PK_INDEX" ) */  NULL FROM
"PUBLIC"."PART"@LOCAL AS "_A1" WHERE "_A1"."P_PARTKEY" = :_V0 FOR UPDATE OF "_A1"."P_PARTKEY"
WITHOUT FETCH
           TARGET DOMAIN : G2(G2N1,G2N2) 1 rows
     3  -  HASH SHARD ( # 3 )
           READ INDEX COLUMN : PART.P_PARTKEY
           READ TABLE COLUMN : PART.P_NAME
             MIN RANGE : PART.P_PARTKEY = 5
             MAX RANGE : PART.P_PARTKEY = 5
           FETCH ONE ROW
<<<  end print plan
```

DML Cluster (SHARD KEY UPDATE)

UPDATE for the sharding key column is classified to the following two types according to whether the sh ard to which the record belongs is altered before and after the data manipulation.

- In-place update: The shard to which the record belongs is same before and after the data manipulati on.
- Out-place update: The shard to which the record belongs is different before and after the data manip ulation.

In-place update alters the value without moving the record. The rowid information of the record which w ere updated through in-place update is not updated.

Out-place update inserts a new record after delete the existing record. New rowid is set in the record whi ch were updated through out-place update.

SHARD KEY UPDATE configures the generated query distinguishing an in-place update and an out-place update.

The generated query for an in-place update consists of UPDATE statement. The generated query has a filt er which guarantees that the shards to which the value belongs before and after update are same.

The generated query for an out-place update separately consists of SELECT statement and DELETE statem ent. Each generated query has a filter which guarantees that the shards to which the value belongs befor e and after update are different. It configures a new record by collecting data before the update through the generated query consisting of SELECT statement. It inserts new records by using <global rowid based DML>. It deletes all previous records through DELETE statement.

If DML cluster usage type is SHARD KEY UPDATE, then the detailed information about DML cluster is con figured with UPDATE SQL for in-place update and SELECT SQL & DELETE SQL for out-place update as foll ows.

```
gSQL> \EXPLAIN PLAN UPDATE part SET p_partkey = p_partkey + 10;
5 rows updated.
>>>  start print plan
< Execution Plan >
================================================================================
¦  IDX  ¦  NODE DESCRIPTION                              ¦            ROWS ¦
--------------------------------------------------------------------------------
¦    0  ¦  UPDATE STATEMENT ("PART")                     ¦               5 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                  ¦               0 ¦
¦    2  ¦      DML CLUSTER                              ¦               5 ¦
¦    3  ¦        UPDATE STATEMENT ("PART")              ¦               0 ¦
¦    4  ¦          QUERY BLOCK ("$QB_IDX_2")            ¦               0 ¦
¦    5  ¦            DML CLUSTER                        ¦               0 ¦
¦    6  ¦              TABLE ACCESS ("PART" AS _A1)     ¦               0 ¦
¦    7  ¦          QUERY BLOCK ("$QB_IDX_6")            ¦               0 ¦
¦    8  ¦            INDEX ACCESS ("PART" AS _A1, ...)  ¦               0 ¦
¦    9  ¦        SELECT STATEMENT                       ¦               5 ¦
¦   10  ¦          QUERY BLOCK ("$QB_IDX_2")            ¦               5 ¦
¦   11  ¦            PLAN BASED CLUSTER                 ¦ LOCAL/REMOTE  5 ¦
¦   12  ¦              TABLE ACCESS ("PART" AS _A1)     ¦               2 ¦
¦   13  ¦        DELETE STATEMENT ("PART")              ¦               5 ¦
¦   14  ¦          QUERY BLOCK ("$QB_IDX_2")            ¦               0 ¦
¦   15  ¦            DML CLUSTER                        ¦               5 ¦
¦   16  ¦              TABLE ACCESS ("PART" AS _A1)     ¦               2 ¦
¦   17  ¦    QUERY BLOCK ("$QB_IDX_6")                  ¦               0 ¦
¦   18  ¦      TABLE ACCESS ("PART")                    ¦               0 ¦
================================================================================
     1  -  TARGET : NOTHING
     2  -  SHARD KEY UPDATE
           UPDATE SQL : UPDATE /*+ FULL( _A1 ) */
"PUBLIC"."PART"@"G1N1"¦"G1N2"¦"G2N1"¦"G2N2"¦"G3N1"¦"G3N2" AS "_A1" SET ( "_A1"."P_PARTKEY" ) =
( CAST( "_A1"."P_PARTKEY" + :_V0 AS NUMBER(10, 0) ) ) FROM
"PUBLIC"."PART"@"G1N1"¦"G1N2"¦"G2N1"¦"G2N2"¦"G3N1"¦"G3N2" AS "_A1" WHERE
SHARD_ID("PUBLIC"."PART", "_A1"."P_PARTKEY") = SHARD_ID("PUBLIC"."PART", CAST(
"_A1"."P_PARTKEY" + :_V0 AS NUMBER(10, 0) ))
           SELECT SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."$PHYSICAL_ROWID", "_A1"."P_PARTKEY",
"_A1"."P_NAME", "_A1"."P_BRAND", "_A1"."P_TYPE", "_A1"."P_SIZE", "_A1"."P_RETAILPRICE" FROM
```

```
"PUBLIC"."PART"@"G1N1"¦"G1N2"¦"G2N1"¦"G2N2"¦"G3N1"¦"G3N2" AS "_A1" WHERE
SHARD_ID("PUBLIC"."PART", "_A1"."P_PARTKEY") <> SHARD_ID("PUBLIC"."PART", CAST(
"_A1"."P_PARTKEY" + :_V0 AS NUMBER(10, 0) ))
          DELETE SQL : DELETE /*+ FULL( _A1 ) */  "_A1" FROM
"PUBLIC"."PART"@"G1N1"¦"G1N2"¦"G2N1"¦"G2N2"¦"G3N1"¦"G3N2" AS "_A1" WHERE
SHARD_ID("PUBLIC"."PART", "_A1"."P_PARTKEY") <> SHARD_ID("PUBLIC"."PART", CAST(
"_A1"."P_PARTKEY" + :_V0 AS NUMBER(10, 0) ))
     4  -  TARGET : NOTHING
     5  -  WITHOUT FETCH
          Non-Fetch SQL : UPDATE /*+ FULL( _A1 ) */ "PUBLIC"."PART"@LOCAL AS "_A1" SET (
"_A1"."P_PARTKEY" ) = ( CAST( "_A1"."P_PARTKEY" + :_V0 AS NUMBER(10, 0) ) ) FROM
"PUBLIC"."PART"@LOCAL AS "_A1" WHERE SHARD_ID("PUBLIC"."PART","_A1"."P_PARTKEY") =
SHARD_ID("PUBLIC"."PART",CAST( "_A1"."P_PARTKEY" + :_V1 AS NUMBER(10, 0) ))
          TARGET DOMAIN : G1(G1N1,G1N2) 0 rows, G2(G2N1,G2N2) 0 rows, G3(G3N1,G3N2) 0 rows
     6  -  HASH SHARD ( # 3 )
          READ COLUMN : _A1.P_PARTKEY
            LOGICAL FILTER : SHARD_ID( "PUBLIC"."PART",_A1.P_PARTKEY) = SHARD_ID(
"PUBLIC"."PART",CAST( _A1.P_PARTKEY + :_V0 AS NUMBER(10, 0) ))
     7  -  TARGET : NOTHING
     8  -  HASH SHARD ( # 3 )
          READ INDEX COLUMN : _A1.P_PARTKEY
    10  -  TARGET : _A1.$PHYSICAL_ROWID, _A1.P_PARTKEY, _A1.P_NAME, _A1.P_BRAND, _A1.P_TYPE,
_A1.P_SIZE, _A1.P_RETAILPRICE
    11  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."$PHYSICAL_ROWID", "_A1"."P_PARTKEY",
"_A1"."P_NAME", "_A1"."P_BRAND", "_A1"."P_TYPE", "_A1"."P_SIZE", "_A1"."P_RETAILPRICE" FROM
"PUBLIC"."PART"@LOCAL AS "_A1" WHERE SHARD_ID("PUBLIC"."PART","_A1"."P_PARTKEY") <>
SHARD_ID("PUBLIC"."PART",CAST( "_A1"."P_PARTKEY" + :_V0 AS NUMBER(10, 0) ))
          TARGET DOMAIN : G1(G1N1,G1N2) 2 rows, G2(G2N1,G2N2) 1 rows, G3(G3N1,G3N2) 2 rows
    12  -  HASH SHARD ( # 3 )
          READ COLUMN : _A1.P_PARTKEY, _A1.P_NAME, _A1.P_BRAND, _A1.P_TYPE, _A1.P_SIZE,
_A1.P_RETAILPRICE
            LOGICAL FILTER : SHARD_ID( "PUBLIC"."PART",_A1.P_PARTKEY) <> SHARD_ID(
"PUBLIC"."PART",CAST( _A1.P_PARTKEY + :_V0 AS NUMBER(10, 0) ))
    14  -  TARGET : NOTHING
    15  -  WITHOUT FETCH
          Non-Fetch SQL : DELETE /*+ FULL( _A1 ) */  "_A1" FROM "PUBLIC"."PART"@LOCAL AS
"_A1" WHERE SHARD_ID("PUBLIC"."PART","_A1"."P_PARTKEY") <> SHARD_ID("PUBLIC"."PART",CAST(
"_A1"."P_PARTKEY" + :_V0 AS NUMBER(10, 0) ))
          TARGET DOMAIN : G1(G1N1,G1N2) 2 rows, G2(G2N1,G2N2) 1 rows, G3(G3N1,G3N2) 2 rows
    16  -  HASH SHARD ( # 3 )
          READ COLUMN : _A1.P_PARTKEY, _A1.P_BRAND
```

```
            LOGICAL FILTER : SHARD_ID( "PUBLIC"."PART",_A1.P_PARTKEY) <> SHARD_ID(
"PUBLIC"."PART",CAST( _A1.P_PARTKEY + :_V0 AS NUMBER(10, 0) ))
    17  -  TARGET : NOTHING
    18  -  HASH SHARD ( # 3 )
           READ COLUMN : PART.P_PARTKEY, PART.P_NAME, PART.P_BRAND, PART.P_TYPE, PART.P_SIZE,
PART.P_RETAILPRICE
<<<  end print plan
```

Join Operation in DML Cluster

The statement including a subquery can be converted into a join operation by the query processor like as **Join including subquery**. The subquery for DML statement can also be converted to a join operation. DML cluster plan node supports the generated query including a join.

The following is an example of performing DELETE including a subquery.

```
gSQL> \EXPLAIN PLAN
      DELETE FROM part WHERE p_partkey IN ( SELECT ps_partkey FROM partsupp );
3 rows deleted.
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                          ¦            ROWS ¦
-----------------------------------------------------------------------------
¦    0  ¦  DELETE STATEMENT ("PART")                 ¦               3 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")              ¦               0 ¦
¦    2  ¦      DML CLUSTER                          ¦               3 ¦
¦    3  ¦        HASH JOIN (SEMI)                   ¦               1 ¦
¦    4  ¦          TABLE ACCESS ("PART")            ¦               2 ¦
¦    5  ¦          HASH JOIN INSTANT (UNIQUE)       ¦               1 ¦
¦    6  ¦            INDEX ACCESS ("PARTSUPP", ...) ¦ (     2)      2 ¦
=============================================================================
     1  -  TARGET : NOTHING
     2  -  WITHOUT FETCH
           Non-Fetch SQL : DELETE /*+ KEEP_JOINED_TABLE USE_HASH_IN( _A2, 10 ) FULL( _A1 )
INDEX( _A2, "PUBLIC"."PARTSUPP_PK_INDEX" ) */  "_A1" FROM (
"PUBLIC"."PART"@"G1N1"¦"G1N2"¦"G2N1"¦"G2N2"¦"G3N1"¦"G3N2" AS "_A1" SEMI JOIN
"PUBLIC"."PARTSUPP"@"G1N1"¦"G1N2"¦"G2N1"¦"G2N2"¦"G3N1"¦"G3N2" AS "_A2" ON "_A2"."PS_PARTKEY" =
"_A1"."P_PARTKEY") ALIAS "_A3"
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 1 rows, G3(G3N1,G3N2) 1 rows
     3  -  JOINED COLUMN : PART.$PHYSICAL_ROWID, PART.P_PARTKEY, PART.P_BRAND
     4  -  HASH SHARD ( # 3 )
```

```
           READ COLUMN : PART.P_PARTKEY, PART.P_BRAND
     5  -  HASH KEY : PARTSUPP.PS_PARTKEY
           READ KEY COLUMN : PARTSUPP.PS_PARTKEY
             HASH FILTER : PARTSUPP.PS_PARTKEY = PART.P_PARTKEY
           FETCH ONE ROW
     6  -  HASH SHARD ( # 3 )
           READ INDEX COLUMN : PARTSUPP.PS_PARTKEY
<<<  end print plan
```

However, DML cluster plan node does not support the manipulating the collected data. The join operatio n which required the data manipulation after collecting the data through the generated query can not co nfigure DML cluster.

The following is an example of when it converts DML including a subquery to a join operation, but can n ot configure DML cluster.

```
gSQL> \EXPLAIN PLAN
      DELETE FROM supplier WHERE s_suppkey IN ( SELECT ps_partkey FROM partsupp );
5 rows deleted.
>>>  start print plan
< Execution Plan >
================================================================================
¦IDX¦  NODE DESCRIPTION                                ¦           ROWS ¦
--------------------------------------------------------------------------------
¦ 0 ¦  DELETE STATEMENT ("SUPPLIER")                   ¦              5 ¦
¦ 1 ¦    QUERY BLOCK ("$QB_IDX_2")                     ¦              5 ¦
¦ 2 ¦      SINGLE CLUSTER                              ¦ LOCAL/REMOTE 5 ¦
¦ 3 ¦        SELECT STATEMENT                          ¦              1 ¦
¦ 4 ¦          QUERY BLOCK ("$QB_IDX_2")               ¦              1 ¦
¦ 5 ¦            NESTED JOIN (SEMI)                    ¦              1 ¦
¦ 6 ¦              INDEX ACCESS ("SUPPLIER" AS _A2, ...) ¦ (     5)     5 ¦
¦ 7 ¦              INDEX ACCESS ("PARTSUPP" AS _A1, ...) ¦ (     1)     1 ¦
================================================================================
     1  -  TARGET : NOTHING
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE USE_NL_IN( _A1 ) INDEX( _A2,
"PUBLIC"."SUPPLIER_PK_INDEX" ) INDEX( _A1, "PUBLIC"."PARTSUPP_PK_INDEX" ) */
"_A2"."$PHYSICAL_ROWID", "_A2"."S_SUPPKEY" FROM (
"PUBLIC"."SUPPLIER"@"G1N1"¦"G1N2"¦"G2N1"¦"G2N2"¦"G3N1"¦"G3N2" AS "_A2" SEMI JOIN
"PUBLIC"."PARTSUPP"@"G1N1"¦"G1N2"¦"G2N1"¦"G2N2"¦"G3N1"¦"G3N2" AS "_A1" ON "_A1"."PS_PARTKEY" =
"_A2"."S_SUPPKEY") ALIAS "_A3"
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows, G2(G2N1,G2N2) 2 rows, G3(G3N1,G3N2) 2 rows
     4  -  TARGET : _A2.$PHYSICAL_ROWID, _A2.S_SUPPKEY
```

```
    5  -  JOINED COLUMN : _A2.$PHYSICAL_ROWID, _A2.S_SUPPKEY
    6  -  CLONED
          READ INDEX COLUMN : _A2.S_SUPPKEY
    7  -  HASH SHARD ( # 3 )
          READ INDEX COLUMN : _A1.PS_PARTKEY
            MIN RANGE : _A1.PS_PARTKEY = {_A2.S_SUPPKEY}
            MAX RANGE : _A1.PS_PARTKEY = {_A2.S_SUPPKEY}
<<<  end print plan
```

Global Rowid Based DML

Global rowid based DML is a method to manipulates the same records stored in different cluster member s by using the rowid. This method is used when adding a new record or when **Query Based DML** is not av ailable.

The rowid provided in cluster environment is a logical identification information given to the record and it is a standard to determine whether the records are same. If a single record is stored in different cluster m embers, then all of them have same rowid. Rowid information is given when creating a record, or a new r owid information is given when modifying the value of the sharding key column.

For more information about the rowid, refer to **ROWID Pseudo Column** .

Data manipulation using the rowid information is divided into a rowid information collection phase and a data manipulation phase. When manipulating two or more records, a rowid information collection phase and a data manipulation phase are repeated per record.

- The rowid information collection phase
  - It collects the rowid information of records to be added or to be manipulated.
- The data manipulation phase
  - It manipulates records of a cluster group corresponding to the collected rowid.
  - It sequentially manipulates records in a slave server after manipulating records in a master server.

Data manipulation using the rowid information supports the following two methods depending on whet her to use the **Global Secondary Index**.

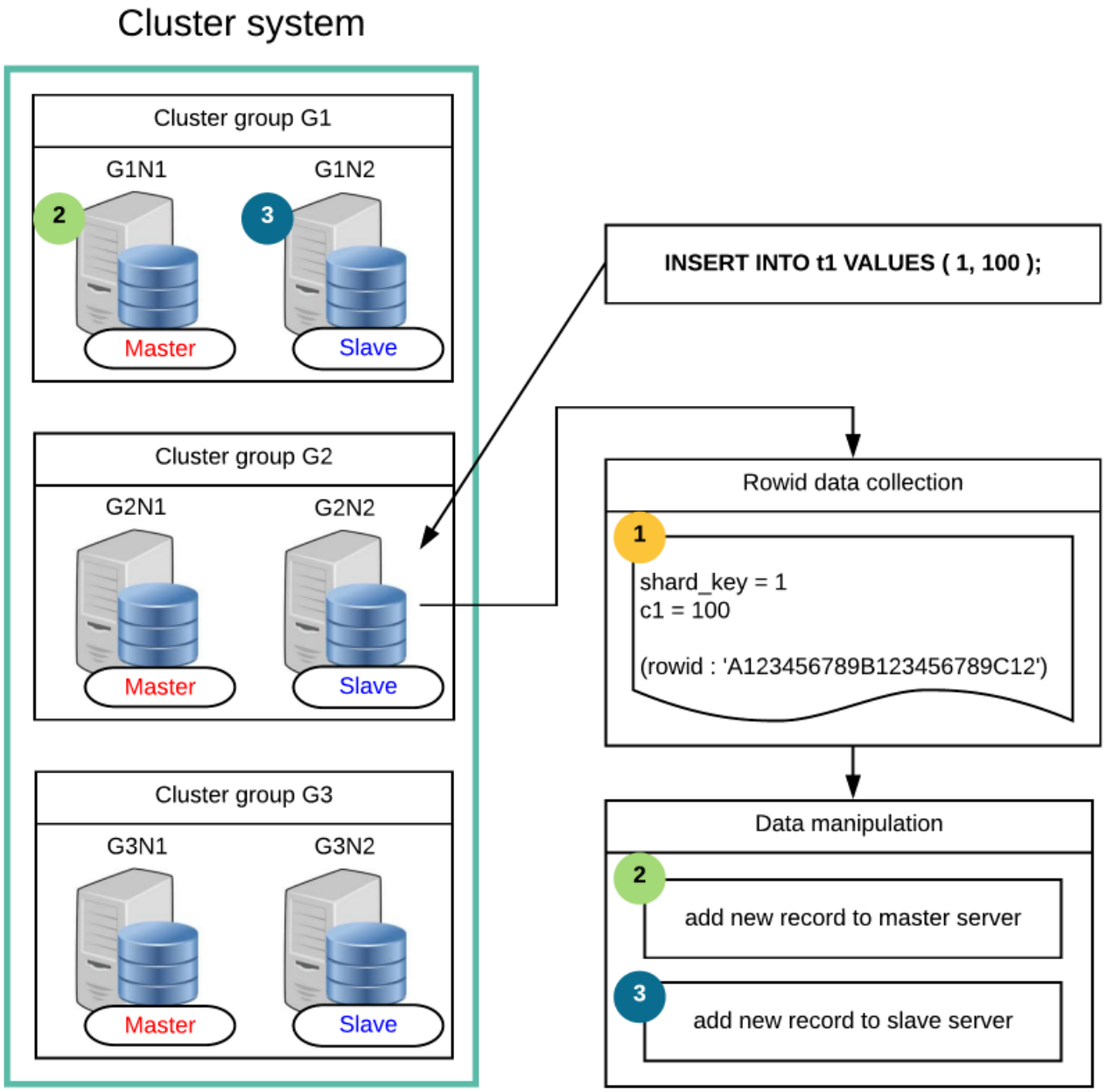
- **Global rowid Based DML without Using Global Secondary Index**
- **Global rowid Based DML Using Global Secondary Index**

Global rowid Based DML without Using Global Secondary Index

Global rowid based DML without using global secondary index is used when it is not required to determi ne whether the update target record in a master server and that in a slave server are same. It is when it c onfigures a new record without referring to the existing record like as INSERT.

Data manipulation using the rowid information without using the global secondary index is performed as follows.

Figure 12 Global rowid based DML without using global secondary index



When adding a record, a rowid is given to a new record and each record is stored at an appropriate positi on in a master server. All records are applied to a slave server after applying to a master server.

If the sharding key column is configured by using the value (shard_key = 1) as given above, it is found tha t the record whose shard_key is 1 is in G1 cluster group by the sharding strategy. Therefore, the record is added only to the G1 cluster group.

The following is a result of performing user query of **Global rowid based DML without using global secon dary index**. As a result, there is not any result output for global rowid based DML.

```
gSQL> \EXPLAIN PLAN INSERT INTO t1 VALUES ( 1, 100 );
1 row created.
>>>  start print plan
< Execution Plan >
=====================================================================
|  IDX  |  NODE DESCRIPTION                      |             ROWS |
---------------------------------------------------------------------
|    0  |  INSERT VALUES STATEMENT ("T1")        |                1 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")           |                0 |
|    2  |      TABLE ACCESS ("T1")               |                0 |
=====================================================================
     1  -  TARGET : NOTHING
     2  -  RANGE SHARD ( # 3 )
           READ COLUMN : T1.SHARD_KEY, T1.C1
<<<  end print plan
```

Global rowid based DML without using global secondary index supports the following query types.

- **INSERT INTO**
- **INSERT INTO name RETURNING**
- **INSERT INTO name RETURNING .. INTO**

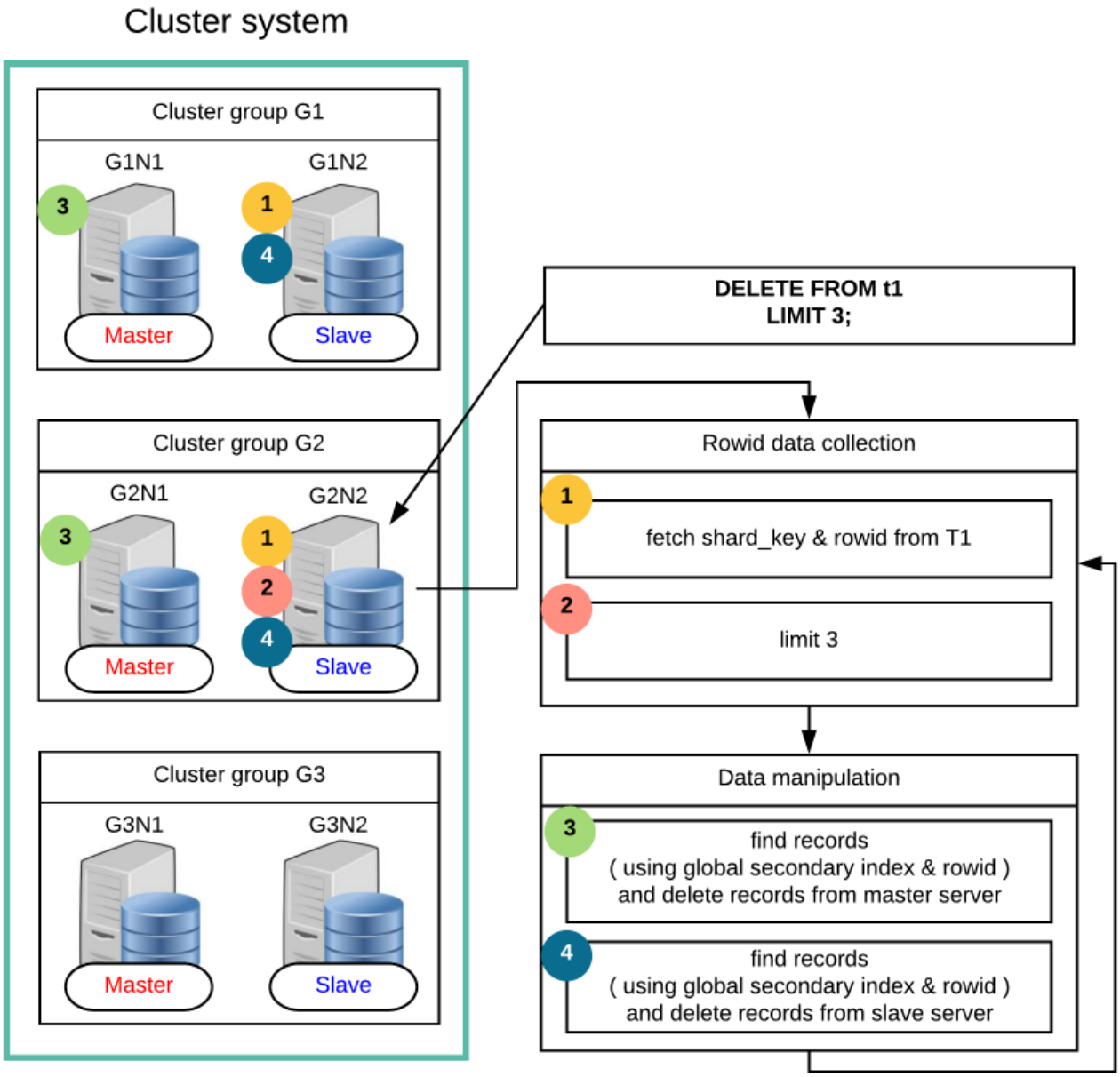
Global rowid Based DML Using Global Secondary Index

The replicated record included in a master server and a slave server have the same global rowid value.

It should be guaranteed that the same data in a master server and a slaver server are manipulated same b y using the global rowid information to manipulate the existing record. The global secondary index is use d to acquire the information about the record replicated through the global rowid.

Data manipulation using the rowid using the global secondary index is performed as follows.

Figure 13 Data manipulation using the rowid using the global secondary index



On the rowid information collection phase, the server received the user query collects row id and column value of the target record to be manipulated.

On the data manipulation phase, data in a master server and a slave server for a cluster group to which th e records selected on the rowid information collection phase belong is sequentially manipulated.

On the data manipulation phase, the server which collected the rowid information, transfers it to a maste r server and a slave server, and requests the data manipulation. The servers received the request retrieve t he target record stored in each server and manipulate it by using the received rowid information and the existing global secondary index.

The rowid information collection phase and the data manipulation phase are repeated until there is not a ny record to be manipulated.

The following is a result of performing user query of **Data manipulation using the rowid using the global secondary index**. As a result, there is not any result output for global rowid based DML.

```
gSQL> \EXPLAIN PLAN DELETE FROM t1 LIMIT 3;
3 rows deleted.
>>>  start print plan
< Execution Plan >
============================================================
|  IDX  |  NODE DESCRIPTION                |             ROWS |
------------------------------------------------------------
|    0  |  DELETE STATEMENT ("T1")         |                3 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")     |                3 |
|    2  |      PLAN BASED CLUSTER          | LOCAL/REMOTE   3 |
|    3  |        TABLE ACCESS ("T1")       |                2 |
============================================================
     1  -  TARGET : NOTHING
     2  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."$PHYSICAL_ROWID" FROM "PUBLIC"."T1"@LOCAL AS
"_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 2 rows, G2(G2N1,G2N2) 1 rows, G3(G3N1,G3N2) 0 rows
     3  -  RANGE SHARD ( # 3 )
           READ COLUMN : NOTHING
<<<  end print plan
```

Global rowid based DML using global secondary index supports the following query types.

- SELECT .. FOR UPDATE
- SELECT .. INTO .. FOR UPDATE
- DELETE FROM
- DELETE FROM name RETURNING
- DELETE FROM name RETURNING .. INTO
- DELETE FROM name WHERE CURRENT OF cursor_name
- UPDATE
- UPDATE name RETURNING
- UPDATE name RETURNING .. INTO
- UPDATE name WHERE CURRENT OF cursor_name

13.

SQL Objects

This chapter describes the concepts and features of the following objects which configure the database.

- Authorization: User and privilege
- Schema
- Tablespace
- Table
- Index
- Sequence
- View
- Synonym
- Stored procedure
- Stored function

13.1 Database

Database-related Statements

For more information, refer to the followings.

- Starting up the database: **ALTER SYSTEM {MOUNT | OPEN} DATABASE**
- Backup and recovery
  - **ALTER DATABASE BACKUP**
  - **ALTER DATABASE DELETE BACKUP**
  - **ALTER DATABASE RECOVER**
  - **ALTER DATABASE REGISTER**
  - **ALTER DATABASE RESTORE**
- Creating, dropping, altering log files
  - **ALTER SYSTEM CHECKPOINT**
  - **ALTER SYSTEM SWITCH LOGFILE**
  - **ALTER DATABASE ARCHIVELOG**
  - **ALTER DATABASE ADD LOGFILE**
  - **ALTER DATABASE DROP LOGFILE**
  - **ALTER DATABASE RENAME LOGFILE**
- Comments on objects: **COMMENT ON name IS**
- System statistics information: **ANALYZE SYSTEM**

The information which is related to the database objects can be retrieved through the following views.

Table 13-1 Database object related information

Schema	View name	Description
DICTIONARY_SCHEMA	ALL_NONSCHEMA_COMMENTS	Comment information of user accessible non-schema objects
INFORMATION_SCHEMA	INFORMATION_SCHEMA_CATALOG_NAME	Database name information
	SQL_FEATURES	The SQL standard compatibility information of GOLDILOCKS
	SQL_IMPLEMENTATION_INFO	The SQL standard compatibility information of GOLDILOCKS
	SQL_PACKAGES	The SQL standard compatibility information of GOL

Schema	View name	Description
		DILOCKS
	SQL_PARTS	The SQL standard compatibility information of GOL DILOCKS
	SQL_SIZING	The SQL standard compatibility information of GOL DILOCKS

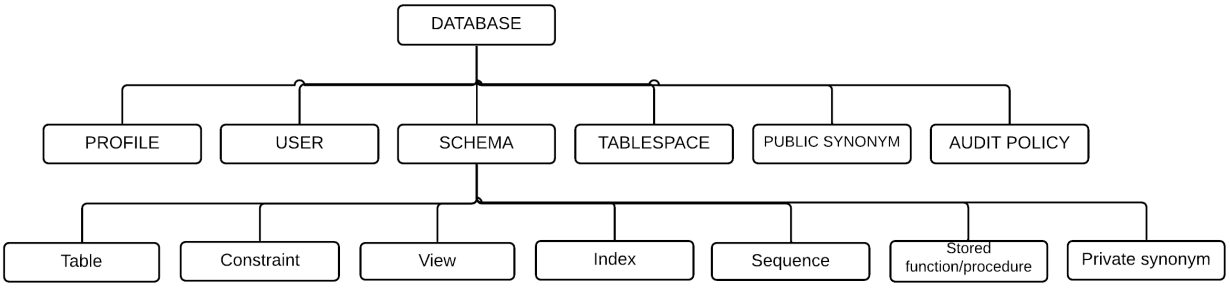
Database Configuration Objects

SQL Objects Which Configures Database

A database consists of multiple SQL objects.

SQL objects in database are classified as SQL schema objects and non-schema objects, depending on whe ther they are included in the SCHEMA.

Figure 1 SQL objects



SQL schema objects are included in the SCHEMA, and they are as follows.

- TABLE: It is an object which stores the physical data, and it consists of columns and rows.
- VIEW: It is a logical object which provides a relation name for the query, and it is similar to a table.
- INDEX: It is an object to improve the query performance.
- SEQUENCE: It is an object to generate numbers.
- CONSTRAINT: It is an object to retain the integrity of tables.
- SYNONYM: It is an alias of TABLE, VIEW, SEQUENCE, and other synonym.
- STORED PROCEDURE: It is a persistent stored module in a procedure form.
- STORED FUNCTION: It is a persistent stored module in a function form.

SQL schema object can be used together with a schema name, or it can be used omitting a schema name. If schema name is omitted, the name is interpreted by the user's schema path.

The following is as example of when objects are created by specifying the schema name.

```
gSQL> CREATE TABLE my_schema.lineitem ( id INTEGER );
gSQL> CREATE INDEX my_schema.my_index ON my_schema.lineitem ( id );
```

Non-schema object is not included in the schema, and they are as follows.

- PROFILE: It is a password management policy.
- AUDIT POLICY: It is an audit policy.
- USER: It is a user.
- SCHEMA: It is the logical locations of the SQL schema objects.
- TABLESPACE: It is the physical spaces of the SQL schema objects.
- PUBLIC SYNONYM: It is an alias of TABLE, VIEW, SEQUENCE, and other synonym which does not hav e schema name

The SQL standard explicitly defines concepts and syntax for the SCHEMA objects. However, the concepts of USER and DATABASE are described only, and their syntaxes are not defined in SQL. The SQL standard does not deal with the TABLESPACE object. Namely, the SQL standard does not explicitly define the non-schema objects.

GOLDILOCKS defines USER, SCHEMA, TABLESPACE as separate descendant of a database. However, oth er DBMS vendors define the concepts of non-schema objects as follows.

- GOLDILOCKS
  - User and schema are separate objects.
  - User either does not have a schema, or has multiple schemas.
  - The relationship between user and schema is User : Schema = 1 : N.
- Oracle
  - User and schema is defined as similar concepts.
  - The relationship between user and schema is User : Schema = 1 : 1.
- DB2
  - User is not a descendant of DATABASE.
  - The relationship between user and schema is User : Schema = 1 : N.
- Postgres
  - User is not a descendant of DATABASE.
  - The relationship between user and schema is User : Schema = 1 : N.
- MySQL
  - User and schema is defined as similar concepts.
  - User is a descendant of DATABASE(SCHEMA).

Name Space of Objects

An object in the database has an identifiable name.
An SQL schema object has a unique name within a schema.
For example, the same lineitem table objects can be created in different schemas as follows.

```
gSQL> CREATE TABLE my_schema.lineitem ( id INTEGER );
gSQL> CREATE TABLE your_schema.lineitem ( name VARCHAR(128) );
```

SQL schema objects have the following name spaces in a single schema as follows.

- TABLE, VIEW, SEQUENCE, PRIVATE SYNONYM, STORED PROCEDURE, STORED FUNCTION
- INDEX
- CONSTRAINT

Table and view can not be created under the same name, but table and index can be created under the s ame name, as follows.

- Table and view can not be created under the same name.

```
gSQL> CREATE TABLE my_relation ( id INTEGER );
gSQL> CREATE VIEW my_relation ( name ) AS SELECT name FROM tmp_relation;
```

- Table and index can be created under the same name.

```
gSQL> CREATE TABLE my_object ( id INTEGER );
gSQL> CREATE INDEX my_object ON my_table ( name );
```

A non-schema object has an identifiable name within a database. Non-schema objects have the name sp aces as follows.

- PROFILE
- AUDIT POLICY
- USER
- SCHEMA
- TABLESPACE

Namely, USERs can not be created under the same name, but USER and SCHEMA can be created under t he same name.

- my_name USER object and my_name SCHEMA object are created.

```
gSQL> CREATE USER my_name IDENTIFIED BY my_name WITH SCHEMA my_name;
```

Built-in Objects

When creating database, GOLDILOCKS automatically creates objects such as user, schema, tablespace which are necessary for system operation.

Built-in User

When creating database, the following accounts are automatically created. The built-in accounts can not be removed except TEST user.

- _SYSTEM account
  - It is an account used by the internal system. When creating database, it is the owner of the built-in objects. When user creates an object, it becomes the grantor who grants the privilege to the object owner.
- SYS account
  - It is a user to manage the database.
- TEST account
  - It is a user for testing, and this object can be removed.
- ADMIN account
  - It is a role to manage the database.
- SYSDBA account
  - It is a role to start up/shut down the database.
- PUBLIC account
  - It is an account for all users. It is used to control the privileges for all users.

Built-in Schema

When creating database, the following schemas are automatically created. All built-in schemas can not be removed.

- DEFINITION_SCHEMA
  - It consists of the physical tables which store all objects information of the database.
- FIXED_TABLE_SCHEMA
  - It consists of the fixed tables which display the data structure information of system in a table form.
- DICTIONARY_SCHEMA
  - It consists of the user perspective views for querying object information of the database.

- INFORMATION_SCHEMA
  - It consists of the views defined in the SQL standard for querying the object information of the dat abase.
- PERFORMANCE_VIEW_SCHEMA
  - It consists of the user perspective views for querying the state of the system.
- SESSION_SCHEMA
  - It is the schema to manage objects in session unit, and the user can not use it.
- PUBLIC
  - It is the schema in which all users can create objects, and it differs from the PUBLIC account, whi ch means all users.

Built-in Tablespace

When creating database, the following tablespaces are created automatically. All the built-in tablespaces can not be removed.

- DICTIONARY_TBS
  - It stores the dictionary tables to manage the SQL objects information.
- MEM_UNDO_TBS
  - It stores undo segments and transactions information.
- MEM_DATA_TBS
  - It is the first created data tablespace.
  - The user created tables are stored in data tablespace.
- MEM_TEMP_TBS
  - It is the first created temporary tablespace.
  - The temporary objects such as sort and hash which are created during query processing are store d in the temporary tablespace.
- DISK_DATA_TBS
  - It is the first created disk data tablespace.
  - The data tablespace stores objects such as a user-created table.
- MEM_AUX_TBS
  - It is a system auxiliary tablespace and it stores records which is automatically created such as an a udit record.
- MEM_TRANS_TBS
  - It is valid in the cluster system, and manages the global transaction information.
  - It is not created in the standalone system.

Built-in Profile

When creating database, the "DEFAULT" profile is automatically created. Password parameter informatio n of the "DEFAULT" profile is as follows.

Table 13-2 Configuration of DEFAULT profile

Parameter	Value
FAILED_LOGIN_ATTEMPTS	10
PASSWORD_LOCK_TIME	1
PASSWORD_LIFE_TIME	180
PASSWORD_GRACE_TIME	7
PASSWORD_REUSE_MAX	UNLIMITED
PASSWORD_REUSE_TIME	UNLIMITED
PASSWORD_VERIFY_FUNCTION	NULL

The followings are characteristics of the default values of "DEFAULT" profile.

- Account lockout
  - An account is locked for one day (PASSWORD_LOCK_TIME) after consecutive 10 (FAILED_LOGIN_ATTEMPTS) times of failed login attempts.
- Password expiration
  - After 180 days (PASSWORD_LIFE_TIME) has elapsed, the password expires after seven days of grace period (PASSWORD_GRACE_TIME).
- Password reusable
  - The old password can be reused.
- Password complexity verification
  - The password complexity is not verified.

13.2 Profile

Profile-related Statements

For more information, refer to the followings.

- Creating a profile: **CREATE PROFILE**.
- Dropping a profile: **DROP PROFILE**.
- Altering a profile: **ALTER PROFILE**.
- Assigning a profile to the user: **CREATE USER**, **ALTER USER**
- Clearing a password history: **ALTER DATABASE CLEAR PASSWORD HISTORY**.

Table 13-3 Profile object related information

Schema	View	Description
DICTIONARY_SCHEMA	**DBA_PROFILES**	All profile information
	DBA_USERS	User profile information

Concepts of Profile

GOLDILOCKS performs user authentication for database security. The password management policy is required because the user authentication password is vulnerable to theft, forgery and misuse.

Profile includes information such as this password management policy. DBA or security managers assign the profile to a user, and apply the password management policy which is appropriate to the corresponding user.

Creating, Altering, Allocating Profile

A profile is created by using CREATE PROFILE statement.

```
CREATE PROFILE profile1 LIMIT
       FAILED_LOGIN_ATTEMPTS     10
       PASSWORD_LOCK_TIME        1
       PASSWORD_LIFE_TIME        180
       PASSWORD_GRACE_TIME       7
       PASSWORD_REUSE_MAX        UNLIMITED
       PASSWORD_REUSE_TIME       UNLIMITED
```

```
        PASSWORD_VERIFY_FUNCTION  NULL;
```

A profile is allocated by using CREATE USER, ALTER USER statements.

```
CREATE USER u1 IDENTIFIED BY u1 PROFILE profile1;
ALTER USER u2 PROFILE profile1;
```

The profile parameters are updated by using ALTER PROFILE statement.

```
ALTER PROFILE profile1 LIMIT
      PASSWORD_REUSE_MAX        3
      PASSWORD_REUSE_TIME       30;
```

If a profile is not allocated to a user, the user is not restricted on creating and using the password.
If the created profile or a DEFAULT profile is allocated to a user, the user complies with the profile's pass word policies when creating and using the password.

Setting Password of DEFAULT Profile

When the default profile is assigned to a user, the password is managed as follows.

Table 13-4 DEFAULT profile of password

Parameter	Default setting	Description
FAILED_LOGIN_ATTEMPS	10	The allowed number of consecutive login failure The account is locked after consecutive 10 times login attempt failures.
PASSWORD_LOCK_TIME	1	The account lockout duration If the consecutive login failures exceed the allowable value, the account is locked for one day.
PASSWORD_LIFE_TIME	180	The password life time The password is expired after 180 days.
PASSWORD_GRACE_TIME	7	The duration to change the password when the the password is expired. The password should be changed within seven days after first login sinc e the password is expired. If a user does not change the password withi n the period, the user can not log in using the password.
PASSWORD_REUSE_MAX	UNLIMITED	The number of times of which passwords are not reusable. PASSWORD_REUSE_MAX should be set together with PASSWORD_RE USE_TIME. If both of the two values are UNLIMITED, the password can always be reused.
PASSWORD_REUSE_TIME	UNLIMITED	The duration which the password can not be reused.

Account Lockout

If the number of consecutive login atempt failures exceed the number of times specified in FAILED_LOGIN_ATTEMPTS, the account is locked during the period specified in PASSWORD_LOCK_TIME.

```
CREATE PROFILE profile1 LIMIT
       FAILED_LOGIN_ATTEMPTS     10
       PASSWORD_LOCK_TIME        1;
ALTER USER u1 PROFILE profile1;
```

When a user u1's login attempts consecutively fails more than 10 times, the account is locked for one days. And after one day the account is automatically unlocked.

If the PASSWORD_LOCK_TIME value is not specified, it is regarded as the value which is specified in PASSWORD_LIFE_TIME of DEFAULT profile.

If the PASSWORD_LOCK_TIME value is UNLIMITED the locked accounts is not automatically released. Therefore, the following statement should be executed to unlock the account.

```
ALTER USER u1 ACCOUNT UNLOCK;
```

If login is successful, the number of failed login attempt is initialized to zero.

A security manager can explicitly lock the user accounts. In this case, the user accounts can not be automatically released so the security manager should unlock the user accounts.

```
ALTER USER u1 ACCOUNT LOCK;
ALTER USER u1 ACCOUNT UNLOCK;
```

Password Lifetime

PASSWORD_LIFE_TIME specifies the life time the password. After the life time, the password is expired. A user, DBA or security manager should change the password after a password is expired.

```
CREATE PROFILE profile1 LIMIT
       PASSWORD_LIFE_TIME        180
       PASSWORD_GRACE_TIME       7;
ALTER USER u1 PROFILE profile1;
```

The grace period starts since when the user u1 has tried to log in for the first time after 180 days.

During seven days of the grace period, the user is reminded to enter a new password whenever accessing the account, until he changes the password.

If seven days of the grace period passed and the password is not changed, the user can not login until en tering a new password.

A password can be expired by using CREATE USER or ALTER USER statements.

```
ALTER USER u1 PASSWORD EXPIRE;
```

When the password is expired, the error (ERR-28000(16312) the password has expired) occurs whenever logging in as follows, then a new password should be entered.

```
% gsql u1 u1
ERR-28000(16312): the password has expired
Changing password for u1
New password:
Retype new password:
Connected to GOLDILOCKS Database.
gSQL>
```

Reusing Password

The password can be reused after it is changed as many times as the specified value in PASSWORD_REUS E_MAX. Also, it should be after the specified time in PASSWORD_REUSE_TIME.

```
CREATE PROFILE profile1 LIMIT
       PASSWORD_REUSE_MAX        2
       PASSWORD_REUSE_TIME       1;
ALTER USER u1 PROFILE profile1;
```

The user u1 can reuse the current password after the password has been changed for two times, and 10 days elapsed.

```
ALTER USER u1 IDENTIFIED BY u1 REPLACE u1;
ALTER USER u1 IDENTIFIED BY u1 REPLACE u1
*
ERROR at line 1:
ORA-28007: the password cannot be reused
ALTER USER u1 IDENTIFIED BY u2 REPLACE u1;
User altered.
ALTER USER u1 IDENTIFIED BY u3 REPLACE u2;
User altered.
```

- 10 days elapsed.

```
ALTER USER u1 IDENTIFIED BY u1 REPLACE u3;
User altered.
```

The both conditions should be satisfied to reuse the old password. If only one of the value in PASSWORD_REUSE_MAX and PASSWORD_REUSE_TIME is UNLIMITED, the password can not be reused.
If both of values are UNLIMITED, the password can always be reused.

Table 13-5 Password reuse

PASSWORD_REUSE_MAX	PASSWORD_REUSE_TIME	Password reusability
Integer value	Integer value	If both of the conditions are satisfied, it can be reused.
Integer value	UNLIMITED	It can not be reused.
UNLIMITED	Integer value	It can not be reused.
UNLIMITED	UNLIMITED	It can always be reused.

Password Complexity Verification

Password complexity verification checks if the password is complex enough to protect against breaking in to the system.
GOLDILOCKS supports the method of password complexity verification as follows.

Table 13-6 Password complexity verification

Method	Description
KISA_VERIFY_FUNCTION	• 8 or more characters • 1 or more letters • 1 or more numbers • 1 or more special characters
ORA12C_VERIFY_FUNCTION	• 8 or more characters • 1 or more letters • 1 or more numbers • Database name should not be included. • Username or the reversed username should not be included. • *goldilocks* should not be included. • *oracle* should not be included. • The following simple password can not be used. ◦ welcome1, database1, account1, user1234, password1, oracle123, computer1, abcdefg1, change_on_intall • The new password should be different at least 3 characters from the old password.
	• 9 or more characters • 2 or more uppercases

Method	Description
ORA12C_STRONG_VERIFY_FUNCTION	• 2 or more lowercases • 2 or more numbers • 2 or more special characters • The new password should be different at least 4 characters from the old password.
VERIFY_FUNCTION_11G	• 8 or more characters • 1 or more letters • 1 or more numbers • Username should not be included. • The new password should be different at least 3 characters from the old password.
VERIFY_FUNCTION	• It should not be same as the username. • 4 or more characters • 1 or more letters • 1 or more numbers • 1 or more special characters • The following simple password can not be used. ◦ welcome, database, account, user, password, oracle, computer, abcd • The new password should be different at least 3 characters from the old password.

For more information about profile and user setting, refer to **CREATE PROFILE**, **CREATE USER**.

13.3 Audit Policy

Audit Policy-related Statement

For more information, refer to the followings.

- Creating audit policy: **CREATE AUDIT POLICY**
- Dropping audit policy: **DROP AUDIT POLICY**
- Altering audit policy: **ALTER AUDIT POLICY**
- Activating audit policy: **AUDIT POLICY**
- Deactivating audit policy: **NOAUDIT POLICY**
- Dropping audit trail: **ALTER DATABASE CLEAR AUDIT TRAIL**

Table 13-7 Audit policy object information

Schema	View	Description
DICTIONARY_SCHEMA	**AUDIT_POLICIES**	Information about all audit policies
	AUDIT_POLICY_OPTIONS	Information about audit policy option
	AUDIT_POLICY_ENABLED	Information about activating audit policy

Examples

AUDIT SYSTEM ON DATABASE privilege is required to perform the followings.

Creating Audit Policy

Perform **CREATE AUDIT POLICY** statement to create an audit policy object.

The following is an example of creating an audit_t1_dml object to audit the DML for a u1.t1 table.

```
CREATE AUDIT POLICY audit_t1_dml
       ACTIONS INSERT ON u1.t1
             , DELETE ON u1.t1
             , UPDATE ON u1.t1
;
Audit policy created.
```

Enquire the information about audit policy options by using **AUDIT_POLICY_OPTIONS** view.

```
SELECT policy_name
     , audit_option
     , object_schema
     , object_name
  FROM audit_policy_options
 WHERE policy_name = 'AUDIT_T1_DML'
 ORDER BY audit_option
;
POLICY_NAME  AUDIT_OPTION OBJECT_SCHEMA OBJECT_NAME
------------ ------------ ------------- -----------
AUDIT_T1_DML DELETE       U1            T1
AUDIT_T1_DML INSERT       U1            T1
AUDIT_T1_DML UPDATE       U1            T1
3 rows selected.
```

Activating Audit Policy

Use **AUDIT POLICY** statement to activate the audit policy.

The following is an example of activating an audit policy to leave an audit record when a user except for u1, sys succeeded to perform DML for u1.t1 table.

```
AUDIT POLICY audit_t1_dml
      EXCEPT u1, sys
      WHENEVER SUCCESSFUL
;
```

The activated audit policy is applied to the newly created session, but it does not affect the existing sessio ns.

View the information about activated audit policy by using **AUDIT_POLICY_ENABLED**.

```
SELECT policy_name
     , enabled_opt
     , user_name
     , when_success
     , when_failure
  FROM audit_policy_enabled
 WHERE policy_name = 'AUDIT_T1_DML'
 ORDER BY user_name
;
POLICY_NAME             ENABLED_OPT           USER_NAME  WHEN_SUCCESS WHEN_FAILURE
```

```
-------------------- ---------------------- ---------- ------------ ---------
AUDIT_T1_DML         EXCEPT                 SYS        YES          NO
AUDIT_T1_DML         EXCEPT                 U1         YES          NO
2 rows selected.
```

After the audit policy is activated, the corresponding actions create audit records.

The following is an example of when u2 user successfully performs SQL statements.

```
SELECT COUNT(*) FROM u1.t1;
INSERT INTO u1.t1 VALUES ( 1 );
UPDATE u1.t1 SET id = id + 1 WHERE id = 1;
DELETE u1.t1 WHERE id = 2;
COMMIT;
```

In the example above, INSERT, UPDATE, DELETE are target actions of an audit, so it creates audit records, but SELECT and COMMIT are not a target action of an audit, so it does not create audit records.

Viewing Audit Trail

SELECT ON DICTIONARY_SCHEMA.AUDIT_TRAIL privilege for AUDIT_TRAIL view is required to view audit records.

The following is an example of viewing an audit trail created by an audit_t1_dml audit policy.

```
SELECT policy_name
     , logon_username
     , action_name
     , object_schema
     , object_name
     , sql_text
  FROM audit_trail
 WHERE policy_name = 'AUDIT_T1_DML'
;
POLICY_NAME  LOGON_USERNAME ACTION_NAME OBJECT_SCHEMA OBJECT_NAME SQL_TEXT
------------ -------------- ----------- ------------- -----------
----------------------------------------
AUDIT_T1_DML U2             INSERT      U1            T1          INSERT INTO u1.t1 VALUES ( 1
)
AUDIT_T1_DML U2             UPDATE      U1            T1          UPDATE u1.t1 SET c1 = c1 + 1
WHERE c1 = 1
AUDIT_T1_DML U2             DELETE      U1            T1          DELETE u1.t1 WHERE c1 = 2
3 rows selected.
```

Dropping Audit Trail

When an audit policy is activated, the size of an audit trail keeps increasing.
Execute the following statement to drop the audit trail.

```
ALTER DATABASE CLEAR AUDIT TRAIL;
```

Store it in the user table and drop it as follows to store the audit trail.

- Creating a user table

```
CREATE TABLE my_audit_trail
AS SELECT *
     FROM audit_trail
     WITH NO DATA;
```

- Storing it in a user table, then dropping it.

```
INSERT INTO my_audit_trail SELECT * FROM audit_trail;
ALTER DATABASE CLEAR AUDIT TRAIL;
```

Create and manage the view as follows to view both the stored audit trail and the current audit_trail.

```
CREATE VIEW unified_audit_trail
AS
SELECT * FROM my_audit_trail
 UNION ALL
SELECT * FROM dictionary_schema.audit_trail
;
```

Deactivating Audit Policy

Deactivate the audit policy by using the following statement.

```
NOAUDIT POLICY audit_t1_dml;
```

Deactivating audit policy affects the newly created session, but it does not affect the activated informatio n about existing sessions.

Dropping Audit Policy

Execute the following statement to drop the audit policy object.

```
DROP AUDIT POLICY audit_t1_dml;
```

The audit policy object should be deactivated to be dropped, and dropping the object does not affect the existing sessions.

Concepts of Audit Policy

Audit Trail

Viewing Audit Trail

Audit record can be viewed by using DICTIONARY_SCHEMA.AUDIT_TRAIL view.

SELECT privilege is required to view AUDIT_TRAIL view.

```
GRANT SELECT ON DICTIONARY_SCHEMA.AUDIT_TRAIL TO user_name;
```

AUDIT_TRAIL view has the following information.

Table 13-8 Column information

Information	Column name	Description
Session information	MEMBER_NAME	Cluster member name
	SESSION_ID	Session identifier
	SESSION_SERIAL	Session serial number
	LOGON_USERNAME	Logon user name of the user whose actions were audited
	CURRENT_USERNAME	Effective user for the statement execution
	SERVER_PROCESS	Server process identifer for the session
Peer client information	CLIENT_PROGRAM_NAME	Client program used for session
	CLIENT_USERNAME	Client operating system user name for the session
	CLIENT_PROCESS	Client process identifer for the session
	CLIENT_HOST	Client host ip address for the session
	CLIENT_PORT	Client port number for the session
	CLIENT_TERMINAL	Client terminal name for the session
	TRANSACTION_ID	Transaction identifier
	SCN	System change number (SCN) string of the query at the time of the event

Information	Column name	Description
SQL information	GCN	Global change number (GCN) of the query at the time of the event
	DCN	Domain change number (DCN) of the query at the time of the event
	LCN	Local change number (LCN) of the query at the time of the event
	STMT_NO	Numeric number for each statement run in a session
	SQL_TEXT	SQL associated with the event
	SQL_BINDS	List of bind variables, if any, associated with SQL_TEXT
	RETURN_CODE	Error code generated by the action, zero if the action succeeded
	ERROR_MESSAGE	Error message generated by the action, null if the action succeeded
Event information	ENTRY_ID	Audit trail entry identifier in the session
	EVENT_TIMESTAMP	Timestamp of the creation of the audit trail entry in local time zone
	POLICY_NAME	Audit policy name that caused the current audit record
	PRIVILEGE_USED	Database privilege used to execute the action
	ACTION_NAME	Action name executed by the user
	OBJECT_TYPE	Object type of object affected by the action
	OBJECT_SCHEMA	Schema name of object affected by the action
	OBJECT_NAME	Object name of object affected by the action

Storing Audit Trail

AUDIT_TRAIL view consists of the following tables.

- AUDIT_TRAIL_SESSION
  - It records a record per a session.
  - Session information
  - Peer client information
- AUDIT_TRAIL_SQL
  - It records a record per SQL.
  - SQL information
- AUDIT_TRAIL_EVENT
  - It records a record per an audit option.
  - Audit event information

Audit records configuring an audit trail is devided into multiple tables then stored.
The schema of the tables is DEFINITION_SCHEMA, and it is stored in MEM_AUX_TBS tablespace.

- DEFINITION_SCHEMA
  - It is a schema storing dictionary tables.
- MEM_AUX_TBS
  - System auxiliary tablespace
  - It is a tablespace to store a record which automatically created by the database.
  - It is managed by a separate tablespace not to affect the service management by an automatically

created record.

Creating Audit Record

If an audit policy is activated, it creates an audit record when the corresponding action occurs.
It creates one or more audit records when multiple corresponding actions occur.

- If similar audit options are listed as follows, it creates a single audit record.
  - Defining an audit policy

```
CREATE AUDIT POLICY p1
       PRIVILEGES INSERT ANY TABLE
       ACTIONS INSERT;
AUDIT POLICY p1;
```

  - Performing an audit action

```
INSERT INTO other_user.t1 VALUES ( 1 );
```

- If different audit options are listed as follows, it creates two audit records.
  - Defining an audit policy

```
CREATE AUDIT POLICY p1
       ACTIONS SELECT ON u1.t1
             , SELECT ON u1.t2;
AUDIT POLICY p1;
```

  - Performing an audit action

```
SELECT COUNT(*) FROM u1.t1 A, u1.t2 B WHERE A.id = B.id;
```

- If multiple audit policies are activated in the identical action as follows, it creates two audit records.
  - Defining an audit policy

```
CREATE AUDIT POLICY p1
       PRIVILEGES INSERT ANY TABLE;
AUDIT POLICY p1;
CREATE AUDIT POLICY p2
       ACTIONS INSERT;
AUDIT POLICY p2;
```

    - Performing an audit action

```
INSERT INTO other.t1 VALUES (1);
```

Dropping Audit Trail (purge)

Execute the following statement to drop the audit trail.

```
ALTER DATABASE CLEAR AUDIT TRAIL;
```

Create a user table and store old audit record in it as follows.

```
CREATE TABLE my_audit_trail AS SELECT * FROM audit_trail WITH NO DATA;
```

Then, regularly store it by using INSERT .. SELECT statement before dropping the audit trail.

```
INSERT INTO my_audit_trail SELECT * FROM audit_trail;
ALTER DATABASE CLEAR AUDIT TRAIL;
```

Execute DELETE statement by using EVENT_TIMESTAMP column to store the audit record of the specified period.

```
INSERT INTO my_audit_trail SELECT * FROM audit_trail;
DELETE FROM my_audit_trail WHERE event_timestamp < ADD_MONTHS( sysdate, -3 );
ALTER DATABASE CLEAR AUDIT TRAIL;
```

View the old audit record and the current audit record together by creating a view as follows.

```
CREATE VIEW audit_trail_view
AS SELECT * FROM dictionary_schema.audit_trail
   UNION ALL
   SELECT * FROM my_audit_trail;
SELECT * FROM audit_trail_view;
```

Configuring Audit Policy

Audit policy may include the following options.

- Privilege auditing
    - It audits SQL performance by using the database privilege.
- Object action auditing
    - It audits SQL performance for the specific object.
- System action auditing

- It audits SQL performance for all objects.

Multiple audit options can be managed by creating multiple audit policies, but it is recommended to man age multiple audit options by creating a small number of audit policies.

The activated audit policy information is constructed as a session information at logon time, so the less th e number of audit policies, the less the load becomes.
Moreover, if multiple audit policies are activated, then it determines whether to create an audit record for an SQL statement, so a load creating multiple audit records may occur.

Audit policy information constructed in a session at logon time is not affected by dropping, altering, acti vating or deactivating an audit policy.
Altering an audit policy is applied only to a newly logon session.

Privilege Auditing

Privilege auditing is set to audit when SQL statement is successfully performed by using the database privi lege.
It does not create an audit record which is based on the privilege auditing about sys user (the database o wner).

The database privilege which can be listed for the privilege auditing can be viewed through **V$AUDITABL E_DB_PRIVILEGES** view.

```
SELECT privilege_name FROM v$auditable_db_privileges;
PRIVILEGE_NAME
----------------------
ADMINISTRATION
ALTER DATABASE
ALTER SYSTEM
ACCESS CONTROL
CREATE USER
ALTER USER
DROP USER
CREATE TABLESPACE
ALTER TABLESPACE
DROP TABLESPACE
USAGE TABLESPACE
CREATE SCHEMA
DROP SCHEMA
ANALYZE ANY
CREATE ANY TABLE
```

```
ALTER ANY TABLE
DROP ANY TABLE
SELECT ANY TABLE
INSERT ANY TABLE
DELETE ANY TABLE
UPDATE ANY TABLE
LOCK ANY TABLE
CREATE ANY VIEW
DROP ANY VIEW
CREATE ANY SEQUENCE
ALTER ANY SEQUENCE
DROP ANY SEQUENCE
USAGE ANY SEQUENCE
CREATE ANY INDEX
ALTER ANY INDEX
DROP ANY INDEX
CREATE ANY SYNONYM
DROP ANY SYNONYM
CREATE PUBLIC SYNONYM
DROP PUBLIC SYNONYM
CREATE PROFILE
ALTER PROFILE
DROP PROFILE
CREATE ANY PROCEDURE
ALTER ANY PROCEDURE
DROP ANY PROCEDURE
EXECUTE ANY PROCEDURE
AUDIT SYSTEM
PURGE DBA_RECYCLEBIN
44 rows selected.
```

The following is an example of when a user u1 who has SELECT ANY TABLE privilege activates an audit p olicy for the privilege auditing.

```
CREATE AUDIT POLICY p1
       PRIVILEGES SELECT ANY TABLE;
AUDIT POLICY p1;
```

Whether the audit record is created when a user u1 performs the following two statements is as follows.

- SELECT * FROM u1.t1
  - It does not use SELECT ANY TABLE privilege because it is an owner of the object.

    - An audit record does not exist.
- SELECT * FROM other.t1
    - It succeeds with SELECT ANY TABLE privilege even though it does not have SELECT ON other.t1 privilege.
    - It creates an audit record.

Use AUDIT_POLICY_OPTIONS view to view the privilege auditing information as follows.

```
SELECT audit_option
     , audit_option_type
     , object_schema
     , object_name
  FROM audit_policy_options
 WHERE policy_name = 'P1'
;
AUDIT_OPTION         AUDIT_OPTION_TYPE    OBJECT_SCHEMA     OBJECT_NAME
-------------------- -------------------- ----------------- ------------
SELECT ANY TABLE     DATABASE PRIVILEGE   null              null
```

Auditing Object Action

It audits SQL which is performed for a specific object.
Actions to be audited per each object type are as follows.

Table 13-9 Audit action per object type

Object type	Actions
Table	ALTER, COMMENT, DELETE, GRANT, INDEX, INSERT, LOCK, RENAME, SELECT, UPDATE
View	ALTER, COMMENT, GRANT, SELECT
Sequence	ALTER, COMMENT, GRANT, SELECT
Stored function/ procedure	ALTER, COMMENT, EXECUTE, GRANT

Create the audit policy as follows to audit DML for table u1.t1.

```
CREATE AUDIT POLICY audit_t1_dml
       ACTIONS INSERT ON u1.t1
             , DELETE ON u1.t1
             , UPDATE ON u1.t1
;
```

Execute AUDIT_POLICY_OPTIONS view as follows to view the information about object action auditing.

```
SELECT audit_option
     , audit_option_type
     , object_schema
     , object_name
  FROM audit_policy_options
 WHERE policy_name = 'AUDIT_T1_DML'
 ORDER BY audit_option
;
AUDIT_OPTION AUDIT_OPTION_TYPE OBJECT_SCHEMA OBJECT_NAME
------------ ----------------- ------------- -----------
DELETE       OBJECT ACTION     U1            T1
INSERT       OBJECT ACTION     U1            T1
UPDATE       OBJECT ACTION     U1            T1
3 rows selected.
```

ALL option such as ALL ON schema.object means all audit actions which can be defined for the correspon ding object.

The following is an example of using ALL option together with other options.

```
CREATE AUDIT POLICY p1
       ACTIONS ALL ON u1.seq1
             , ALTER ON u1.seq1
;
Audit policy created.
SELECT audit_option
     , audit_option_type
     , object_schema
     , object_name
  FROM audit_policy_options
 WHERE policy_name = 'P1'
 ORDER BY audit_option
;
AUDIT_OPTION AUDIT_OPTION_TYPE OBJECT_SCHEMA OBJECT_NAME
------------ ----------------- ------------- -----------
ALL          OBJECT ACTION     U1            SEQ1
ALTER        OBJECT ACTION     U1            SEQ1
2 rows selected.
```

When dropping ALL option as follows, not every audit options are dropped, but only ALL option is dropp ed.

```
ALTER AUDIT POLICY p1
      DROP ACTIONS ALL ON u1.seq1;
Audit Policy altered.
SELECT audit_option
     , audit_option_type
     , object_schema
     , object_name
  FROM audit_policy_options
 WHERE policy_name = 'P1'
 ORDER BY audit_option;
AUDIT_OPTION AUDIT_OPTION_TYPE OBJECT_SCHEMA OBJECT_NAME
------------ ----------------- ------------- -----------
ALTER        OBJECT ACTION     U1            SEQ1
1 row selected.
```

Auditing success or failure of EXECUTE a stored function or a stored procedure is determined based only on whether it is executable at the time of the execution.

- WHENEVER NOT SUCCESSFUL creates an audit record when it can not performs a stored function/ procedure.
- WHENEVER SUCCESSFUL creates an audit record even when an error occurs while performing an SQL statement in a stored function/ procedure.
- If an auditing the failure of an SQL statement in a stored function/ procedure, then that SQL statement should be included in an auditing target.

The following is an example of executing SELECT statement including a stored function.

```
SELECT others.func1( t1.c1 )
  FROM t1;
```

It corresponds to WHENEVER NOT SUCCESSFUL when it fails to call others.func1() due to an error such as a lack of privilege. It corresponds to WHENEVER SUCCESSFUL even though an error occurs while executing an SQL statement in a stored function when it succeeds to call others.func1().

Auditing System Action

It audits an SQL statement regardless of a specific object.
A valid system action enquires V$AUDITABLE_SYSTEM_ACTIONS.

```
SELECT action_name FROM v$auditable_system_actions;
ACTION_NAME
------------------------
ALL
```

```
DDL
SELECT
INSERT
UPDATE
DELETE
EXECUTE
CREATE TABLE
DROP TABLE
ALTER TABLE
LOCK TABLE
TRUNCATE TABLE
ANALYZE TABLE
RENAME
CREATE INDEX
DROP INDEX
ALTER INDEX
CREATE SEQUENCE
DROP SEQUENCE
ALTER SEQUENCE
GRANT
REVOKE
CREATE SYNONYM
DROP SYNONYM
CREATE VIEW
DROP VIEW
ALTER VIEW
CREATE PROCEDURE
DROP PROCEDURE
ALTER PROCEDURE
CREATE FUNCTION
DROP FUNCTION
ALTER FUNCTION
COMMENT
ALTER DATABASE
CREATE PROFILE
DROP PROFILE
ALTER PROFILE
CREATE TABLESPACE
DROP TABLESPACE
ALTER TABLESPACE
CREATE USER
```

```
DROP USER
ALTER USER
CHANGE PASSWORD
CREATE SCHEMA
DROP SCHEMA
CREATE AUDIT POLICY
DROP AUDIT POLICY
ALTER AUDIT POLICY
AUDIT
NOAUDIT
ALTER SYSTEM
ALTER SESSION
ANALYZE SYSTEM
COMMIT
ROLLBACK
SAVEPOINT
LOGON
LOGOFF
SET SESSION
SET TRANSACTION
SET CONSTRAINTS
CREATE CLUSTER GROUP
DROP CLUSTER GROUP
ALTER CLUSTER GROUP
CREATE CLUSTER LOCATION
DROP CLUSTER LOCATION
ALTER CLUSTER LOCATION
PURGE CONSTRAINT
PURGE INDEX
PURGE TABLE
PURGE TABLESPACE
PURGE RECYCLEBIN
PURGE DBA_RECYCLEBIN
FLASHBACK TABLE
76 rows selected.
```

A system action name corresponding to each SQL statement is viewed by executing V$SQL_COMMAND v iew.

```
SELECT command, audit_action FROM v$sql_command;
COMMAND                                                  AUDIT_ACTION
```

```
---------------------------------------------------------- ------------------------
ALTER AUDIT POLICY                                         ALTER AUDIT POLICY
ALTER CLUSTER GROUP .. ADD CLUSTER MEMBER                  ALTER CLUSTER GROUP
ALTER CLUSTER GROUP .. OFFLINE CLUSTER MEMBER              ALTER CLUSTER GROUP
ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS               ALTER DATABASE
ALTER DATABASE OFFLINE INACTIVE CLUSTER MEMBERS            ALTER DATABASE
... Ellipsis ...
DROP CLUSTER LOCATION                                      DROP CLUSTER LOCATION
PROCEDUAL LANGUAGE BLOCK                                   null
PURGE CONSTRAINT                                           PURGE CONSTRAINT
PURGE INDEX                                                PURGE INDEX
PURGE TABLE                                                PURGE TABLE
PURGE TABLESPACE                                           PURGE TABLESPACE
PURGE RECYCLEBIN                                           PURGE RECYCLEBIN
PURGE DBA_RECYCLEBIN                                       PURGE DBA_RECYCLEBIN
FLASHBACK TABLE                                            FLASHBACK TABLE
185 rows selected.
```

The following is an example of creating an audit policy including a system action, and enquiring an audit option.

```
CREATE AUDIT POLICY p1
       ACTIONS SELECT
             , DROP TABLE
             , DROP USER
;
Audit Policy created.
SELECT audit_option
     , audit_option_type
     , object_schema
     , object_name
  FROM audit_policy_options
 WHERE policy_name = 'P1'
 ORDER BY audit_option
;
AUDIT_OPTION AUDIT_OPTION_TYPE OBJECT_SCHEMA OBJECT_NAME
------------ ----------------- ------------- -----------
DROP TABLE   SYSTEM ACTION     null          null
DROP USER    SYSTEM ACTION     null          null
SELECT       SYSTEM ACTION     null          null
3 rows selected.
```

Useful Audit Policy

The following is an example of defining a useful audit policy.

- Auditing a logon failure

```
CREATE AUDIT POLICY AUDIT_LOGON_FAILURES
       ACTIONS LOGON
;
AUDIT POLICY AUDIT_LOGON_FAILURES
      WHENEVER NOT SUCCESSFUL
;
```

- Auditing a Data Definition Language (DDL) performance

```
CREATE AUDIT POLICY AUDIT_DDL
       ACTIONS DDL
;
AUDIT POLICY AUDIT_DDL
      WHENEVER SUCCESSFUL
;
```

- Auditing a parameter alteration

```
CREATE AUDIT POLICY AUDIT_DATABASE_PARAMETER
       ACTIONS ALTER DATABASE
             , ALTER SYSTEM
;
AUDIT POLICY AUDIT_DATABASE_PARAMETER
      WHENEVER SUCCESSFUL
;
```

- Auditing an account alteration

```
CREATE AUDIT POLICY AUDIT_ACCOUNT_MGMT
       ACTIONS CREATE USER
             , DROP USER
             , ALTER USER
             , CHANGE PASSWORD
             , GRANT
             , REVOKE
;
AUDIT POLICY AUDIT_ACCOUNT_MGMT
```

```
;
```

- Auditing the recommendations of Center for Internet Security (CIS)

```
CREATE AUDIT POLICY AUDIT_CIS_RECOMMENDATIONS
       PRIVILEGES ALTER SYSTEM
                , ALTER DATABASE
          ACTIONS CREATE USER
                , DROP USER
                , ALTER USER
                , CHANGE PASSWORD
                , GRANT
                , REVOKE
                , CREATE PROFILE
                , ALTER PROFILE
                , DROP PROFILE
                , CREATE SYNONYM
                , DROP SYNONYM
                , CREATE PROCEDURE
                , DROP PROCEDURE
                , ALTER PROCEDURE
;
AUDIT POLICY AUDIT_CIS_RECOMMENDATIONS
      WHENEVER SUCCESSFUL
;
```

Operationg Audit Policy

Activating Audit Poilcy

An audit policy object does not start auditing until it is activated.

An audit policy object should be activated by using AUDIT POLICY statement as follows to perform auditi ng.

```
CREATE AUDIT POLICY audit_t1_dml
       ACTIONS INSERT ON u1.t1
             , DELETE ON u1.t1
             , UPDATE ON u1.t1
;
Audit policy created.
AUDIT POLICY audit_t1_dml;
```

```
Audit succeeded.
```

When activating an audit policy, it audits new sessions only, but it does not affect the existing sessions.

When activating an audit policy by using AUDIT POLICY statement, it can specifies a user which will audit using BY clause or EXCEPT clause, or it may audit success/ failure of an audit action by using WHENEVER clause.

- BY | EXCEPT
  - BY user_list: It specifies a user to audit.
  - EXCEPT user_list: It audits all users except for specified users.
  - When it is omitted, it audits all users.
- WHENEVER
  - WHENEVER SUCCESSFUL: It creates an audit record when an audit action succeeds.
  - WHENEVER NOT SUCCESSFUL: It creates an audit record when an audit action fails.
  - When it is omitted, it creates an audit record regardless of a success/ failure of an audit action.

The information about activated audit policy can be viewed by executing AUDIT_POLICY_ENABLED view.

```
SELECT policy_name
     , enabled_opt
     , user_name
     , when_success
     , when_failure
  FROM audit_policy_enabled
 WHERE policy_name = 'AUDIT_T1_DML'
;
POLICY_NAME  ENABLED_OPT USER_NAME  WHEN_SUCCESS WHEN_FAILURE
------------ ----------- ---------- ------------ ------------
AUDIT_T1_DML BY          ALL USERS  YES          YES
```

If the auditing target user is omitted as the example above, it outputs ALL USERS meaning all users.

Note the followings when using BY clause and EXCEPT clause.

- BY clause and EXCEPT clause can not be used together for the same audit policy.

```
AUDIT POLICY audit_t1_dml BY u1;
Audit succeeded.
AUDIT POLICY audit_t1_dml EXCEPT u2;
ERR-42000(16475): audit policy already applied with the BY clause
```

- If multiple AUDIT POLICY BY clauses are used in the same audit policy, it is activated for the user sets.

In other words, the following examples have the same meaning.

◦ Example 1

```
AUDIT POLICY audit_t1_dml BY u1;
Audit succeeded.
AUDIT POLICY audit_t1_dml BY u2;
Audit succeeded.
SELECT policy_name
     , enabled_opt
     , user_name
     , when_success
     , when_failure
  FROM audit_policy_enabled
 WHERE policy_name = 'AUDIT_T1_DML'
;
POLICY_NAME  ENABLED_OPT USER_NAME WHEN_SUCCESS WHEN_FAILURE
------------ ----------- --------- ------------ ------------
AUDIT_T1_DML BY          U1        YES          YES
AUDIT_T1_DML BY          U2        YES          YES
2 rows selected.
```

◦ Example 2

```
AUDIT POLICY audit_t1_dml BY u1, u2;
Audit succeeded.
SELECT policy_name
     , enabled_opt
     , user_name
     , when_success
     , when_failure
  FROM audit_policy_enabled
 WHERE policy_name = 'AUDIT_T1_DML'
;
POLICY_NAME  ENABLED_OPT USER_NAME WHEN_SUCCESS WHEN_FAILURE
------------ ----------- --------- ------------ ------------
AUDIT_T1_DML BY          U1        YES          YES
AUDIT_T1_DML BY          U2        YES          YES
2 rows selected.
```

- If multiple AUDIT POLICY EXCEPT clauses are used in the same audit policy, only the last AUDIT POLICY statement is valid. In other words, the following examples have different meanings.

- Example 1

```
AUDIT POLICY audit_t1_dml EXCEPT u1;
Audit succeeded.
AUDIT POLICY audit_t1_dml EXCEPT u2;
Audit succeeded.
SELECT policy_name
     , enabled_opt
     , user_name
     , when_success
     , when_failure
  FROM audit_policy_enabled
 WHERE policy_name = 'AUDIT_T1_DML'
;
POLICY_NAME  ENABLED_OPT USER_NAME WHEN_SUCCESS WHEN_FAILURE
------------ ----------- --------- ------------ ------------
AUDIT_T1_DML EXCEPT      U2        YES          YES
1 row selected.
```

- Example 2

```
AUDIT POLICY audit_t1_dml EXCEPT u1, u2;
Audit succeeded.
SELECT policy_name
     , enabled_opt
     , user_name
     , when_success
     , when_failure
  FROM audit_policy_enabled
 WHERE policy_name = 'AUDIT_T1_DML'
;
POLICY_NAME  ENABLED_OPT USER_NAME WHEN_SUCCESS WHEN_FAILURE
------------ ----------- --------- ------------ ------------
AUDIT_T1_DML EXCEPT      U1        YES          YES
AUDIT_T1_DML EXCEPT      U2        YES          YES
2 rows selected.
```

- WHENEVER clause which is used together with BY clause is accumulated. In other words, the followi ng examples have the same meaning.

- Example 1

```
AUDIT POLICY audit_t1_dml BY u1 WHENEVER SUCCESSFUL;
Audit succeeded.
SELECT policy_name
     , enabled_opt
     , user_name
     , when_success
     , when_failure
  FROM audit_policy_enabled
 WHERE policy_name = 'AUDIT_T1_DML'
;
POLICY_NAME  ENABLED_OPT USER_NAME WHEN_SUCCESS WHEN_FAILURE
------------ ----------- --------- ------------ ------------
AUDIT_T1_DML BY          U1        YES          NO
1 row selected.
AUDIT POLICY audit_t1_dml BY u1 WHENEVER NOT SUCCESSFUL;
Audit succeeded.
SELECT policy_name
     , enabled_opt
     , user_name
     , when_success
     , when_failure
  FROM audit_policy_enabled
 WHERE policy_name = 'AUDIT_T1_DML'
;
POLICY_NAME  ENABLED_OPT USER_NAME WHEN_SUCCESS WHEN_FAILURE
------------ ----------- --------- ------------ ------------
AUDIT_T1_DML BY          U1        YES          YES
1 row selected.
```

- Example 2

```
AUDIT POLICY audit_t1_dml BY u1;
Audit succeeded.
SELECT policy_name
     , enabled_opt
     , user_name
     , when_success
     , when_failure
  FROM audit_policy_enabled
 WHERE policy_name = 'AUDIT_T1_DML'
;
```

```
POLICY_NAME  ENABLED_OPT USER_NAME WHEN_SUCCESS WHEN_FAILURE
------------ ----------- --------- ------------ ------------
AUDIT_T1_DML BY          U1        YES          YES
1 row selected.
```

- If WHENEVER clause is used together with EXCEPT clause, then only the last part of it is valid. In othe r words, the following examples have different meanings.
    - Example 1

```
AUDIT POLICY audit_t1_dml EXCEPT u1 WHENEVER SUCCESSFUL;
Audit succeeded.
AUDIT POLICY audit_t1_dml EXCEPT u1 WHENEVER NOT SUCCESSFUL;
Audit succeeded.
SELECT policy_name
     , enabled_opt
     , user_name
     , when_success
     , when_failure
  FROM audit_policy_enabled
 WHERE policy_name = 'AUDIT_T1_DML'
;
POLICY_NAME  ENABLED_OPT USER_NAME WHEN_SUCCESS WHEN_FAILURE
------------ ----------- --------- ------------ ------------
AUDIT_T1_DML EXCEPT      U1        NO           YES
1 row selected.
```

    - Example 2

```
AUDIT POLICY audit_t1_dml EXCEPT u1;
Audit succeeded.
SELECT policy_name
     , enabled_opt
     , user_name
     , when_success
     , when_failure
  FROM audit_policy_enabled
 WHERE policy_name = 'AUDIT_T1_DML'
;
POLICY_NAME  ENABLED_OPT USER_NAME WHEN_SUCCESS WHEN_FAILURE
------------ ----------- --------- ------------ ------------
AUDIT_T1_DML EXCEPT      U1        YES          YES
```

```
1 row selected.
```

Deactivating Audit Policy

NOAUDIT POLICY statement should be executed to deactivate an audit policy.
NOAUDIT POLICY statement is applied only to a new session, and it does not affect to the existing sessio n.

If all information is set to be deactivated by using the query below, then an audit policy is completely dea ctivated.

```
SELECT policy_name
     , enabled_opt
     , user_name
     , when_success
     , when_failure
  FROM audit_policy_enabled
 WHERE policy_name = 'AUDIT_T1_DML'
;
no rows selected.
```

NOAUDIT POLICY statement deletes each activated information which is created according to the specifie d AUDIT POLICY method.

The following is an example of deactivating only the auditing for u1 user of audit_t1_dml.

```
SELECT policy_name
     , enabled_opt
     , user_name
     , when_success
     , when_failure
  FROM audit_policy_enabled
 WHERE policy_name = 'AUDIT_T1_DML'
;
POLICY_NAME  ENABLED_OPT USER_NAME WHEN_SUCCESS WHEN_FAILURE
------------ ----------- --------- ------------ ------------
AUDIT_T1_DML BY          U1        YES          YES
AUDIT_T1_DML BY          SYS       YES          YES
2 rows selected.
NOAUDIT POLICY audit_t1_dml BY u1;
Noaudit succeeded.
SELECT policy_name
```

```
     , enabled_opt
     , user_name
     , when_success
     , when_failure
  FROM audit_policy_enabled
 WHERE policy_name = 'AUDIT_T1_DML'
;
POLICY_NAME  ENABLED_OPT USER_NAME WHEN_SUCCESS WHEN_FAILURE
------------ ---------- --------- ------------ ------------
AUDIT_T1_DML BY         SYS       YES          YES
1 row selected.
```

If AUDIT POLICY name BY clause is used it should be deactivated by using NOAUDIT POLICY name BY statement. If AUDIT POLICY name EXCEPT clause is used it should be deactivated by using NOAUDIT POLICY name statement without BY clause.

NOAUDIT POLICY statement should be used according to the usage as follows to deactivate each option of AUDIT POLICY statement.

Table 13-10 Activating/ deactivating audit policy

Type	AUDIT POLICY statement	NOAUDIT POLICY statement
All users	AUDIT POLICY p1	NOAUDIT POLICY p1
Using BY	AUDIT POLICY p1 BY u1	NOAUDIT POLICY p1 BY u1
Using EXCEPT	AUDIT POLICY p1 EXCEPT u1	NOAUDIT POLICY p1

If all users are activated as follows, NOAUDIT POLICY BY clause does not affect anything.

```
AUDIT POLICY audit_t1_dml;
Audit succeeded.
SELECT policy_name
     , enabled_opt
     , user_name
     , when_success
     , when_failure
  FROM audit_policy_enabled
 WHERE policy_name = 'AUDIT_T1_DML'
;
POLICY_NAME  ENABLED_OPT USER_NAME WHEN_SUCCESS WHEN_FAILURE
------------ ---------- --------- ------------ ------------
AUDIT_T1_DML BY         ALL USERS YES          YES
1 row selected.
NOAUDIT POLICY audit_t1_dml BY u1;
```

```
Noaudit succeeded.
SELECT policy_name
     , enabled_opt
     , user_name
     , when_success
     , when_failure
  FROM audit_policy_enabled
 WHERE policy_name = 'AUDIT_T1_DML'
;
POLICY_NAME  ENABLED_OPT USER_NAME WHEN_SUCCESS WHEN_FAILURE
------------ ---------- --------- ------------ ------------
AUDIT_T1_DML BY         ALL USERS YES          YES
1 row selected.
```

If one or more users are separately activated, NOAUDIT POLICY statement should be used according to th e AUDIT POLICY configuration.

- When it is activated by using BY clause
    - The following is an example of activating by using BY clause.

```
AUDIT POLICY p1 WHENEVER NOT SUCCESSFUL;
AUDIT POLICY p1 BY u1;
AUDIT POLICY p1 BY u2;
```

    - The information about activating is viewed as follows.

```
SELECT policy_name
     , enabled_opt
     , user_name
     , when_success
     , when_failure
  FROM audit_policy_enabled
 WHERE policy_name = 'P1';
POLICY_NAME ENABLED_OPT USER_NAME WHEN_SUCCESS WHEN_FAILURE
----------- ---------- --------- ------------ ------------
P1          BY         ALL USERS NO           YES
P1          BY         U1        YES          YES
P1          BY         U2        YES          YES
3 rows selected.
```

    - The following is the information about activating when NOAUDIT statement is performed.

```
NOAUDIT POLICY p1;
Noaudit succeeded.
SELECT policy_name
     , enabled_opt
     , user_name
     , when_success
     , when_failure
  FROM audit_policy_enabled
 WHERE policy_name = 'P1';
POLICY_NAME ENABLED_OPT USER_NAME WHEN_SUCCESS WHEN_FAILURE
----------- ----------- --------- ------------ ------------
P1          BY          U1        YES          YES
P1          BY          U2        YES          YES
2 rows selected.
```

In the example above, the auditing for ALL USERS is deactivated, but the auditing for user u1 and u2 are still activated.

If NOAUDIT POLICY statement is used again by using BY option as follows, then it completely deactivate t he audit policy p1.

```
NOAUDIT POLICY p1 BY u1, u2;
Noaudit succeeded.
SELECT policy_name
     , enabled_opt
     , user_name
     , when_success
     , when_failure
  FROM audit_policy_enabled
 WHERE policy_name = 'P1';
no rows selected.
```

- When it is activated by using EXCEPT
    - The following is an example of activating by using the audit policy.

```
AUDIT POLICY p1 EXCEPT u1, sys;
```

    - The information about activating is viewed as follows.

```
SELECT policy_name
     , enabled_opt
```

```
     , user_name
     , when_success
     , when_failure
  FROM audit_policy_enabled
 WHERE policy_name = 'P1';
POLICY_NAME ENABLED_OPT USER_NAME WHEN_SUCCESS WHEN_FAILURE
----------- ----------- --------- ------------ ------------
P1          EXCEPT      U1        YES          YES
P1          EXCEPT      SYS       YES          YES
2 rows selected.
```

- Unlike AUDIT POLICY statement, NOAUDIT POLICY statement does not have an EXCEPT option, and it executes the statement without an option as follows.

```
NOAUDIT POLICY p1;
Noaudit succeeded.
SELECT policy_name
     , enabled_opt
     , user_name
     , when_success
     , when_failure
  FROM audit_policy_enabled
 WHERE policy_name = 'P1';
no rows selected.
```

In other words, if an audit policy is activated by using an EXCEPT option, then a separate user can not be re activated by using NOAUDIT POLICY statement.

13.4 Authorization

Authorization-related Statements

For more information, refer to the followings.

- Creating a user: **CREATE USER**
- Dropping a user: **DROP USER**
- Altering a user: **ALTER USER**

- Granting privileges: **GRANT privileges TO**
- Revoking privileges: **REVOKE privileges FROM**

Information related to a user object and the authorization can be retrieved through the following views.

Table 13-11 Authorization objects related information

Schema	View	Description
	ALL_COL_PRIVS	Privileges for the user-accessible column
	ALL_COL_PRIVS_MADE	Privileges for the column whose user is the grantor
	ALL_COL_PRIVS_RECD	Privileges for the column whose user is the grantee
	ALL_DB_PRIVS	DB privileges which is related to a user
	ALL_DB_PRIVS_MADE	Privileges for the DB whose user is the grantor
	ALL_DB_PRIVS_RECD	Privileges for the DB whose user is the grantee
	ALL_PROC_PRIVS	Stored procedure/function privileges which is related to a user
	ALL_PROC_PRIVS_MADE	Privileges for the stored procedure/function whose user is the grantor
	ALL_PROC_PRIVS_RECD	Privileges for the stored procedure/function whose user is the grantee
	ALL_SCHEMA_PRIVS	Privileges for the user-accessible schema
	ALL_SCHEMA_PRIVS_MADE	Privileges for the schema whose user is the grantor
	ALL_SCHEMA_PRIVS_RECD	Privileges for the schema whose user is the grantee
	ALL_SEQ_PRIVS	Privileges for the user-accessible sequence
	ALL_SEQ_PRIVS_MADE	Privileges for the sequence whose user is the grantor
	ALL_SEQ_PRIVS_RECD	Privileges for the sequence whose user is the grantee
	ALL_TAB_PRIVS	Privileges for the user-accessible table
	ALL_TAB_PRIVS_MADE	Privileges for the table whose user is the grantor
	ALL_TAB_PRIVS_RECD	Privileges for the table whose user is the grantee
	ALL_TBS_PRIVS	Privileges for the user-accessible tablespace

Schema	View	Description
DICTIONARY_SCHEMA	ALL_TBS_PRIVS_MADE	Privileges for the tablespace whose user is the grantor
	ALL_TBS_PRIVS_RECD	Privileges for the tablespace whose user is the grantee
	ALL_USERS	Information about the user-accessible user
	USER_COL_PRIVS	Privilege information about the user owned column
	USER_COL_PRIVS_MADE	Information of granting privileges about the user owned column
	USER_COL_PRIVS_RECD	Information of acquiring privilege about the user owned column
	USER_PROC_PRIVS	Privilege information about the user owned stored procedure/function
	USER_PROC_PRIVS_MADE	Information of granting privileges about the user owned stored procedure/function
	USER_PROC_PRIVS_RECD	Information of acquiring privilege about the user owned stored procedure/function
	USER_SCHEMA_PRIVS	Privilege information about the user owned schema
	USER_SCHEMA_PRIVS_MADE	Information of granting privileges about the user owned schema
	USER_SCHEMA_PRIVS_RECD	Information of acquiring privilege about the user owned schema
	USER_SEQ_PRIVS	Privilege information about the user owned sequence
	USER_SEQ_PRIVS_MADE	Information of granting privileges about the user owned sequence
	USER_SEQ_PRIVS_RECD	Information of acquiring privilege about t the user owned sequence
	USER_TAB_PRIVS	Privileges information about the user owned table
	USER_TAB_PRIVS_MADE	Information of granting privileges about the user owned table
	USER_TAB_PRIVS_RECD	Information of acquiring privilege about the user owned table
	USER_USERS	Information about the current user
INFORMATION_SCHEMA	COLUMN_PRIVILEGES	Privilege information of user-accessible column
	ROUTINE_PRIVILEGES	Privilege information of user-accessible stored procedure/function
	TABLE_PRIVILEGES	Privilege information of user-accessible table
	USAGE_PRIVILEGES	Privilege information of user-accessible sequence

Concepts of User

A user object consists of the user's execution privilege set. The user should have the appropriate privilege to execute the SQL statements for the corresponding object.

For example, the user which is created by using **CREATE USER** statement is an object without any privileges. The user can not access the database nor does it execute any SQL statement. For access, the user should have CREATE SESSION ON DATABASE privilege which allows creating sessions in database object. The appropriate privilege should be granted after executing CREATE USER statement as follows.

```
CREATE USER u1 IDENTIFIED BY u1_password;
GRANT CREATE SESSION ON DATABASE TO u1;
COMMIT;
```

For more information about granting privilege after creating user object, refer to **Examples** in **CREATE USER** and **GRANT privileges TO**.

Creating Objects and Privileges

Creating SQL Schema Object

When creating an SQL schema object such as a table, the user should have the privileges for the superordinate non-schema object to which the table belongs in order to execute CREATE TABLE statements.

The following is an example of CREATE TABLE statements.

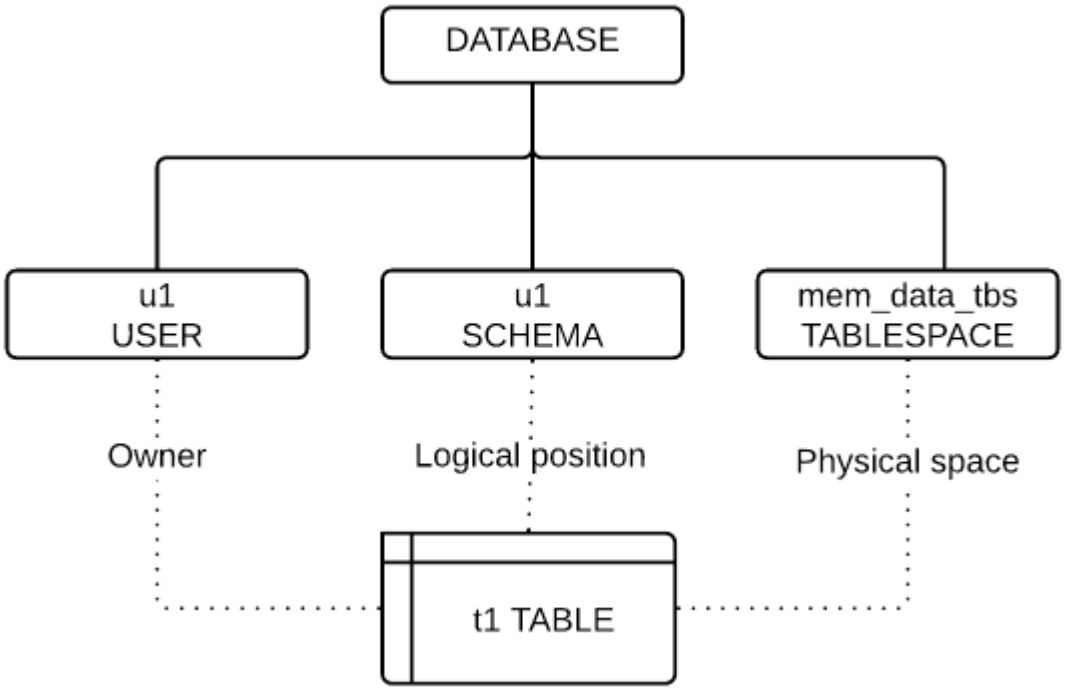
```
CREATE TABLE t1 ( id BIGINT, name VARCHAR(128) );
```

When it is interpreted as follows,

```
CREATE TABLE u1.t1 ( id BIGINT, name VARCHAR(128) ) TABLESPACE mem_data_tbs;
```

In the figure below, the owner of the table t1 is the account u1 who is the owner of the schema u1. The schema u1 is the logical location which includes the table, and the table space mem_data_tbs is the physical storage which stores the table.

Figure 2 CREATE TABLE and non-schema objects



In this case, the user u1 who performed the statement needs the following privileges for the schema and tablespace to which the table t1 will belong.

One of the following privileges is required to create the table which will be stored on its logical location (s chema u1).

- Privilege to create the table on the schema u1
  - CREATE TABLE ON SCHEMA ON u1
- Privilege to create any table on the database
  - CREATE ANY TABLE ON DATABASE
  - A table can be created when having privileges for the database which is superordinate object of t he schama u1 even without the schema privilege.

One of the following privileges is required to create a table in tablespace mem_data_tbs which is the phy sical storage space of tables.

- Privilege to create an object on the tablespace mem_data_tbs
  - CREATE OBJECT ON TABLESPACE mem_data_tbs
- Privilege to use all tablespaces in the database
  - USAGE TABLESPACE ON DATABASE
  - A table can be created when having privileges for the database which is the superordinate object of mem_data_tbs tablespace even without the tablespace privilege.

The owner of the table object is determined as follows.

- The owner of the schema to which the table belongs
- If the owner of the schema to which the table belongs is PUBLIC, then it is the user who executed the statement.

The owner of the table object has the following privileges to change the table structure and manipulate d ata in the table.

- SELECT ON TABLE
  - The privilege to execute SELECT statement on the table
- INSERT ON TABLE
  - The privilege to execute INSERT statement on the table
- UPDATE ON TABLE
  - The privilege to execute UPDATE statement on the table
- DELETE ON TABLE
  - The privilege to execute DELETE statement on the table
- LOCK ON TABLE
  - The privilege to execute LOCK statement on the table
- INDEX ON TABLE
  - The privilege to execute CREATE/DROP/ALTER INDEX statement on the table
- ALTER ON TABLE
  - The privilege to execute ALTER TABLE statement on the table

The grantor of the given privilege is the _SYSTEM account which is internally used, and the grantee is t he table owner. Therefore, any user including SYS account is not allowed to remove or change the owner privilege using **REVOKE privileges FROM**. The table owner's privileges are also removed when the table is removed.

Creating Non-schema Object

As like creating the SQL schema object such as a table, the appropriate privileges for the database (a sup erordinate object) are required to create the non-schema objects such as user, schema, tablespace.

A user who executes the following statements should have the following privileges for the database obje ct (a superordinate object) per each statement.

- CREATE USER statement
  - CREATE USER ON DATABASE
  - The privilege to execute CREATE USER statement on the database
- CREATE TABLESPACE statement
  - CREATE TABLESPACE ON DATABASE
  - The privilege to execute CREATE TABLESPACE statement on the database
- CREATE SCHEMA statement
  - CREATE SCHEMA ON DATABASE

    - The privilege to execute CREATE SCHEMA statement on the database
- CREATE PUBLIC SYNONYM statement
    - The privilege to execute CREATE PUBLIC SYNONYM statement on the database

Unlike creating the SQL schema object, the user who performed CREATE statement is not the owner of n on-schema object. Each object has the following characteristics.

- User object
    - There is not an owner.
    - DROP USER ON DATABASE privilege is required to drop a user object.
- Tablespace object
    - There is not an owner.
    - DROP TABLESPACE ON DATABASE privilege is required to drop a tablespace object.
- Schema object
    - It is an owner which is specified in the CREATE SCHEMA statement.
    - The schema object owner can drop it.
- Public synonym object
    - There is not an owner.
    - DROP PUBLIC SYNONYM ON DATABASE privilege is required to drop a public synonym object.

Privileges

Granting Privileges

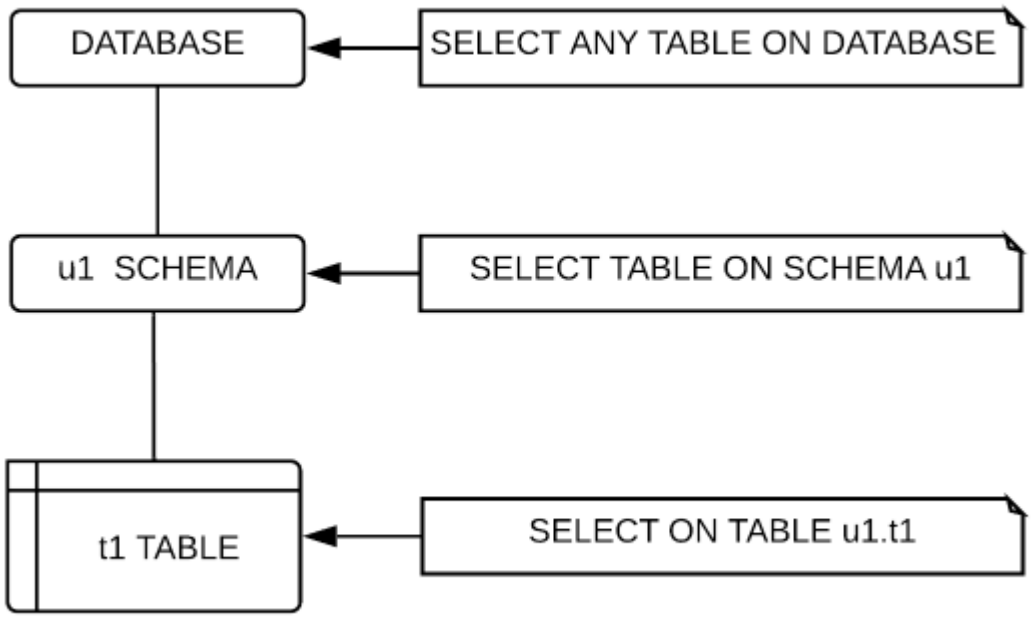
To INSERT, DELETE, UPDATE, or SELECT data when the user is not the owner of SQL schema object such as table, then the user should be granted the appropriate privileges by using **GRANT privileges TO** statem ent.

For example, a user who is not the owner of the table requires one of the following privileges to execute SELECT statement. As the figure below, the user can query the u1.t1 table if the user has SELECT privileg es on the superordinate object u1 schema or the database even when the user does not have SELECT priv ilege on table t1.

- The test user executes the following.

```
SELECT id, name FROM u1.t1 WHERE id < 100;
```

Figure 3 Privilege to execute SELECT statement



- SELECT ON TABLE u1.t1
  - SELECT privilege for the table u1.t1
- SELECT ON SCHEMA u1
  - SELECT privilege for the schema u1 which is the superordinate object
  - SELECT privilege for all tables of the schema u1
- SELECT ANY TABLE ON DATABASE
  - SELECT privilege for the database which is the second superordinate object
  - SELECT privilege for all tables of the database

The user who executed the GRANT statement should be one of the followings to grant the SELECT ON T ABLE u1.t1 privileges to another user.

- Owner of the object u1.t1
  - The object owner has the privilege of SELECT ON TABLE u1.t1 WITH GRANT OPTION granted by the _SYSTEM account.
- User with SELECT ON TABLE u1.t1 WITH GRANT OPTION
  - WITH GRANT OPTION enables the grantee to grant the granted privileges to another user.
- User with the privilege of ACCESS CONTROL ON DATABASE
  - The user can control all privileges on all objects.

The following is an example that the object owner grants privileges to another user.

- The owner of the table u1.t1 executes GRANT statement.

```
GRANT SELECT ON TABLE u1.t1 TO test;
```

For more information about the privilege types, refer to **GRANT privileges TO**.
For more information about privileges executing each SQL statements, refer to the Invocation and Access Rule of each statement in **SQL References** part.

Revoking Privileges

The granted privileges are revoked from a user by using **REVOKE privileges FROM**. The privileges of the object's owner can no be revoked until the object is removed.

For example, a user who can execute REVOKE statement to revoke the SELECT privilege from the test user are as follows. The user can not revoke privileges which were not granted by the user even when the user is the owner of the table.

```
REVOKE SELECT ON TABLE u1.t1 FROM test;
```

- User who grants privileges
  - The user grants the privilege of SELECT ON TABLE u1.t1 to the test1 user.
- User who has the privilege of ACCESS CONTROL ON DATABASE
  - The user can control all privileges on all objects.

When the test user is granted the same privileges from multiple users as follows, the test user can execute SELECT statement until all the privileges are revoked.

- The object owner u1 grants the privilege to the test user.

```
GRANT SELECT ON TABLE u1.t1 TO test;
```

- The object owner u1 grants the privilege of WITH GRANT OPTION to the user u2.

```
GRANT SELECT ON TABLE u1.t1 TO u2 WITH GRANT OPTION;
```

- The user u2 executes it.
- The u2 user with the privilege of WITH GRANT OPTION grants the privilege to the the test user.

```
GRANT SELECT ON TABLE u1.t1 TO test;
```

In the example above, the test user has two SELECT ON TABLE u1.t1 privileges which were granted by user u1 and user u2. The privilege information consists of {grantor, grantee, object privileges}.

PUBLIC Account

PUBLIC account is a special account which means every user.

For example, if the SELECT privilege is granted to PUBLIC account as follows, every user can execute SELECT statements on the table, u1.t1.

```
GRANT SELECT ON TABLE u1.t1 TO PUBLIC;
```

Even a user without SELECT privilege can execute SELECT statements on the table u1.t1 using the SELECT privilege on PUBLIC account.

When a privilege is granted to PUBLIC account, it is granted not to the existing users but to PUBLIC account itself. Even the newly created user can also execute SELECT statements.

Likewise, the user can execute SELECT statements if the user has the SELECT privilege on the table u1.t1 because revoking a privilege from PUBLIC account does not mean revoking the privilege from every user.

- The privilege is granted to the test user.

```
GRANT SELECT ON TABLE u1.t1 TO test;
```

- The privilege is granted to PUBLIC account.

```
GRANT SELECT ON TALBE u1.t1 TO PUBLIC;
```

- The privilege is revoked from PUBLIC account, and the test user still has the privilege on SELECT ON TABLE u1.t1.

```
REVOKE SELECT ON TABLE u1.t1 FROM PUBLIC;
```

Column Privilege

By granting privileges only on specific columns of a table, it is possible to control the execution of DML or SELECT statements of other user.

The following is an example table.

```
CREATE TABLE u1.t1
(
   id     BIGINT,
   name   VARCHAR(128),
   addr   VARCHAR(1024),
   salary NUMBER(20,0)
);
```

If SELECT privilege on the table u1.t1 excluding the salary column information is granted to another user, **GRANT privileges TO** statement is executed by listing the columns to be grantees of the privilege as follo ws. The test user who is the grantees of the privilege can not query the salary column.

- The privilege on SELECT columns is granted to the test user.

```
GRANT SELECT( id, name, addr ) ON TABLE u1.t1 TO test;
```

If the privilege on a table is granted as follows, the privilege on every column in the table is automatically granted. If both table privilege and column privilege are granted, the privilege information for a column is duplicated and the information is not dually managed.

- SELECT privilege is granted to the test user.

```
GRANT SELECT ON TABLE u1.t1 TO test;
```

- Privileges on every column in the table u1.t1 are automatically granted.

```
GRANT SELECT(id) ON TABLE u1.t1 TO test;
GRANT SELECT(name) ON TABLE u1.t1 TO test;
GRANT SELECT(addr) ON TABLE u1.t1 TO test;
GRANT SELECT(salary) ON TABLE u1.t1 TO test;
```

The privilege on a column is revoked by using **REVOKE privileges FROM** as follows.

- The privilege on SELECT columns is revoked from the test user.

```
REVOKE SELECT( id, name, addr ) ON TABLE u1.t1 FROM test;
```

If the privilege on a table is revoked as follows, the privilege on every column in the table is automatically revoked. Even if the column privileges and table privileges are separately granted, the columns privileges are revoked when the table privileges are revoked.

- SELECT privilege is revoked from the test user.
- Privileges on every column in the table u1.t1 are revoked.

```
REVOKE SELECT ON TABLE u1.t1 FROM test;
```

When revoking only the column privilege and the table privilege still exists as follows, then the following statements can be executed by using the table privilege. Therefore, to grant the privilege only on a specifi c column, the table privilege should be revoked, and then each column privilege should be granted.

- SELECT table privilege is granted to the test user.

```
GRANT SELECT ON TABLE u1.t1 TO test;
```

- The SELECT(salary) column privilege is revoked from the test user.
- The test user can query the salary column in the table u1.t1 because the test user has the SELECT priv ilege on the table.

```
REVOKE SELECT(salary) ON TABLE u1.t1 FROM test;
```

For more information about the privileges on a table and columns, refer to **GRANT privileges TO**.

13.5 Schema

Schema-related Statements

For more information about creating and dropping a schema, refer to the followings.

- Creating a schema: **CREATE SCHEMA**.
- Dropping a schema: **DROP SCHEMA**.

Information which is related to a schema object can be retrieved through the following views.

Table 13-12 Schema object-related information

Schema	View	Description
DICTIONARY_SCHEMA	**ALL_SCHEMAS**	User accessible schema information
	ALL_SCHEMA_PATH	User accessible schema path
	USER_SCHEMAS	User owned schema information
	USER_SCHEMA_PATH	User's schema path information
INFORMATION_SCHEMA	**SCHEMATA**	User accessible schema information

Concepts of Schema

The database consists of one or more schemas. The schema consists of objects such as tables, indexes, vi ews, which processes data. SQL schema object is an object which belongs to the schema.

Schema is similar to the directory in OS. The relationship between schema and tables is similar to the relat ionship between the directories and files in OS.
Schema is the logical position of the SQL schema object and it is criteria of distinguishing the name. Every SQL schema object should have a unique name and the naming space in the schema is as follows.

- Table, view, sequence, private synonym, stored procedure, stored function
- Index
- Constraint

The same name tables can be defined in the different schema. The different tables with same name are a ccessible and executed by specifying together with the schema name.

```
gSQL> SELECT u1.t1.id, u1.t1.name, u2.t1.addr
        FROM u1.t1, u2.t1
       WHERE u1.t1.id = u2.t1.id;
```

The name of SQL schema object is specified together with the schema name or without it. If the schema name is omitted, it is determined by the user's schema path. For more information about the schema pat h, refer to **Schema Path** clause.

- When the schema name is specified

```
gSQL> SELECT u1.t1.id, u1.t1.name FROM u1.t1;
```

- When the schema name is omitted

```
gSQL> SELECT t1.id, t1.name FROM t1;
```

User and Schema

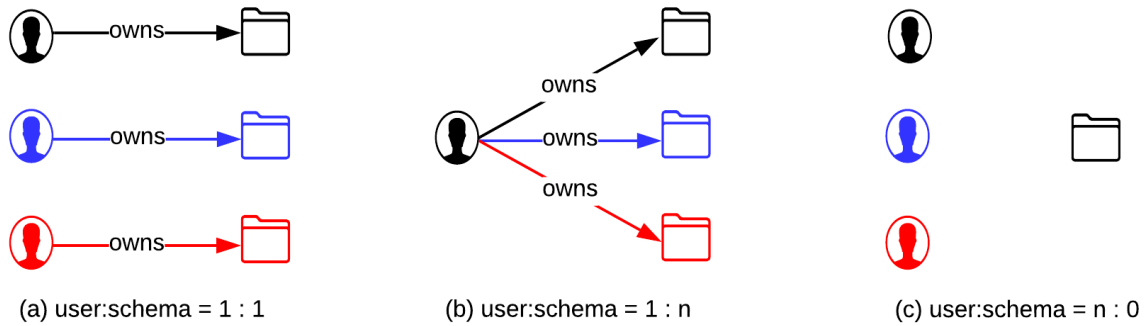
In GOLDILOCKS, relationship between the user and the schema is 1:N. A user does not own a schema, or the user can have multiple schemas.

The SQL standard does not explicitly define the relationship of the non-schema objects such as user, sche ma, database. Each DBMS defines the relationship of the non-schema objects in different ways, and they are as follows.

- Oracle
  - The relationship between the user and the schema is 1:1.
- DB2
  - The relationship between the user and the schema is 1:N.
- Postgres
  - The relationship between the user and the schema is 1:N.
- MySQL
  - The relationship between the database and the schema is 1:1.
  - User is a subordinate object of the schema (database).

When establishing the database, GOLDILOCKS configures various relationships between the users and sc hemas in accordance with the characteristics of the client system as follows.

Figure 4 Relationship between user and schema



To configure the database in which each user has its own schema as shown in the figure (a), create the u sers and schemas as follows.

- Create a user and the schema u1 together so that the user owns the schema.

```
gSQL> CREATE USER u1 IDENTIFIED BY u1_password WITH SCHEMA;
```

- WITH SCHEMA statement is optional and a schema whose name is as same as the user is created.

```
gSQL> CREATE USER u2 IDENTIFIED BY u2_password;
gSQL> CREATE USER u3 IDENTIFIED BY u3_password;
```

To configure the database in which a single user has multiple schemas as shown in the figure (b), create t he users and schemas as follows.

- Create the user u1 only without creating a schema.

```
gSQL> CREATE USER u1 IDENTIFIED BY u1_password WITHOUT SCHEMA;
```

- Create multiple schemas and specify the owner of each schema as a user u1.

```
gSQL> CREATE SCHEMA s1 AUTHORIZATION u1;
gSQL> CREATE SCHEMA s2 AUTHORIZATION u1;
gSQL> CREATE SCHEMA s3 AUTHORIZATION u1;
```

To configure the database in which multiple users share a single schema without creating a schema as sh own in the figure (c), create users as follows.
All users share **PUBLIC Schema** in the following examples.

- Create users only without creating a schema.

```
gSQL> CREATE USER u1 IDENTIFIED BY u1_password WITHOUT SCHEMA;
gSQL> CREATE USER u2 IDENTIFIED BY u2_password WITHOUT SCHEMA;
gSQL> CREATE USER u3 IDENTIFIED BY u3_password WITHOUT SCHEMA;
```

For more information about creating user and schema, refer to **CREATE USER**, **CREATE SCHEMA**.

Schema Path

Schema path is a path to find the schema name when the SQL schema object name such as a table is use d without the schema name. Schema path is similar to the PATH environment variable of Unix system. It i s similar in searching for a command in the PATH in the specified order when executing commands on U nix system.

When a user has multiple schemas, then creates or queries the table without a schema name as follows, t he schema path determines in which schema the table will be created.

- A table t1 is created in the schema s1.

```
gSQL> CREATE TABLE s1.t1 ( id INTEGER );
```

- A table t1 is created in the schema s2.

```
gSQL> CREATE TABLE s2.t1 ( name VARCHAR(128) );
```

- In which schema the table t1 will be created?

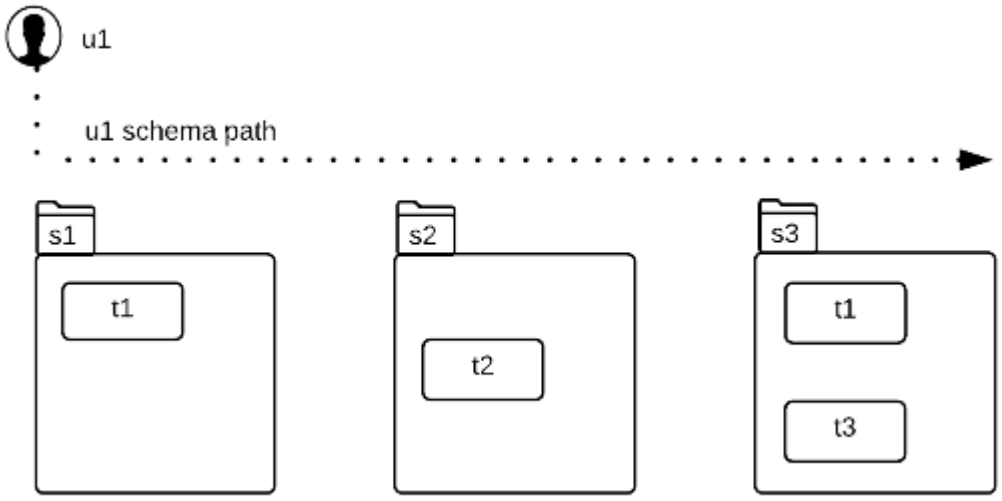
```
gSQL> CREATE TABLE t1 ( address VARCHAR(1024) );
```

- Which schema's table t1 is retrieved?

```
gSQL> SELECT * FROM t1;
```

The figure below illustrates an example of a schema path for user u1. The schema path of user u1 is desi gnated in an order of {s1, s2, s3}. Schema s1 has a table t1, schema s2 has a table t2, and the schema s3 has tables t1 and t3.

Figure 5 Example of schema path



In the SELECT statement without the schema name, the schema name is construed by the schema path a s follows.

- It is construed as table s1.t1 by the schema path.

```
gSQL> SELECT * FROM t1;
gSQL> SELECT * FROM s1.t1;
```

- It is construed as table s2.t2 by the schema path.

```
gSQL> SELECT * FROM t2;
gSQL> SELECT * FROM s2.t2;
```

- It is construed as table s3.t3 by the schema path.

```
gSQL> SELECT * FROM t3;
gSQL> SELECT * FROM s3.t3;
```

When omitting the schema name to retrieve the table s3.t1 as the example above, the table s3.t1 is deter mined by schema path, so the schema name should be specified as follows.

- It retrieves the table s1.t1 by the schema path.

```
gSQL> SELECT * FROM t1;
```

- The schema s3 should be specified to retrieve the table s3.t1.

```
gSQL> SELECT * FROM s3.t1;
```

The following CREATE TABLE statement in which the schema name is omitted creates a table in the first s chema s1 of the schema path.

- An error occurs because the identical table s1.t1 exists.

```
gSQL> CREATE TABLE t1 ( id INTEGER );
gSQL> CREATE TABLE s1.t1 ( id INTEGER );
```

- The table s2.t2 exists, but the table s1.t2 is created in different schemas.

```
gSQL> CREATE TABLE t2 ( name VARCHAR(128) );
gSQL> CREATE TABLE s1.t2 ( name VARCHAR(128) );
```

- The table s3.t3 exists, but the table s1.t3 is created in different schemas.

```
gSQL> CREATE TABLE t3 ( name VARCHAR(128) );
gSQL> CREATE TABLE s1.t3 ( name VARCHAR(128) );
```

If the current user omits the schema name when creating an object, the schema name to be used can be retrieved by using **CURRENT_SCHEMA** which is the SQL standard function.

```
gSQL> SELECT current_schema FROM dual;
CURRENT_SCHEMA
--------------
S1
1 row selected.
```

The schema path information of the current user can be retrieved by using ALL SCHEMA PATH view in DI CTIONARY SCHEMA schema.

```
gSQL> SELECT * FROM all_schema_path;
AUTH_NAME SCHEMA_NAME             SEARCH_ORDER
--------- ----------------------- ------------
U1        S1                                1
U1        S2                                2
U1        S3                                3
U1        PUBLIC                            4
PUBLIC    DICTIONARY_SCHEMA                 5
PUBLIC    INFORMATION_SCHEMA                6
PUBLIC    DEFINITION_SCHEMA                 7
```

```
PUBLIC    PERFORMANCE_VIEW_SCHEMA           8
PUBLIC    FIXED_TABLE_SCHEMA                9
9 rows selected.
```

In the example above, the schema path of the user u1 is in an order of {s1, s2, s3, public}, and the schem a path of PUBLIC account is in an order of {DICTIONARY_SCHEMA, INFORMATION_SCHEMA, DEFINITION _SCHEMA, PERFORMANCE_VIEW_SCHEMA, FIXED_TABLE_SCHEMA}.
If omitting the schema name and retrieving a table t1, then it searches for the schema path for the curren t user u1. If the schema path does not exist in it, then it searches for the schema path of PUBLIC account.

Schema path of a specific user can be altered by using **ALTER USER** statement as follows. The schema pat h of PUBLIC account can be altered by using ALTER USER PUBLIC SCHEMA PATH statement.
CURRENT PATH clause is used to additionally alter other schemas together with the user's current schem a.

- The schema path of user u1 is altered.

```
gSQL> ALTER USER u1 SCHEMA PATH ( s3, s2, s1 );
User altered.
```

- The schema path of PUBLIC account is altered.

```
gSQL> ALTER USER PUBLIC SCHEMA PATH ( s1, s2, s3 );
User altered.
```

- The schema path which includes the current schema path is altered by using CURRENT PATH.

```
gSQL> ALTER USER u1 SCHEMA PATH ( s4, CURRENT PATH );
User altered.
```

The schema path is automatically determined when a user is created by using **CREATE USER** statement. A nd if the user schema is created by using **CREATE SCHEMA** statement later, then it is not automatically in cluded in the user's schema path, so, if necessary, it should be included in the schema path by using **ALTE R USER**. For more information, refer to each statement.

PUBLIC Schema

PUBLIC schema is a shared schema in which any user can create objects. As shown in the example below, if the user who does not have the schema creates the table whose schema name is not specified, then th e table's schema is PUBLIC.

- A user u1 who does not have the schema is created.

```
gSQL> CREATE USER u1 IDENTIFIED BY u1_password WITHOUT SCHEMA;
gSQL> GRANT CREATE SESSION TO u1;
gSQL> GRANT CREATE OBJECT ON TABLESPACE mem_data_tbs TO u1;
```

- The user u1 creates a table.

```
% gsql u1 u1_password
gSQL> CREATE TABLE t1 ( id INTEGER );
```

- The statements above have the same meaning as the following.

```
gSQL> CREATE TABLE public.t1 ( id INTEGER );
```

In the example above, the table t1's schema which is created by user u1, is PUBLIC. PUBLIC schema is a built-in schema which is automatically created when creating the database.
It is granted the privilege of the same meaning as the following statement so that any user can create an object in the schema.

```
gSQL> GRANT CREATE TABLE, CREATE VIEW, CREATE INDEX, CREATE SEQUENCE, ADD CONSTRAINT
         ON SCHEMA PUBLIC
         TO PUBLIC;
```

PUBLIC schema, shared schema, is different from PUBLIC account which means all users. In the statement above, the privilege of creating objects in the PUBLIC schema (ON SCHEMA PUBLIC) is granted to PUBLIC account (TO PUBLIC) which means all users.
Any user can create or manage tables, but an appropriate privilege is required for retrieving the table created in PUBLIC schema by another user.

The followings are examples of GRANT statements using PUBLIC account and PUBLIC schema.

- SELECT privilege on table u1.t1 is granted to PUBLIC account.
- Any user can retrieve the table u1.t1.

```
gSQL> GRANT SELECT ON TABLE u1.t1 TO PUBLIC;
```

- SELECT privilege on all tables in PUBLIC schema is granted to user u1.
- The user u1 can retrieve all tables in PUBLIC schema.

```
gSQL> GRANT SELECT TABLE ON SCHEMA PUBLIC TO u1;
```

User and Examples of Using Schema

For example, if a single administrator and multiple developers use a single schema, the schema can be co ntrolled by using the following SQL statement.

The following examples describe how to create a mgr_usr user who manages our_schema schema, and a number of users such as app_user1, app_user2, app_user3 who develops applications using our_schema.

A mgr_user user and a our_schema schema are created as follows.

- mgr_user and our_schema are created.
- mgr_user owns our_schema.

```
gSQL> CREATE USER mgr_user IDENTIFIED BY mgr_user WITH SCHEMA our_schema;
User created.
```

- An appropriate privilege is granted to mgr_user.

```
gSQL> GRANT ALL PRIVILEGES ON DATABASE TO mgr_user;
Grant succeeded.
gSQL> COMMIT;
Commit complete.
```

Multiple app_user users are created as follows.
The app_user users are allowed to execute only SELECT and DML statements on our_schema.

- Multiple app_user users who do not own the schema are created as follows.

```
gSQL> CREATE USER app_user1 IDENTIFIED BY app_user1 WITHOUT SCHEMA;
User created.
gSQL> CREATE USER app_user2 IDENTIFIED BY app_user2 WITHOUT SCHEMA;
User created.
gSQL> CREATE USER app_user3 IDENTIFIED BY app_user3 WITHOUT SCHEMA;
User created.
gSQL> COMMIT;
Commit complete.
```

- The access privileges are granted to multiple app_user users.

```
gSQL> GRANT CREATE SESSION ON DATABASE TO app_user1, app_user2, app_user3;
Grant succeeded.
gSQL> COMMIT;
```

```
Commit complete.
```

- Only read/write privileges on our_schema are granted to multiple app_user users.

```
gSQL> GRANT SELECT TABLE, INSERT TABLE, UPDATE TABLE, DELETE TABLE ON SCHEMA our_schema TO
app_user1, app_user2, app_user3;
Grant succeeded.
gSQL> COMMIT;
Commit complete.
```

- SCHEMA_PATH of multiple app_user users are specified as our_schema.

```
gSQL> ALTER USER app_user1 SCHEMA PATH ( our_schema );
User altered.
gSQL>  ALTER USER app_user2 SCHEMA PATH ( our_schema );
User altered.
gSQL> ALTER USER app_user3 SCHEMA PATH ( our_schema );
User altered.
gSQL> COMMIT;
Commit complete.
```

Through the operations above, mgr_user has DDL privileges of creating/ dropping/ altering an object in o ur_schema, but multiple app_user can only read/write operations on the tables in our_schema.

mgr_user can perform management tasks such as creating a table as follows.

```
gSQL> \connect mgr_user mgr_user
gSQL> CREATE TABLE t1 ( c1 INTEGER );
Table created.
gSQL> INSERT INTO t1 VALUES ( 1 );
1 row created.
gSQL> INSERT INTO t1 VALUES (2), (3);
2 rows created.
gSQL> COMMIT;
Commit complete.
```

Multiple app_user can read/write operation for the table in our_schema without specifying the schema n ame, but they are not allowed to create or drop an object as follows.

```
gSQL> \connect app_user1 app_user1
gSQL> SELECT * FROM t1;
C1
--
```

```
 1
 2
 3
3 rows selected.
gSQL> INSERT INTO t1 VALUES (4);
1 row created.
gSQL> COMMIT;
Commit complete.
gSQL> DROP TABLE t1;
ERR-42000(16208): insufficient privileges
```

13.6 Tablespace

Tablespace-related Statements

For more information, refer to the followings.

- Creating tablespace
  - **CREATE TABLESPACE**
  - **CREATE MEMORY DATA TABLESPACE**
  - **CREATE MEMORY TEMPORARY TABLESPACE**
  - **CREATE DISK DATA TABLESPACE**

- Dropping tablespace: **DROP TABLESPACE**

- Altering tablespace
  - **ALTER TABLESPACE**
  - **ALTER TABLESPACE name RENAME TO**
  - **ALTER TABLESPACE name BACKUP**
  - **ALTER TABLESPACE name [ONLINE|OFFLINE]**

- Adding, dropping, altering the datafile which configures the tablespace
  - **ALTER TABLESPACE name ADD [DATAFILE|MEMORY]**
  - **ALTER TABLESPACE name DROP [DATAFILE|MEMORY]**
  - **ALTER TABLESPACE name RENAME DATAFILE**
  - **ALTER DATABASE DATAFILE AUTOEXTEND**

Information which is related to a tablespace object can be retrieved through the following views.

Table 13-13 Tablespace object related information

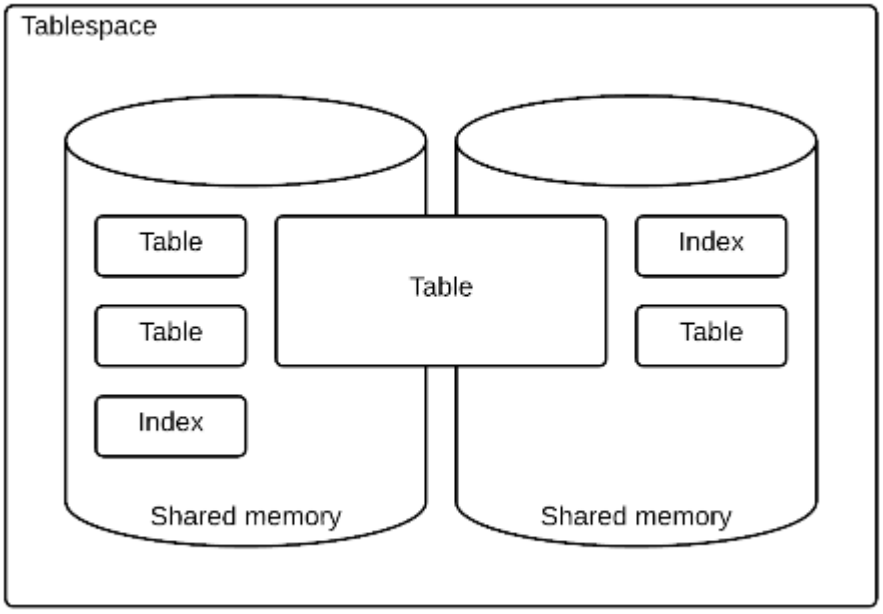
Schema	View	Description
DICTIONARY_SCHEMA	**USER_TABLESPACES**	Information of user accessible tablespaces

Concepts of Tablespace

A tablespace is a logical concept and it consists of one or more physical shared memory. It is a space to st ore data, such as tables, indexes.
Physical objects such as tables, indexes which are stored in a tablespace can be spanned multiple shared memories as shown below.

A tablespace can be extended by adding the shared memory.

Figure 6 Concept of tablespace



Tablespaces are classified into three types depending on the stored data type as follows.

- DATA TABLESPACE
  - It is tablespace to store and manage permanent data such as tables, logging indexes.
- TEMPORARY TABLESPACE
  - It is tablespace to store and manage volatile data such as the indexes without logging, the hash which is generated during query execution and the sort.
- UNDO TABLESPACE
  - It is tablespace to store and manage the data change information for rollback the transaction.

Tables, indexes (LOGGING) stored in DATA tablespace create redo logs to permanently manage data. Ho wever, indexes without logging stored in TEMPORARY tablespace does not create redo logs. The index w ithout logging does not log the changes. When restarting the system, it is rebuilt based on the table data, and the index facility is retained.

A tablespace is a physical storage in which SQL schema objects are stored. A particular tablespace can be specified when creating a table and index. A table, the index which is related to the table, and the indexe s which is created for the constraint condition of the table can be stored in different tablespaces.
For more information about specifying the tablespace when creating an object, refer to the followings.

- **CREATE TABLE**
- **CREATE INDEX**

- ALTER TABLE name ADD CONSTRAINT

If a tablespace is not specified when creating an object such as a table or index, the default tablespace is used.
For more information about the user's default tablespace, refer to the followings.

- CREATE USER
- ALTER USER

For more information about tablespace, refer to **Managing Tablespace**.

13.7 Table

Table-related Statements

Statements for creating, dropping, altering a table are as follows.

- Creating table
  - CREATE TABLE
  - CREATE TABLE AS SELECT
  - CREATE GLOBAL TEMPORARY TABLE
  - CREATE IMMUTABLE TABLE

- Dropping table
  - DROP TABLE
  - TRUNCATE TABLE

- Altering table
  - ALTER TABLE
  - ALTER TABLE name RENAME TO
  - ALTER TABLE name STORAGE

- Adding, dropping, altering a column of the table
  - ALTER TABLE name ADD COLUMN
  - ALTER TABLE name SET UNUSED COLUMN
  - ALTER TABLE name ALTER COLUMN
  - ALTER TABLE name RENAME COLUMN

- Adding, dropping, altering a constraint of the table
  - ALTER TABLE name ADD CONSTRAINT
  - ALTER TABLE name DROP CONSTRAINT
  - ALTER TABLE name ALTER CONSTRAINT

- Adding, dropping additional log of the table
  - ALTER TABLE name ADD SUPPLEMENTAL LOG
  - ALTER TABLE name DROP SUPPLEMENTAL LOG

- Table statistics information: ANALYZE TABLE.

- Managing recycle bin of the table
  - FLASHBACK TABLE

- PURGE

Information which is related to a table object can be retrieved through the following views.

Table 13-14 Table object related information

Schema	View	Description
DICTIONARY_SCHEMA	ALL_ALL_TABLES	Information about user accessible tables
	ALL_COL_COMMENTS	Information about comments of user accessible column
	ALL_CONSTRAINTS	Information about user accessible constraints
	ALL_CONS_COLUMNS	Column information about user accessible constraints
	ALL_TABLES	Information about user accessible tables
	ALL_TAB_COLS	Information about user accessible columns
	ALL_TAB_COLUMNS	Information about user accessible columns
	ALL_TAB_COMMENTS	Information about comments of user accessible tables
	ALL_TAB_IDENTITY_COLS	Identity column information about user accessible tables
	USER_ALL_TABLES	Information about user owned tables
	USER_COL_COMMENTS	Information about comments of user owned columns
	USER_CONSTRAINTS	Information about user owned constraints
	USER_CONS_COLUMNS	Column information about user owned constraints
	USER_RECYCLEBIN	Information about user owned recycle bin object
	USER_TABLES	Information about user owned tables
	USER_TAB_COLS	Information about user owned columns
	USER_TAB_COLUMNS	Information about user owned columns
	USER_TAB_COMMENTS	Information about comments of user owned tables
	USER_TAB_IDENTITY_COLS	Identity column information about user owned tables
INFORMATION_SCHEMA	COLUMNS	Information about user accessible columns
	CONSTRAINT_COLUMN_USAGE	Column information about user accessible constraints
	CONSTRAINT_TABLE_USAGE	Table information about user accessible constraints
	KEY_COLUMN_USAGE	Column information about user accessible key constraints
	TABLES	Information about user accessible tables
	TABLE_CONSTRAINTS	Information about user accessible constraints

Concepts of Table

Table is an underlying object which configures the database. In the SQL standard, the table is called as a base table, and the view is called as a viewed table.

The table consists of columns and rows. The table consists of multiple rows, the number and order of columns in each row is same.
The table consists of one or more columns, and each column has a name but a row does not have a name. The order of the row is not always as same as the order in which data is added.
The value is the data in intersection of a column and a row. Column is the set of value which have the same data type.

Each column of a table has a unique name to be distinguished from other columns in the table. It has a data type which corresponds to the characteristics of the value. For more information about data type, refer to **Data Type** section.
The constraints can be added to the table for data integrity. For more information about constraint, refer to **CREATE TABLE**, **ALTER TABLE name ADD CONSTRAINT**.
The table index can be created to improve performance of queries. For more information about index, refer to **Index** section.

The following is an example of creating a lineitem table using **CREATE TABLE** statement.

```
CREATE TABLE lineitem
(
    l_orderkey      INTEGER    NOT NULL
  , l_partkey       INTEGER    NOT NULL
  , l_suppkey       INTEGER    NOT NULL
  , l_linenumber    INTEGER    NOT NULL
  , l_quantity      NUMERIC(12,2)
  , l_extendedprice NUMERIC(12,2)
  , l_discount      NUMERIC(12,2)
  , l_tax           NUMERIC(12,2)
  , l_returnflag    CHAR(1)    NOT NULL  DEFAULT 'F'
  , l_linestatus    CHAR(1)
  , l_shipdate      DATE
  , l_commitdate    DATE
  , l_receiptdate   DATE
  , PRIMARY KEY (l_orderkey, l_linenumber) INDEX lineitem_pk_idx TABLESPACE mem_temp_tbs
) TABLESPACE mem_data_tbs;
```

In the example above, multiple columns and constraints are defined in the lineitem table. When using constraints to define columns, NOT NULL constaints are defined for the columns l_orderkey, l_partkey, l_suppkey, l_linenumber.
PPRIMARY KEY constraint is defined by combining columns of l_orderkey, l_linenumber.
In-line constraint is described together with the column definitions. Out-line constraint is described separately from the column definitions.

Using DEFAULT clause in the column l_returnflag, the value 'F' is declared as a default value for the column. The index which is created when creating PRIMARY KEY constraints is named separately as lineitem_pk_idx. The tablespace in which the index will be stored is named as mem_temp_tbs. The table is physically stored in the tablespace mem_data_tbs.

The following is an example of adding a constraint to a table using **ALTER TABLE name ADD CONSTRAINT** statement.

```
ALTER TABLE lineitem
      ADD CONSTRAINT lineitem_unique_all_key
      UNIQUE( l_orderkey ASC, l_partkey DESC, l_suppkey DESC, l_linenumber ASC);
```

In the example above, UNIQUE constraint is added to the lineitem table, and the column sort order ASC/DESC is specified for the automatically created index of constraints.

The following is an example of adding columns to the table using **ALTER TABLE name ADD COLUMN** statement.

```
ALTER TABLE lineitem ADD COLUMN
(
    l_shipinstruct  CHAR(25)
  , l_shipmode      CHAR(10)
  , l_comment       VARCHAR(44)
);
```

In the example above, multiple columns are added to a table. The in-line constraints or default values can be specified when adding columns.

The following is an example of creating an index in the table using **CREATE INDEX** statement.

```
CREATE INDEX lineitem_idx_shipdate ON lineitem( l_shipdate ASC NULLS LAST );
```

The example above describes the index which is created for the column l_shipdate which is often used as the query conditions. The column sort order is ascending (ASC), and if NULL value exists, it is specified to be located at the end.

For more information about DML statements of inserting/deleting/updating data to a table, refer to **Data Manipulation Language** clause.
For more information about SELECT statements of querying data in a table, refer to **Data Query Language** clause, **SELECT** statement.

Global Temporary Table

Global temporary table is a kind of temporary table and the table definition is shared by all users, but the data is separated per section.

The table definition is created when executing CREATE GLOBAL TEMPORARY TABLE, but the physical se gment is created in a state which is dependent on a session when executing INSERT to that table for the f irst time. The segments which are allocated to all global temporary tables created in the session are releas ed when the session is terminated. An option defined at the time of creating a global temporary table det ermines whether to truncate the data left after committing or rolling back.

It supports all DDL and DML provided by a table except for cluster-related statements. A DDL statement r eturns an error to the global temporary table which is used by the current session. However, TRUNCATE T ABLE statement for a global temporary table is applied only to the current session, so it does not return a n error even when it is used by another session.

A global temporary table can be defined only in a temporary tablespace, so it does not record a redo log f or restart recovery. However, it records the undo log for MVCC and rollback, and the space on which the undo log is recorded can be selected to system undo tablespace or system temp tablespace by using **TEM P_UNDO_ENABLED**.

If TEMP_UNDO_ENABLED is 1, it records undo logs in a temp undo relation of the session separately fro m the undo relation of the transaction. If the transaction performs only DML for a global temporary table, then it does not record the transaction record nor the commit log, so it improves the DML performance.

When releasing the segment used in the session, it is returned to the corresponding tablespace, and it is allocated again from the tablespace when allocating again. The process allocating and returning segment s in a tablespace costs a lot to keep concurrency with other sessions and to allocate and release segments. Therefore, the segment which was released after used in a session can be reused without returning it by using **TEMP_SEGMENT_CACHE_SIZE**.

In other words, if TEMP_SEGMENT_CACHE_SIZE is set to 0 (default value), then it immediately returns th e segement which is released after used to the tablespace. If TEMP_SEGMENT_CACHE_SIZE is set to the value bigger than 1 (maximum 4294967295), then it reuses as many segments as set in a session when r eturning the segment.

When a global temporary table is not used in a session any more, then cleanup segments in a segment ca che at once by using **ALTER SESSION CLEANUP GLOBAL TEMPORARY SEGMENT POOL;**.

The following is an example of creating a global temporary table by using **CREATE GLOBAL TEMPORARY TABLE**.

```
CREATE  GLOBAL TEMPORARY TABLE SESSION_TABLE1(
        COL1    CHAR(10)
       ,COL2    VARCHAR2(20)
       ,COL3    NUMBER(10)
)   ON  COMMIT  DELETE ROWS;
```

The information about the created global temporary table can be viewed in DICTIONARY tables or views i n the same way as viewing the information of an ordinary table.

Table Function Derived Table

The table function derived table is a logical table which consists of the result sets returned by performing the table function. The table function is a function which is defined as the return table type. The definitio n of the table function derived table follows the table column list definition defined in the table function 's return statement. Unlike other general tables, the table function derived table can not have an index n or a constraint. The row of the table function derived table is the result set to which the table function is r eturned. The table function derived table is useful when retrieving the desired data from the desired table by configuring the sub sets of a specific table.
For more information about the table function, refer to **Stored Function**.

Figure 7 Definition of table function derived table

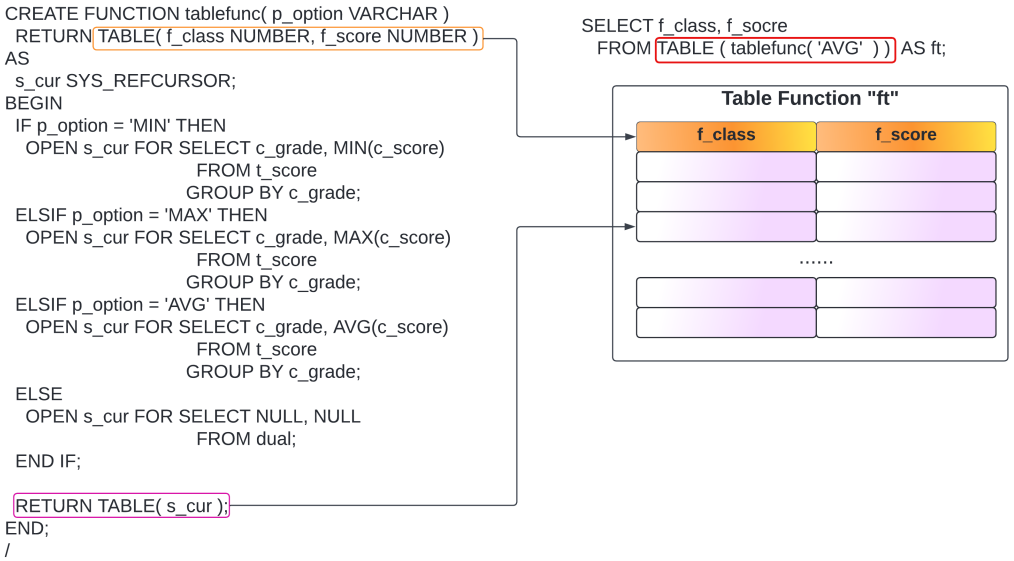


Table in Cluster

For more information about a table in the cluster environment, refer to **Cluster Table and Shard**.

Managing Recycle Bin of Table

Syntax

- Restoring an object stored in the recyclebin.
  - **FLASHBACK TABLE**
- Dropping an object stored in the recyclebin.
  - **PURGE**

The information about the recycle bin can be retrieved through the following views.

Table 13-15 Information about the recyclebin

Schema	View	Description
DICTIONARY_SCHEMA	**DBA_RECYCLEBIN**	Information about all recyclebins in the database
	USER_RECYCLEBIN	Information about the recycle bin which is owned by itself
	RECYCLEBIN	Alias of USER_RECYCLEBIN

Description

It stores the dropped object in the recycle bin instead of completely dropping it. Constraints and indexes r elated to the table are also stored in the recyclebin.

The concept of the recycle bin is also called as *flashback drop* feature, and objects stored in the recycle bi n can be dropped or be restored by using PURGE or FLASHBACK TABLE statement.

If a table is dropped and stored in the recyclebin, then all names of objects related to the table are altered and stored. The altered name is in a form of BIN$unique_name, and the database creates and gives the u nique values. unique_name is created in a value of 32 characters.

When restoring tables stored in the recycle bin, constraints and indexes related to the table are restored i n its original of when before they were dropped. However, if a name of when before the object was drop ped already exists, then the object is restored in the name of when it is stored in the recycle bin.

RECYCLEBIN property should be activated to use the recycle bin feature. The property can be altered with ALTER SESSION and ALTER SYSTEM, and ALTER SYSTEM has DEFERRED property. The default value is FA LSE.

```
gSQL> ALTER SESSION SET RECYCLEBIN = ON;
Session altered.
gSQL> ALTER SYSTEM SET RECYCLEBIN = ON DEFERRED;
System altered.
```

Feature

- The object stored in the recycle bin is not physically dropped. Therefore the tablespace used by the co rresponding object is not released.
- Each user has its own recycle bin and a user can view its own recycle bin.
- If there are multiple tables with the same name, then the oldest object is dropped when dropping an object stored in the recycle bin.
- If there are multiple tables with the same name, then the newest object is restored when restoring an object stored in the recycle bin.
- DICTIONARY VIEW can retrieve objects stored in the recycle bin, but INFORMATION VIEW can not ret rieve objects stored in the recycle bin.
- PURGE DBA_RECYCLEBIN ON DATABASE privilege is required to drop all recycle bins in the database.
- PURGE and FLASHBACK TABLE statements can be audited with AUDIT SYSTEM ACTION.
- If a tablespace is dropped, then the recycle bins included in the tablespace are also dropped.
- If a schema is dropped, then the recycle bins included in the schema are also dropped.
- If a user is dropped, then the user's recycle bin is also dropped.

Only some DML and DDL are allowed for the object stored in the recycle bin, and statements ex cept for the statements below cause an error.

- SELECT
- SELECT .. FOR UPDATE
- LOCK TABLE
- COMMENT ON TABLE name IS
- COMMENT ON COLUMN name IS
- COMMENT ON INDEX name IS
- COMMENT ON CONSTRAINT name IS
- GRANT privileges TO
- REVOKE privileges FROM
- CREATE TABLE AS SELECT
- CREATE GLOBAL TEMPORARY TABLE AS SELECT
- CREATE AUDIT POLICY
- ALTER AUDIT POLICY
- CREATE VIEW
- CREATE SYNONYM
- CREATE FUNCTION

- CREATE PROCEDURE
- ALTER FUNCTION
- ALTER PROCEDURE
- ALTER DATABASE MOVE SHARD
- ALTER DATABASE REBALANCE
- ALTER DATABASE REBALANCE EXCLUDE CLUSTER GROUP
- ALTER TABLE name REBALANCE
- ALTER TABLE name REBALANCE EXCLUDE CLUSTER GROUP
- ALTER TABLE name MOVE SHARD
- ALTER TABLE name SPLIT SHARD
- ALTER TABLE name MERGE SHARD
- ALTER TABLE name SYNCHRONIZE IDENTITY COLUMN

Example

RECYCLEBIN property should be activated to use the recycle bin feature.

```
gSQL> ALTER SESSION SET RECYCLEBIN = ON;
Session altered.
gSQL> CREATE TABLE t1 ( id INTEGER PRIMARY KEY, name VARCHAR(32) );
Table created.
gSQL> DROP TABLE t1;
Table dropped.
gSQL> CREATE TABLE t1 ( id INTEGER PRIMARY KEY, name VARCHAR(32) );
Table created.
gSQL> DROP TABLE t1;
Table dropped.
gSQL> COMMIT;
Commit complete.
gSQL> SELECT OBJECT_NAME, ORIGINAL_NAME, OBJECT_TYPE, DROPPED_TIME FROM USER_RECYCLEBIN;
OBJECT_NAME                          ORIGINAL_NAME        OBJECT_TYPE DROPPED_TIME
------------------------------------ -------------------- -----------
-------------------------
BIN$8981D28E172C11EAA7C5D51B86D72AB6 T1                   TABLE       2019-12-05
15:57:44.120000
BIN$8981D2C0172C11EAA7C5D51B86D72AB6 T1_PRIMARY_KEY       CONSTRAINT  2019-12-05
15:57:44.120000
BIN$8981D2AC172C11EAA7C5D51B86D72AB6 T1_PRIMARY_KEY_INDEX INDEX       2019-12-05
15:57:44.120000
BIN$8F1E9614172C11EAA7C5D51B86D72AB6 T1                   TABLE       2019-12-05
```

```
15:57:53.540000
BIN$8F1E9650172C11EAA7C5D51B86D72AB6 T1_PRIMARY_KEY       CONSTRAINT  2019-12-05
15:57:53.540000
BIN$8F1E963C172C11EAA7C5D51B86D72AB6 T1_PRIMARY_KEY_INDEX INDEX       2019-12-05
15:57:53.540000
6 rows selected.
gSQL> PURGE TABLE t1;
Table purged.
gSQL> SELECT OBJECT_NAME, ORIGINAL_NAME, OBJECT_TYPE, DROPPED_TIME FROM USER_RECYCLEBIN;
OBJECT_NAME                          ORIGINAL_NAME        OBJECT_TYPE DROPPED_TIME
------------------------------------ -------------------- -----------
--------------------------
BIN$8F1E9614172C11EAA7C5D51B86D72AB6 T1                   TABLE       2019-12-05
15:57:53.540000
BIN$8F1E9650172C11EAA7C5D51B86D72AB6 T1_PRIMARY_KEY       CONSTRAINT  2019-12-05
15:57:53.540000
BIN$8F1E963C172C11EAA7C5D51B86D72AB6 T1_PRIMARY_KEY_INDEX INDEX       2019-12-05
15:57:53.540000
3 rows selected.
gSQL> FLASHBACK TABLE t1 TO BEFORE DROP;
Flashback complete.
gSQL> DESC T1
COLUMN_NAME TYPE         IS_NULLABLE
----------- ------------ -----------
ID          NUMBER(10,0) FALSE
NAME        VARCHAR(32)  TRUE
INDEX_NAME           TABLESPACE_NAME INDEX_TYPE IS_UNIQUE COLUMNS
-------------------- --------------- ---------- --------- -------
T1_PRIMARY_KEY_INDEX MEM_TEMP_TBS    BTREE      TRUE      ID
CONSTRAINT_NAME CONSTRAINT_TYPE ASSOCIATED_INDEX     COLUMNS
--------------- --------------- -------------------- -------
T1_PRIMARY_KEY  PRIMARY KEY     T1_PRIMARY_KEY_INDEX ID
gSQL> SELECT OBJECT_NAME, ORIGINAL_NAME, OBJECT_TYPE, DROPPED_TIME FROM USER_RECYCLEBIN;
no rows selected.
```

13.8 Index

Index-related Statements

Statements for creating, dropping, altering an index are as follows.

- Creating an index: Refer to **CREATE INDEX**.
- Dropping an index: Refer to **DROP INDEX**.
- Updating an index: Refer to **ALTER INDEX**.

Information which is related to an index object can be retrieved through the following views.

Table 13-16 Index object related information

Schema	View	Description
DICTIONARY_SCHEMA	**ALL_INDEXES**	Information about user accessible indexes
	ALL_IND_COLUMNS	Column information about user accessible index
	USER_INDEXES	Information about user owned indexes
	USER_IND_COLUMNS	Column information about user owned index

Concepts of Index

Index is a table related object, and it is used to improve data access performance when retrieving the tabl e. Each index consists of key values using the data in one or more columns of the table. It is an object whi ch is separate from a table.
Database automatically builds the key data of the index when creating the index, and the index key data i s automatically managed when adding/deleting/ updating the table data.

The following is an example of a query.

```
SELECT data FROM t1 WHERE id = 12345;
```

If an index does not exist, the results which satisfy the condition are found by checking all rows in the tab le. If the table consists of multiple rows while the number of results to satisfy the condition is relatively sm all, the query above has a very inefficient response time.
When creating an index in id column by using the **CREATE INDEX** statement as follows, the optimizer eva luates the costs between the full scan and index scan of a table, and selects the index scan to improve qu ery performance.

```
CREATE INDEX t1_idx_id ON t1(id);
```

When creating an index, two or more columns can be used as an index key, and the index which consists of two or more keys is called as a composite index. The composite Index is sorted by the first key, or sorte d by the second key if the first key value is same. It is sorted as many as the number of the keys in this wa y.

When creating indexes, the column sort order can be specified in ascending (ASC) or descending(DESC). The sort order of NULL value can be specified as NULLS FIRST or NULLS LAST. Refer to the following exam ple.

```
gSQL> CREATE TABLE t1 ( value INTEGER );
Table created.
gSQL> INSERT INTO t1 VALUES (1), (NULL), (3), (2), (NULL);
5 rows created.
gSQL> CREATE INDEX idx1 ON t1 ( value ASC NULLS LAST );
Index created.
gSQL> CREATE INDEX idx2 ON t1 ( value DESC NULLS FIRST );
Index created.
gSQL> SELECT /*+ INDEX(t1, idx1) */ * FROM t1;
VALUE
-----
    1
    2
    3
 null
 null
5 rows selected.
gSQL> SELECT /*+ INDEX(t1, idx2) */ * FROM t1;
VALUE
-----
 null
 null
    3
    2
    1
5 rows selected.
```

In the example above, the index idx1 is specified in the ascending order (ASC), which is NULLS LAST, and the index idx2 is specified in the descending order (DESC), which is NULLS FIRST.
It uses the different index hints when retrieving the table with the same query, so all rows are retrieved by using each index. Results using the index idx1 is sorted in ascending order, and NULL value is located at t

he end. However, the results using the index idx2 is sorted in descending order and NULL value is located in the first.

Concepts of UNIQUE

Index can be created as UNIQUE index or non-unique index. If the key values is not UNIQUE when creatin g UNIQUE index, then an error occurs.
NULL value is allowed as a key value in UNIQUE index and UNIQUE constraint.
If NULL value is included, the truth table for UNIQUE is as follows. In other words, if the key is one, then it can have multiple null value.

Table 13-17 Truth table for UNIQUE in two values

Value1	Value2	UNIQUE
1	1	false
1	2	true
1	null	true
null	null	true

The UNIQUE index or UNIQUE constraint consisting of two or more keys can have null as the whole value or partial value. If NULL is included in the composite key, the truth table for UNIQUE is as follows.

Table 13-18 Truth table for UNIQUE in composite key

Row1	Row2	UNIQUE
(1, 1)	(1, 1)	false
(1, 1)	(1, 2)	true
(1, null)	(1, null)	false
(1, null)	(2, null)	true
(null, null)	(null, null)	true

Note that the definition of the UNIQUE has been changed in the SQL standard as follows.

- UNIQUE definition until SQL1999
  If there are no two rows in T such that the value of each column in one row is non-null and **is equal to** the value of the corresponding column in the other row according to Subclause 8.2, “<comparison predicate>”, then the result of the <unique predicate> is true; otherwise, the result of the <unique predicate> is false.

- UNIQUE definition after SQL2003

> If there are no two rows in T such that the value of each column in one row is non-null and **is not distinct from** the value of the corresponding column in the other row, then the result of the <unique predicate> is True; otherwise, the result of the <unique predicate> is False.

GOLDILOCKS follows the SQL2011 standards which is the standard after SQL2003, and the SQL standard UNIQUE is defined by whether or not UNIQUE of composite key exists as shown in the following table.

Table 13-19 Truth table for UNIQUE of the SQL standard composite key

Row1	Row2	Until SQL1999	After SQL2003
(1, 1)	(1, 1)	false	false
(1, 1)	(1, 2)	true	true
(1, null)	(1, null)	true	false
(1, null)	(2, null)	true	true
(null, null)	(null, null)	true	true

Each DBMS vendor follows the SQL standard for UNIQUE definition as follows.

- DBMS which follows the UNIQUE definition after SQL2003: Oracle, SQL server
- DBMS which follows the UNIQUE definition until SQL1999: Postgres, MySQL

13.9 View

View-related Statements

Statements for creating, dropping, altering a view are as follows.

- Creating a view: Refer to **CREATE VIEW**.
- Dropping a view: Refer to **DROP VIEW**.
- Altering a view: Refer to **ALTER VIEW**.

Information which is related to a view object can be retrieved through the following views.

Table 13-20 View object related information

Schema	View	Description
DICTIONARY_SCHEMA	**ALL_VIEWS**	Information about user accessible views
	ALL_DEPENDENCIES	Information about objects related to user accessible views
	USER_VIEWS	Information about user owned views
	USER_DEPENDENCIES	Information about objects related to user owned views
INFORMATION_SCHEMA	**VIEWS**	Information about user accessible views
	VIEW_TABLE_USAGE	Information about the table used when creating a view
	VIEW_ROUTINE_USAGE	Information about the stored function used when creating a view

Concepts of View

While a table is a physical relation of storing data, a view is a logical relation consisting of queries. In the SQL standard, it is called as the viewed table. Queries about view can be used as same as the table.

A view has the following advantages.

- Data access can be restricted to allow querying only part of the information in the table.
- The complex and frequent queries can be written in a single view to decrease query's complexity.
- Data can be presented in a perspective different from the table perspective by changing the column names or data of the view.
- Creating applications of views is not affected by the changes in table structures.

The view which is created by a **CREATE VIEW** statement is replaced with in-line view when executing queries as follows.

- Creating a view

```
CREATE VIEW v1 ( v_id, v_sum )
AS
SELECT l_partkey, SUM( l_quantity )
  FROM lineitem
 GROUP BY l_partkey;
```

- Querying the view

```
SELECT v_id, v_sum
  FROM v1
 WHERE v_sum > 1000;
```

- Translating the view

```
SELECT v_id, v_sum
  FROM ( SELECT l_partkey, SUM( l_quantity )
           FROM lineitem
          GROUP BY l_partkey
       ) v1 ( v_id, v_sum )
 WHERE v_sum > 1000;
```

In the following example, the asterisk (*) is used in SELECT statement when creating a view v1, and the a sterisk means all columns.
In this case, as follows, all the columns including the added column can be retrieved by executing query o f view v1 even after a new column addr is added to the table t1 of which the view is approaching.

```
gSQL> CREATE TABLE t1 ( id INTEGER, name VARCHAR(128) );
Table created.
gSQL> INSERT INTO t1 VALUES ( 1, 'leekmo' );
1 row created.
```

- Creating a view by using an asterisk (*)

```
gSQL> CREATE VIEW v1 AS SELECT * FROM t1;
View created.
```

- Querying the view

```
gSQL> SELECT * FROM v1;
ID NAME
```

```
-- ------
 1 leekmo
1 row selected.
```

- Adding a column to the table which is referenced by the view

```
gSQL> ALTER TABLE t1 ADD COLUMN addr VARCHAR(1024) DEFAULT 'N/A';
Table altered.
```

- Querying the view after adding the column

```
gSQL> SELECT * FROM v1;
ID NAME   ADDR
-- ------ ----
 1 leekmo N/A
1 row selected.
```

However, creating a view using the asterisk (*) is not recommended because it can cause changes in the application when changing the table structure.

13.10 Sequence

Sequence-related Statements

Statements for creating, dropping, altering, using a sequence are as follows.

- Creating a sequence: Refer to **CREATE SEQUENCE**.
- Dropping a sequence: Refer to **DROP SEQUENCE**.
- Altering a sequence: Refer to **ALTER SEQUENCE**.
- Using a sequence: Refer to **NEXTVAL**, **CURRVAL**.

Information which is related to a sequence object can be retrieved through the following views.

Table 13-21 Sequence object related information

Schema	Vew	Description
DICTIONARY_SCHEMA	**ALL_SEQUENCES**	Information about user accessible sequences
	USER_SEQUENCES	Information about user owned sequences
INFORMATION_SCHEMA	**SEQUENCES**	Information about user accessible sequences

Concepts of Sequence

Sequence is an object which automatically creates a sequential number, and it is called as sequence gene rator in the SQL standard. Sequence is a useful object to automatically manage the unique key or primary key. A sequence can be spanned multiple tables.

The following is an example of using a single sequence object to automatically generate the id column va lue, and using it across multiple tables.

```
gSQL> CREATE SEQUENCE seq;
Sequence created.
gSQL> INSERT INTO t1 (id, name) VALUES ( seq.NEXTVAL, 'leekmo' );
1 row created.
gSQL> INSERT INTO t2 (id, addr) VALUES ( seq.CURRVAL, 'Seoul, Korea' );
1 row created.
gSQL> SELECT * FROM t1;
ID NAME
-- ------
 1 leekmo
1 row selected.
```

```
gSQL> SELECT * FROM t2;
ID ADDR
-- ------------
 1 Seoul, Korea
1 row selected.
```

In the example above, the next number of the id column in table t1 is automatically created by using the seq.NEXTVAL function. The same value is used for id column in table t2 by using the seq.CURRVAL functi on.
When creating the sequence, the starting value, incremental value, minimum value, maximum value, cycl e or no cycle and cached value of the automatically generated number can be specified.
For more information, refer to **CREATE SEQUENCE** statement.

An identity column is similar to a sequence, and it automatically generates numbers in a table. It can be u sed as follows.

```
gSQL> CREATE TABLE t1 ( id INTEGER GENERATED ALWAYS AS IDENTITY, name VARCHAR(128) );
Table created.
gSQL> INSERT INTO t1 (name) VALUES ( 'leekmo' );
1 row created.
gSQL> INSERT INTO t1 (name) VALUES ( 'mkkim' );
1 row created.
gSQL> INSERT INTO t1 (name) VALUES ( 'xcom73' );
1 row created.
gSQL> SELECT * FROM t1;
ID NAME
-- ------
 1 leekmo
 2 mkkim
 3 xcom73
3 rows selected.
```

In the example above, the id column is created as an identity column when creating table t1. The identity column automatically generated id column values, and the values are inserted when executing INSERT sta tement.
For more information, refer to **<identity column specification>** clause of **CREATE TABLE** statement.

The sequence and the identity column are functionally similar because they create the sequential number s. However, they are different in the following aspects.

- Sequence is an SQL schema object but identity column is a column of the table.
- Sequence can be spanned multiple tables but identity column can be used only in one table.

After creating a sequence, the sequence values can be used by using **NEXTVAL** or **CURRVAL** function. Th e sequence value is created independently from the transaction, and it is not affected by COMMIT or ROL LBACK of the transaction.

```
gSQL> INSERT INTO t1(id) VALUES( seq.NEXTVAL );
1 row created.
gSQL> SELECT id FROM t1;
ID
--
 1
1 row selected.
gSQL> ROLLBACK;
Rollback complete.
gSQL> INSERT INTO t1(id) VALUES( seq.NEXTVAL );
1 row created.
gSQL> SELECT id FROM t1;
ID
--
 2
1 row selected.
```

In the example above, seq.NEXTVAL function generates a value of 1 in the first INSERT statement.
Then, the seq.NEXTVAL value 2 is generated after the transaction ROLLBACK by increasing the value sinc e the next value, independently from the transaction.

The sequence value can be used only in the following statements.

- The select list value of the top-level SELECT statement
  - SELECT seq.NEXTVAL FROM dual;
- The select list value of INSERT .. SELECT statement
  - INSERT INTO t1(id) SELECT seq.NEXTVAL FROM daul;
- Input value of INSERT .. VALUES statement
  - INSERT INTO t1(id) VALUES (seq.NEXTVAL);
- SET value of UPDATE statement
  - UPDATE t1 SET id = seq.NEXTVAL;

The sequence value can be used only in the location as specified above. It can not be used in subquery, a ggregation function argument, or clauses such as WHERE, DISTINCT, GROUP BY, HAVING, ORDER BY.

Cluster Sequence

When using GOLDILOCKS by configuring the cluster system, the global sequence object is internally used. The global sequence object sets the pool of sequence values to be commonly used over all cluster system, and allocates it as much as the cache size when each member node calls NEXTVAL. In other words, if valu es of 20 sequences are alloceted to a specific node, then the value allocated to other nodes starts from th e next value. A member node loads sequence values allocated by the global sequence object in its local c ache, then returns them as a result of NEXTVAL call until all of them are run out.

The global sequence object has the following features and constraints comparing to the sequence for the standalone database.

- The sign can not be modified by using the INCREMENT BY option in the ALTER SEQUENCE statement. (The size can be modified.)
- When using the CYCLE option, a duplicated value can be returned to member nodes due to the size of the entire sequence pool. Therefore, if the CYCLE option is required, then the sequence pool shoul d be set big enough considering INCREMENT BY, CACHE SIZE, and the numner of cluster member no des.
- Even when there is not an error, the sequence values returned by a specific member node may not be sequential. However, a sequential sequence value is obtained when only a single member node calls NEXTVAL.
- If it is NOCACHE, then the CACHE SIZE is 1, so the additional sequence values are not loaded on a loc al cache. In this case, the global sequence object allocates a single sequence whenever calling NEXTV AL, then it increases the networking cost and may cause the poor performance.
- It is generally operated as AUTO COMMIT when creating, altering and deleting the sequence.
- When modifying the size of CACHE and INCREMENT by using the ALTER statement, then all sequenc e values loaded on local caches of all nodes are reset. In other words, the new sequence set should b e allocated again by the global sequence object when calling NEXTVAL later.

13.11 Synonym

Synonym-related Statements

Statements for creating and dropping a synonym are as follows.

- Creating a synonym: Refer to **CREATE SYNONYM**.
- Dropping a synonym: Refer to **DROP SYNONYM**.

Information which is related to a synonym object can be retrieved through the following views.

Table 13-22 Synonym object related information

Schema	View	Description
DICTIONARY_SCHEMA	**ALL_SYNONYMS**	All synonym information
	USER_SYNONYMS	Information about user owned synonym

Concepts of Synonym

Synonym is an alias for the following objects.

- Table
- View
- Sequence
- Stored procedure
- Stored function
- Other synonyms

Synonyms can be used as an alias in SELECT, INSERT, UPDATE, DELETE, LOCK TABLE, GRANT, REVOKE, C OMMENT statements.

Using synonym is very convenient. It is because only the synonym should be redefined without modifying the application even when the schema of underlying objects is changed.
The database security can be improved by hiding the object's real name and its owner. Moreover, the dat abase usability is enhanced by changing the long object name to a short name.

Synonyms are classified as private synonym and public synonym. Private synonym is a schema object and public synonym is a non-schema object.

The following examples of creating and using the private synonym and the public synonym indicated by t

he table below describe the concepts of them.

```
gSQL> \CONNECT u1 u1
gSQL> CREATE TABLE u1.t1 (col1 INTEGER );
gSQL> INSERT INTO u1.t1 VALUES(1);
gSQL> COMMIT;
```

Private Synonym

Private synonym is a schema object. If a synonym is created without the schema name, the default schem a name of the user performing the statement is used.

```
gSQL> \CONNECT u2 u2
gSQL> CREATE SYNONYM u2.syn1 FOR u1.t1;
Synonym created.
gSQL> SELECT * FROM u2.syn1;
ERR-42000(16254): lacks privilege (SELECT ON TABLE "U1"."T1")
```

Synonym is only an alias. Therefore, if a user does not have the appropriate privileges on the underlying o bject u1.t1, then the user can not use it even when the user created the synonym.

```
gSQL> \CONNECT u1 u1
gSQL> GRANT SELECT ON TABLE u2.syn1 TO u2;
gSQL> \CONNECT u2 u2
gSQL> SELECT * FROM u2.syn1;
COL1
----
   1
1 row selected.
gSQL> SELECT * FROM u1.t1;
COL1
----
   1
1 row selected.
gSQL> DROP SYNONYM u2.syn1;
Synonym dropped.
```

In the example above, the SELECT privilege of u2.syn1 is granted to u2. This is as same as the SELECT priv ilege of u1.t1 is granted to u2. Therefore, be cautious when granting privileges to synonyms.

Public Synonym

Public Synonym is a non schema object. It is not allowed to specify the schema name when it is created o r droppped.

```
gSQL> \CONNECT u2 u2
gSQL> CREATE PUBLIC SYNONYM pubSyn1 FOR u1.t1;
Synonym created.
gSQL> SELECT * FROM pubSyn1;
ERR-42000(16254): lacks privilege (SELECT ON TABLE "U1"."T1")
```

Public synonym dose not have an owner, and it is accessible for all users. However, a user without an app ropriate privilege on the underlying objects can not access the underlying objects.

```
gSQL> \CONNECT u1 u1
gSQL> GRANT SELECT ON TABLE pubSyn1 TO u2;
gSQL> \CONNECT u2 u2
gSQL> SELECT * FROM pubSyn1;
COL1
----
   1
1 row selected.
gSQL> SELECT * FROM u1.t1;
COL1
----
   1
1 row selected.
gSQL> DROP SYNONYM pubSyn1;
Synonym dropped.
```

13.12 Stored Procedure

Stored Procedure-related Statements

Statements for creating, dropping and altering a stored procedure are as follows.

- Creating a stored procedure: Refer to **CREATE PROCEDURE**.
- Dropping a stored procedure: Refer to **DROP PROCEDURE**.
- Altering a stored procedure: Refer to **ALTER PROCEDURE**.

Information which is related to a stored procedure can be retrieved through the following views.

Table 13-23 Stored procedure object related information

Schema	View	Description
DICTIONARY_SCHEMA	ALL_ARGUMENTS	Argument information of user accessible procedure and function
	ALL_DEPENDENCIES	Information of an object related to user accessible procedure and function
	ALL_PROCEDURES	Object information of user accessible procedure and function
	ALL_SOURCE	Source text information of user accessible procedure and function
	USER_ARGUMENTS	Argument information of user owned procedure and function
	USER_DEPENDENCIES	Information of an object related to user owned procedure and function
	USER_PROCEDURES	Object information of user owned procedure and function
	USER_SOURCE	Source text information of user accessible procedure and function
INFORMATION_SCHEMA	PARAMETERS	Argument information of user accessible procedure and function
	ROUTINES	Object information of user accessible procedure and function
	ROUTINE_ROUTINE_USAGE	Information of procedure and function which is referenced by user accessible procedure and function
	ROUTINE_SEQUENCE_USAGE	Information of sequence which is referenced by user accessible procedure and function
	ROUTINE_TABLE_USAGE	Information of table and view which is referenced by user accessible procedure and function

Concepts of Stored Procedure

A stored procedure is a kind of a persistent stored module in procedure form and it is defined and managed in schema unit as like other schema-level database objects. The return value is not defined because it is in procedure form. It is used by directly calling it in CALL statement, another stored procedure, or stored

function.

For more information about a stored procedure, refer to **Schema-level Procedure**.

A store procedure is used as follows.

```
CREATE OR REPLACE PROCEDURE PROC1( A1 INTEGER, A2 INTEGER )
IS
  V1 INTEGER;
BEGIN
  SELECT COUNT(*)
    INTO V1
    FROM T1
    WHERE T1.I1 >= A1 AND T1.I1 <= A2;
  DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
END;
/
BEGIN
  PROC1( 2, 4 ); -- call schema-level procedure
END;
/
V1 = 3
Anonymous PL block executed.
```

13.13 Stored Function

Stored Function-related Statements

Statements for creating, dropping and altering a stored function are as follows.

- Creating a stored function: Refer to **CREATE FUNCTION**.
- Dropping a stored function: Refer to **DROP FUNCTION** .
- Altering a stored function: Refer to **ALTER FUNCTION**.

Information which is related to a stored function can be retrieved through the following views.

Table 13-24 Stored function object related information

Schema	View	Description
DICTIONARY_SCHEMA	ALL_ARGUMENTS	Argument information of user accessible procedure and function
	ALL_DEPENDENCIES	Information of an object related to user accessible procedure and function
	ALL_PROCEDURES	Object information of user accessible procedure and function
	ALL_SOURCE	Source text information of user accessible procedure and function
	USER_ARGUMENTS	Argument information of user owned procedure and function
	USER_DEPENDENCIES	Information of an object related to user owned procedure and function
	USER_PROCEDURES	Object information of user owned procedure and function
	USER_SOURCE	Source text information of user accessible procedure and function
INFORMATION_SCHEMA	PARAMETERS	Argument information of user accessible procedure and function
	ROUTINES	Object information of user accessible procedure and function
	ROUTINE_ROUTINE_USAGE	Information of procedure and function which is referenced by user accessible procedure and function
	ROUTINE_SEQUENCE_USAGE	Information of sequence which is referenced by user accessible procedure and function
	ROUTINE_TABLE_USAGE	Information of table and view which is referenced by user accessible procedure and function

Concepts of Stored Function

A stored function is a kind of a persistent stored module in function form and it is defined and managed in schema unit as like other schema-level database objects. The stored function is classified as follows according to the definition of RETURN clause.

- The function which defined the datatype of the return value returned from the function like as RETU RN <datatype>
- The function which defined the table type of the returning result set like as RETURN TABLE (<column _list>)
  - This is a table function.

For more information about a stored function, refer to **schema-level function**.

RETURN <datatype> Function

It defines the datatype of an expression returned by performing the function in RETURN clause. The funct ion should define the result value to return. Such functions can be directly executed through CALL statem ent. Or, it is used as an expression within the stored procedure or the stored function, or used as an expr ession in SQL statement.

```
gSQL> CREATE OR REPLACE FUNCTION FUNC1( A1 INTEGER, A2 INTEGER )
RETURN INTEGER
IS
  V1 INTEGER;
BEGIN
  SELECT COUNT(*)
    INTO V1
    FROM T1
    WHERE T1.I1 >= A1 AND T1.I1 <= A2;
  RETURN V1;
END;
/
Function created.
gSQL> SELECT FUNC1( 2, 4 ) FROM DUAL;
FUNC1( 2, 4 )
-------------
            3
1 row selected.
```

Table Function

RETURN clause defines the table column list of the result set returned by executing the function, and it is called as a table function. The table function defines the result set to return by using the select statement or the cursor variable. The table function can be used as a table function derived table in FROM clause of SELECT statement.

The table function returns the defined select statement or the result set of performing the cursor variable

's cursor query to the superordinate SELECT statement. The returned result set configures the table functi on derived table in SELECT statement. Also, it can refer to columns of the table listed before the table fun ction derived table in FROM clause as a table function's argument.

```
gSQL> CREATE TABLE t_score( c_grade INTEGER, c_score INTEGER );
Table created.
gSQL> INSERT INTO t_score VALUES ( 1 , 98 ) , ( 1 , 97 ) , ( 1 , 99 ),
                                 ( 2 , 95 ) , ( 2 , 98 ) , ( 2 , 92 ),
                                 ( 3 , 98 ) , ( 3 , 96 ) , ( 3 , 94 );
9 rows created.
gSQL> COMMIT;
Commit complete.
```

- Returning the execution result of the cursor variable

```
gSQL>
CREATE OR REPLACE FUNCTION tf_cv( p_grade INTEGER )
  RETURN TABLE( rf_grade INTEGER, rf_score INTEGER ) AS
  cv SYS_REFCURSOR;
BEGIN
  OPEN cv FOR SELECT * FROM t_score WHERE c_grade = p_grade;

  RETURN TABLE( cv );
END;
/
Function created.
gSQL> SELECT rf_grade, rf_score FROM TABLE( tf_cv( 2 ) );
RF_GRADE RF_SCORE
-------- --------
       2       95
       2       98
       2       92
3 rows selected.
```

- Returning the execution result of the SELECT statement

```
gSQL>
CREATE OR REPLACE FUNCTION tf_select( p_grade INTEGER )
  RETURN TABLE( rf_grade INTEGER, rf_score INTEGER ) AS
BEGIN
  RETURN TABLE ( SELECT * FROM t_score WHERE c_grade = p_grade );
END;
```

```
/
Function created.
gSQL> SELECT rf_grade, rf_score FROM TABLE( tf_select( 2 ) );
RF_GRADE RF_SCORE
-------- --------
       2       95
       2       98
       2       92
3 rows selected.
```

13.14 Package

Package-related Statement

Statements for creating, dropping and altering a package are as follows.

- Creating package: Refer to **CREATE PACKAGE**
- Creating package body: Refer to **CREATE PACKAGE BODY**
- Altering package: Refer to **ALTER PACKAGE**
- Dropping package: Refer to **DROP PACKAGE**

The information related to the package object can be retrieved through the following views.

Table 13-25 Stored function object-related information.

Schema	View	Description
DICTIONARY_SCHEMA	ALL_OBJECTS	It is the information about the object which is accessible by a user.
	ALL_PACKAGE_PRIVS	It is the information about the privilege related to the user package.
	ALL_PACKAGE_PRIVS_MADE	It is the information about the privilege which a user granted to allow access to the package.
	ALL_PACKAGE_PRIVS_RECD	It is the information about the privilege which was granted to a user allow access to the package.
	ALL_SOURCE	It is the information about the source text of procedure, function, package which are accessible by a user.
	USER_OBJECTS	It is the information about the user-owned object.
	USER_PACKAGE_PRIVS	It is the information about the privilege related to the user-owned package.
	USER_PACKAGE_PRIVS_MADE	It is the information about the privilege which granted to allow access to user-owned package.
	USER_PACKAGE_PRIVS_RECD	It is the information about the privilege which was granted to allow access to user-owned package.
	USER_SOURCE	It is the information about the source text of procedure, function, package which are owned by a user.
	MODULES	It is the information about SQL-server module (package) accessible by a user.
	MODULE_BODY	It is the information about package body accessible by a user.
	MODULE_BODY_MODULE_USAGE	It is the information about another package which is being used by the package body accessible by a user.
	MODULE_BODY_ROUTINE_USAGE	It is the information about the procedure or the function which is being used by the package body accessible by a user.

Schema	View	Description
INFORMATION_SCHEMA	MODULE_BODY_SEQUENCE_USAGE	It is the information about the sequence which is being used by the package body accessible by a user.
	MODULE_BODY_TABLE_USAGE	It is the information about the table which is being used by the package body accessible by a user.
	MODULE_MODULE_USAGE	It is the information about another package which is being used by the package accessible by a user.
	MODULE_PRIVILEGES	It is the information about privilege related the package accessible by a user.
	MODULE_ROUTINE_USAGE	It is the information about the procedure or the function which is being used by the package accessible by a user.
	MODULE_SEQUENCE_USAGE	It is the information about the sequence which is being used by the package accessible by a user.
	MODULE_TABLE_USAGE	It is the information about the table which is being used by the package accessible by a user.
	ROUTINE_MODULE_USAGE	It is the information about the package being used by the procedure or by the function, which is accessible by a user.
	VIEW_MODULE_USAGE	It is the information about the package which is being used by the view accessible by a user.

Concepts of Package

A package is a schema object which binds PSM type, a variable, a subprogram, a cursor, and an exception which are logically related. The package is stored in the database through compiling so that another program (another package, a procedure, an external program) to refer, share and execute the package items.

For more information about the package, refer to **PSM Packages**.

The following is an example of creating the package.

```
CREATE TABLE emp( empno NUMBER, sal NUMBER, comm NUMBER );
Table created.
INSERT INTO emp VALUES( 3548, 6000, 1000 );
1 row created.
INSERT INTO emp VALUES( 9369, 5000, NULL );
1 row created.
INSERT INTO emp VALUES( 7294, 4000, 500 );
1 row created.
COMMIT;
Commit complete.
```

```
CREATE OR REPLACE PACKAGE emp_mgmt
IS
  PROCEDURE adjust_sal(v_flag VARCHAR, v_empno NUMBER, v_pct NUMBER);
  FUNCTION get_annual_sal(v_empno NUMBER) RETURN NUMBER;
END;
/
Package created.
CREATE OR REPLACE PACKAGE BODY emp_mgmt
IS
  PROCEDURE adjust_sal(v_flag VARCHAR, v_empno NUMBER, v_pct NUMBER) IS
  BEGIN
    IF v_flag = 'INCREASE' THEN
      UPDATE emp SET sal = sal + (sal * (v_pct / 100)) WHERE empno = v_empno;
    ELSE
      UPDATE emp SET sal = sal - (sal * (v_pct / 100)) WHERE empno = v_empno;
    END IF;
  END;
  FUNCTION get_annual_sal (v_empno NUMBER) RETURN NUMBER
  IS
    v_sal NUMBER;
  BEGIN
    SELECT (sal + NVL(comm,0)) * 12 INTO v_sal FROM emp WHERE empno = v_empno;
    RETURN v_sal;
  END;
END;
/
Package created.
```

The following is an example of using the package.

```
call emp_mgmt.adjust_sal('INCREASE',7369, 10);
Procedure Call complete.
SELECT emp_mgmt.get_annual_sal(7294) FROM DUAL;
EMP_MGMT.GET_ANNUAL_SAL(7294)
-----------------------------
                        54000
1 row selected.
```

14.

Cluster Objects

14.1 Cluster System

Cluster System Related Statements

For more information, refer to the followings.

- Expanding a cluster system
  - **CREATE CLUSTER GROUP**
  - **ALTER CLUSTER GROUP name ADD MEMBER**

- Controlling an inactive cluster member
  - **ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS**
  - **ALTER SYSTEM JOIN DATABASE**

- Rebalancing data
  - **ALTER DATABASE REBALANCE**
  - **ALTER TABLE name REBALANCE**

Information which is related to a cluster system can be retrieved through the following views.

Table 14-1 Cluster system related information

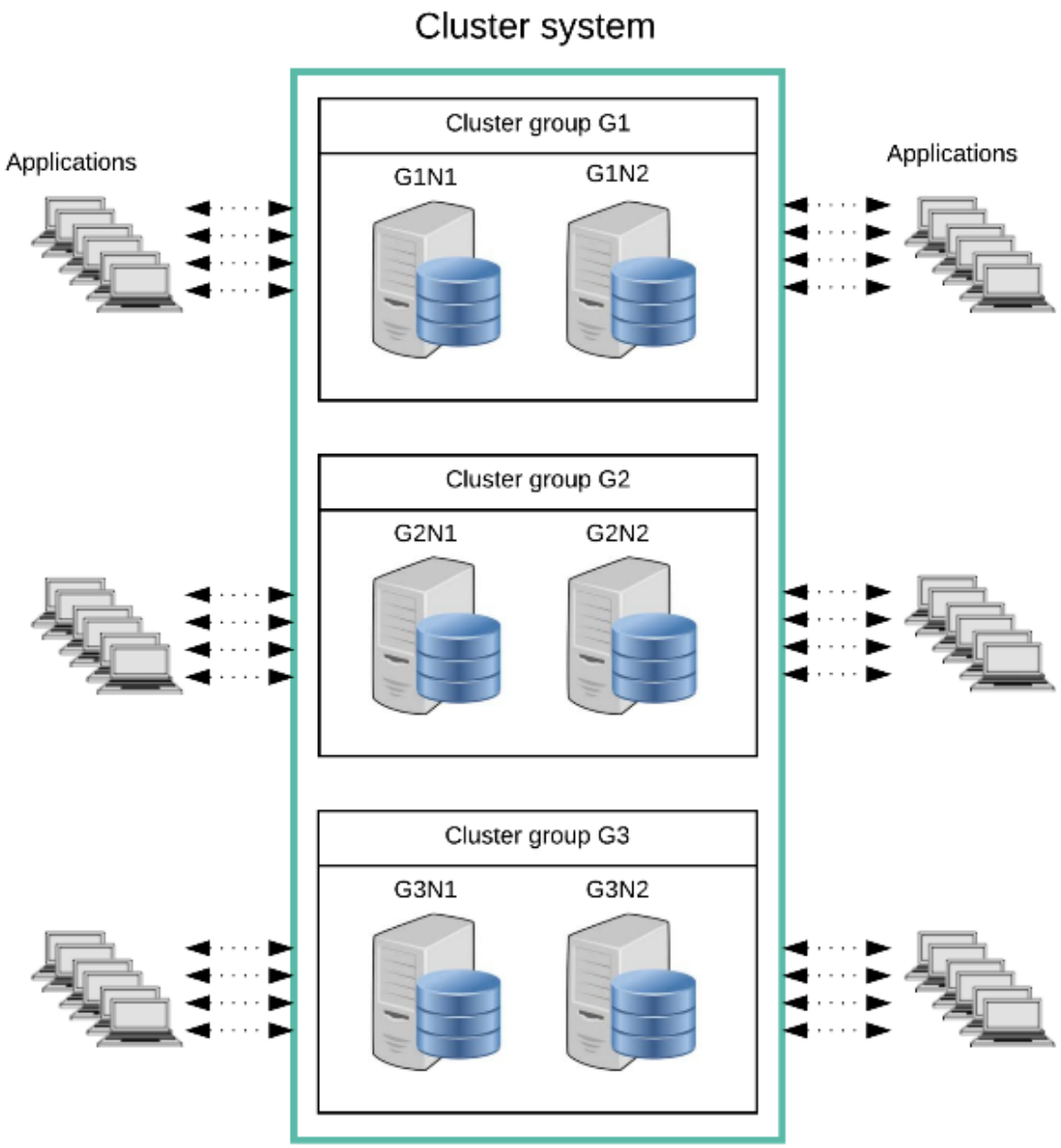
Schema	View	Description
DICTIONARY_SCHEMA	**DBA_CLUSTER**	Object information of a cluster group and a cluster member which configure a cluster
	DBA_CLUSTER_COMMENTS	Comment information of a cluster group and a cluster member
PERFORMANCE_VIEW_SCHEMA	**V$CLUSTER_MEMBER**	Status information of a cluster member

Concepts of Cluster System

GOLDILOCKS cluster system manages data of a single database by sharding or duplicating the data into several servers. Applications can be run on every server configuring a cluster system, and run in the same way as using a single database system regardless of a system configuration or a connected server.

GOLDILOCKS cluster system consists of one or more cluster groups, and a cluster group consists of one or more cluster members. It does not require a separate application server or a meta server, but applications are connected to a cluster member corresponding to a data server, and run.

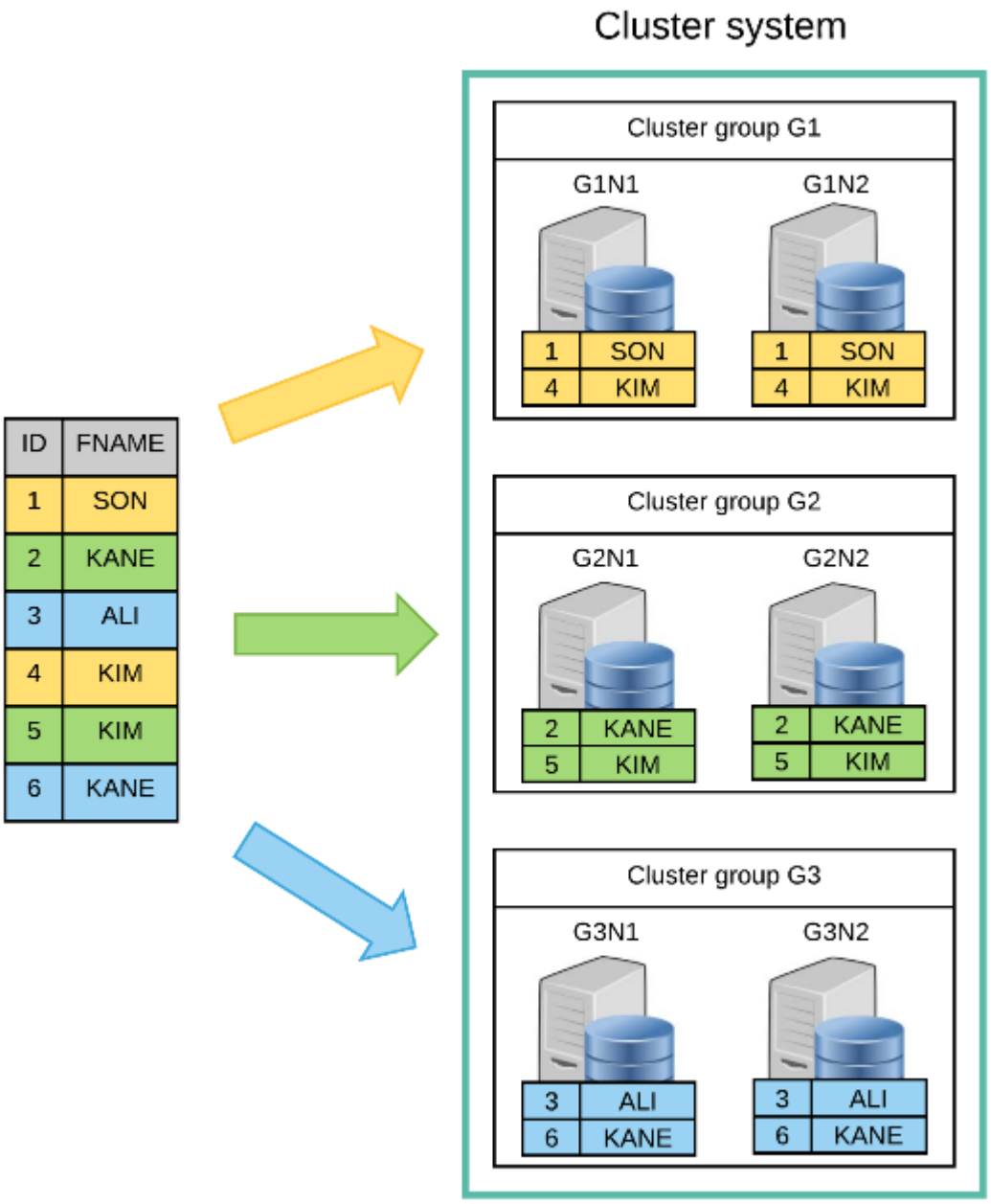
Figure 1 3 x 2 cluster system



The figure above is a 3x2 cluster system which consists of two cluster members consisting of three cluster groups and a single cluster group. In the figure above, the cluster system consists of cluster groups (G1, G2, G3), and the cluster group G1 consists of cluster members (G1N1, G1N2), the cluster group G2 cons ists of cluster members (G2N1 and G2N2), and the cluster group G3 consists of cluster members (G3N1, G3N2). Applications can access any of those six cluster members and it is run as same as using a single da tabase.

The table data is sharded and placed in each cluster group, and cluster members in a cluster group maint ain the replications same. The figure below describes the concepts of the table data placement in a 3x2 cl uster.

Figure 2 Concepts of sharding and duplicating by the cluster



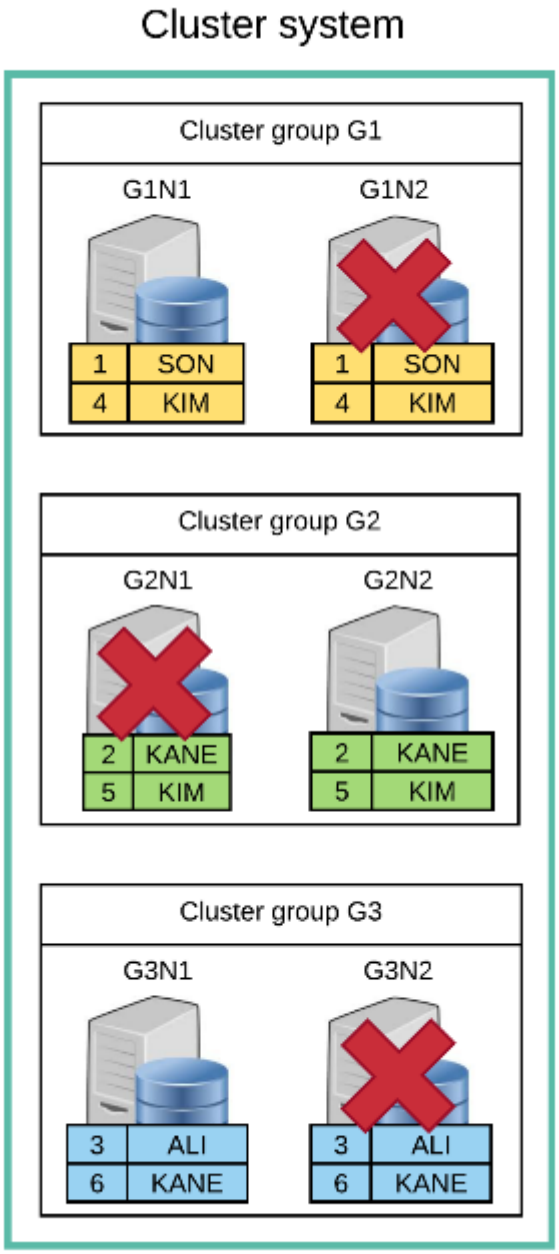
The table data is sharded and placed in each cluster group according to the sharding strategy defined by a user. (According to the ID column in the figure above) The placed data in a cluster group maintains the replication of a cluster member in a cluster group.

Availability of Cluster System

Cluster continues to provide service even when a specific server is broken or the network is cut. Cluster m embers configuring each cluster group maintain the same data replications, so the service does not stop even when a single cluster member is broken. In other words, unless the data is lost due to the malfuncti on of all cluster members in a cluster group, the service continues to be provided.

The cluster continues to provide service even when three devices are broken in the 3x2 cluster as follows.

Figure 3 Cluster availability



If an additional errors occur in G1N1, G2N2, G3N1 in the situation above, then the data loss occurs and t he service can not be provided any more. Therefore, a user should make the broken device to participate i n a cluster system, or add a new cluster member before an additional error occurs.

- **ALTER SYSTEM JOIN DATABASE** statement is used to make the broken cluster member to participate in a cluster system again.
- **ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS** statement is used to drop the broken cluster member from the cluster system.
- **ALTER CLUSTER GROUP name ADD MEMBER** statement is used to add a cluster member to a cluster group for the high availability.
- **ALTER DATABASE REBALANCE** statement and **ALTER TABLE name REBALANCE** statement are used t o rebalance data to a newly added cluster member.

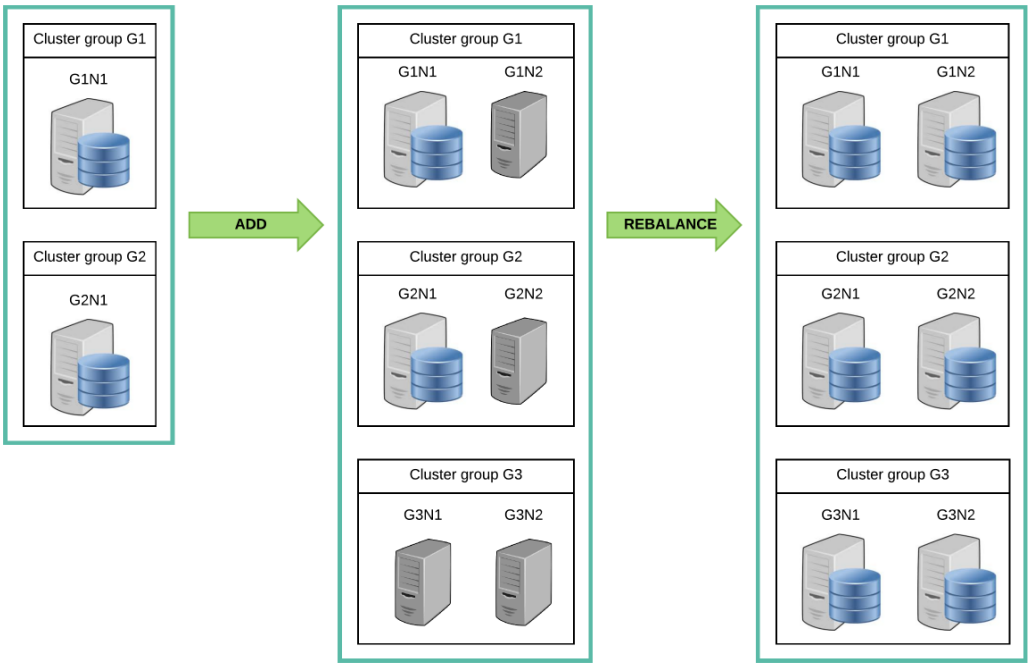
Expanding Cluster System

The cluster can be expanded by adding a new server without stopping the service.

Cluster is expanded by adding a cluster member or a cluster group and by rebalancing the data to an crea ted server.

The following is an example of expanding a 2x1 cluster to a 3x2 cluster.

Figure 4 Expanding cluster system



Add a cluster group and a cluster member by using the following statements to expand a cluster.

- ALTER CLUSTER GROUP name ADD MEMBER
- CREATE CLUSTER GROUP

To add a new cluster member to the cluster system, tablespaces in the cluster member and those in the cluster system should be same. In other words, tablespaces as same as all tablespaces in the cluster system should be created in the cluster member.

The following is an example of adding a cluster group and a cluster member of the 3x2 cluster to the 2x1 cluster. Add the member G1N2 to the group G1, and add the member G2N2 to the group G2. Then, create the group G3 including the members (G3N1, G3N2).

- Add the member G1N2 to the group G1.

```
gSQL>
ALTER CLUSTER GROUP G1
      ADD CLUSTER MEMBER G1N2 HOST '192.168.0.12' PORT 10120;
Cluster Group altered.
```

- Add the member G2N2 to the group G2.

```
gSQL>
ALTER CLUSTER GROUP G2
      ADD CLUSTER MEMBER G2N2 HOST '192.168.0.22' PORT 10220;
Cluster Group altered.
```

- Create the group G3.

```
gSQL>
CREATE CLUSTER GROUP G3
       CLUSTER MEMBER G3N1 HOST '192.168.0.31' PORT 10310,
       CLUSTER MEMBER G3N2 HOST '192.168.0.32' PORT 10320;
Cluster Group created.
```

The cluster group and the cluster member which are newly added to the cluster system can provide the service by synchronizing the dictionary information of the SQL object. However, the data is not yet placed in the added cluster member, so the availability can not be increased nor can the load balancing be expected. Therefore, the data should be rebalanced to the created cluster member for the high availability and the load balancing.

Use the following statements sequentially to rebalance the data.

- ALTER DATABASE REBALANCE

- **ALTER TABLE name REBALANCE**

The following is an example of rebalancing data to all tables in the database.

```
gSQL> ALTER DATABASE REBALANCE;
Database altered.
```

14.2 Cluster Group

Cluster Group Related Statements

For more information about creating, dropping, and altering a cluster group, refer to the followings.

- Creating a cluster group: **CREATE CLUSTER GROUP**
- Dropping a cluster group: **DROP CLUSTER GROUP**
- Altering a cluster group: **ALTER CLUSTER GROUP name ADD MEMBER**

Information which is related to a cluster group can be retrieved through the following views.

Table 14-2 Cluster group related information

Schema	View	Description
DICTIONARY_SCHEMA	DBA_CLUSTER	Object information of a cluster group and a cluster member which configure the cluster
	DBA_CLUSTER_COMMENTS	Comment information of a cluster group and a cluster member

Concepts of Cluster Group

At least one cluster group should be created to run the cluster system.

Creating Cluster Group

The first created cluster group should include itself as a cluster member. For more information about creating a cluster group, refer to **CREATE CLUSTER GROUP**.

A cluster member is a physical concept meaning the data server, but a cluster group is a logical concept consisting of one or more cluster members.

The availability and load balancing of the cluser system depends on the configration of the cluster group. The more cluster member the cluster group includes the higher the availability. The more the number of the cluster groups the bigger the throughput of the cluster system because the data is distributed.

All cluster members in a cluster group maintain the same data replications, so it can continuously provides the service unless an error occurs in all cluster members configuring a cluster group. It is recommended to configure the cluster group with two or more cluster members to maintain the availability of the cluste

r system.

The table data is sharded according to the sharding strategy, and it is stored and managed in each differe nt cluster group according to the shard placement strategy. An appropriate table sharding strategy and t he placement strategy according to the service feature determines the entire system performance. Each tr ansaction and query is processed focusing on a cluster group in which the data is stored, so the performa nce is improved if the referenced data exist in the same cluster group.

Dropping Cluster Group

The cluster group participating in a cluster system and providing a service can be dropped for various reas ons. For more information about dropping the cluster group, refer to **DROP CLUSTER GROUP**.

To drop the cluster group, all shards in a sharded table which is created in that cluster group should be tr ansferred to another cluster group. It should be transferred separately according to cluster-wide, group-s pecific table.

14.3 Cluster Member

Cluster Member Related Statements

For more information, refer to the followings.

- Adding a cluster member: **ALTER CLUSTER GROUP name ADD MEMBER**
- Dropping a cluster member: **ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS**
- Controlling a cluster member
  - **ALTER SYSTEM JOIN DATABASE**
  - **ALTER CLUSTER GROUP name OFFLINE MEMBER**
  - **ALTER DATABASE RESET LOCAL CLUSTER MEMBER**

Information which is related to a cluster member can be retrieved through the following views.

Table 14-3 Cluster member related information

Schema	View	Description
DICTIONARY_SCHEMA	**DBA_CLUSTER**	Object information of a cluster group and a cluster member which configure a cluster
	DBA_CLUSTER_COMMENTS	Comment information of a cluster group and a cluster member
PERFORMANCE_VIEW_SCHEMA	**V$CLUSTER_MEMBER**	Status information of a cluster member

Concepts of Cluster Member

The cluster member is a server configuring a cluster system, and it maintains the replications as same as those of cluster member in a cluster group.

The cluster member is a data server storing a part of the cluster database data, and it is also an application server processing the connection and request of an application. Moreover, it is a meta server duplicating and managing the meta information. In other words, GOLDILOCKS cluster does not require any data server, application server, or meta server.

Cluster members which belong to the same cluster group have the same data replications. Therefore, an error in a specific cluster member does not cause an error in the entire system. It is recommended to include two or more cluster members in each cluster group for the high availability of the system. A single cluster group can be configured with maximum 32 cluster members.

Perform **ALTER CLUSTER GROUP name ADD MEMBER** statement to add a new cluster member to a clust er group.

The added cluster member maintains the meta information of the SQL object as same as that in the clust er system, so it can process the connection and request of the application. However, the data is not place d, so adding a cluster member does not guarantee the high availability of the cluster group.

Perform the following statements to place the data after adding a cluster member.

- The placement of the entire data: Refer to **ALTER DATABASE REBALANCE**.
- The placement of only a part of the table: **ALTER TABLE name REBALANCE**.

When an error occurs on a cluster member, service is continously provided but DDL can not be performed. For a nomal service operation, an appropriate action should be taken for the cluster member with an erro r.

To make the cluster member with an error to participate in the cluster system again, then perform **ALTER SYSTEM JOIN DATABASE** statement after connecting to the cluster member and driving up to the LOCAL OPEN phase.

Even when a part of cluster members are not driven or the cluster system is driven while the network is di sconnected, perform **ALTER SYSTEM JOIN DATABASE** statement after driving up the cluster member to t he LOCAL OPEN phase.

If the cluster member device with an error can not be restored, then drop that cluster member by perfor ming **ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS** statement in the cluster system.

The cluster member dropped from the cluster system still includes the previous information, and it can no t participate in the cluster system again. Perform **ALTER DATABASE RESET LOCAL CLUSTER MEMBER** stat ement to reset the cluster member to when it is before participating in the cluster system.

The statement above is different from newly creating the database of the cluster member because it main tains the tablespace information. Therefore, it can shorten the time to create the tablespace when adding a new cluster member to the cluster system.

14.4 Cluster Location

Cluster Location Related Statements

For more information about creating, dropping, and altering a cluster location, refer to the followings.

- Creating a cluster location: **CREATE CLUSTER LOCATION**
- Dropping a cluster location: **DROP CLUSTER LOCATION**
- Altering a cluster location: **ALTER CLUSTER LOCATION**

Information which is related to a cluster location can be retrieved through the following views.

Table 14-4 Cluster location related information

Schema	View	Description
PERFORMANCE_VIEW_SCHEMA	**V$CLUSTER_LOCATION**	Information of cluster location

Concepts of Cluster Location

Cluster location is a connection information to connect the internal cluster networks of each cluster mem ber registered on the cluster system. Each cluster members use the cluster-exclusive tcp network to transf er and receive various protocols such as the transaction processing and the exchanging the management information. In this case, the member name, host ip address, and port are used for the connection and th ey are called as a cluster location by the lump.

A unique cluster location information should be specified for each member, and if the information is dupl icate, then the cluster network connection fails and the cluster system does not operate normally.

Cluster location information is automatically added or deleted when adding or dropping a cluster membe r, so a user rarely need to directly and solely add or delete the location information. However, DDL statem ent related to the cluster location can be used in the following case.

- When the cluster location information is lost due to the deleted location control file: **CREATE CLUSTE R LOCATION**
- When the previously registered connection information of the cluster member is altered, which mean s that the hardware or the connected ip and port is altered: **ALTER CLUSTER LOCATION**

14.5 Cluster Table and Shard

Shard Related Statements

For more information about definition and rebalance of the shard, refer to the followings.

- Definition of a shard: <table sharding strategy> clause of **CREATE TABLE**.
- Rebalancing a shard: **ALTER TABLE name REBALANCE**

Information which is related to a shard in a cluster table can be retrieved through the following views.

Table 14-5 Shard in a cluster table related information

Schema	View	Description
DICTIONARY_SCHEMA	ALL_CLUSTER_TABLES	Information of user accessible cluster table
	ALL_SHARD_KEY_COLUMNS	Shard key column information of user accessible cluster table
	ALL_TAB_PLACE	Placement information of user accessible cluster table
	ALL_TAB_SHARDS	Shard information of user accessible cluster table
	USER_CLUSTER_TABLES	Information of user owned cluster table
	USER_SHARD_KEY_COLUMNS	Shard key column information of user owned cluster table
	USER_TAB_PLACE	Placement information of user owned cluster table
	USER_TAB_SHARDS	Shard information of user owned cluster table

Cluster Table Type

A table which is created by a user in a cluster environment is one of the followings.

- Cloned table: Equally duplicating the table data and managing it
- Sharded table: Horizontally sharding the table data and managing it

A cloned table is appropriate for a table such as a product list or a provider list whose data is relatively small and is not often altered, because a cloned table duplicates all table data and manages it. When inserting, deleting, updating data to the cloned table, they are applied same to all cluster members to which the cloned table is placed.

A sharded table is appropriate for a table such as a transaction history or a call history whose data is big so required to be sharded. They are classifed according to three sharding strategies as follows.

- Hash sharded table
  - It divides the table data into several shards based on the hash value of a sharding key, then place s them in the cluster system.
- Range sharded table
  - It divides the table data into several shards based on the range value of a sharding key, then plac es them in the cluster system.
- List sharded table
  - It divides the table data into several shards based on the list value of a sharding key, then places t hem in the cluster system.

A sharded table horizontally divides rows and manages them in shard unit, the sharding strategy is define d by using the SHARDING BY clause in **CREATE TABLE** statement. The row set classified by the sharding s trategy is called as shard.

Each shard is placed in a cluster group according to the placement strategy defined by a user. A shard ca n automatically be placed by using AT CLUSTER WIDE of **CREATE TABLE**, or a cluster group to place a sha rd can be specified by using AT CLUSTER GROUP clause. When a shard is automatically placed by using A T CLUSTER WIDE, **ALTER TABLE name REBALANCE** is performed after creating the cluster group by using **CREATE CLUSTER GROUP**. However, when specifying a cluster group to allocate a shard, the shard is not placed in the newly created cluster group.

The following is an example of creating a table according to the cluster table type and placing the data in the 3x2 cluster environment.

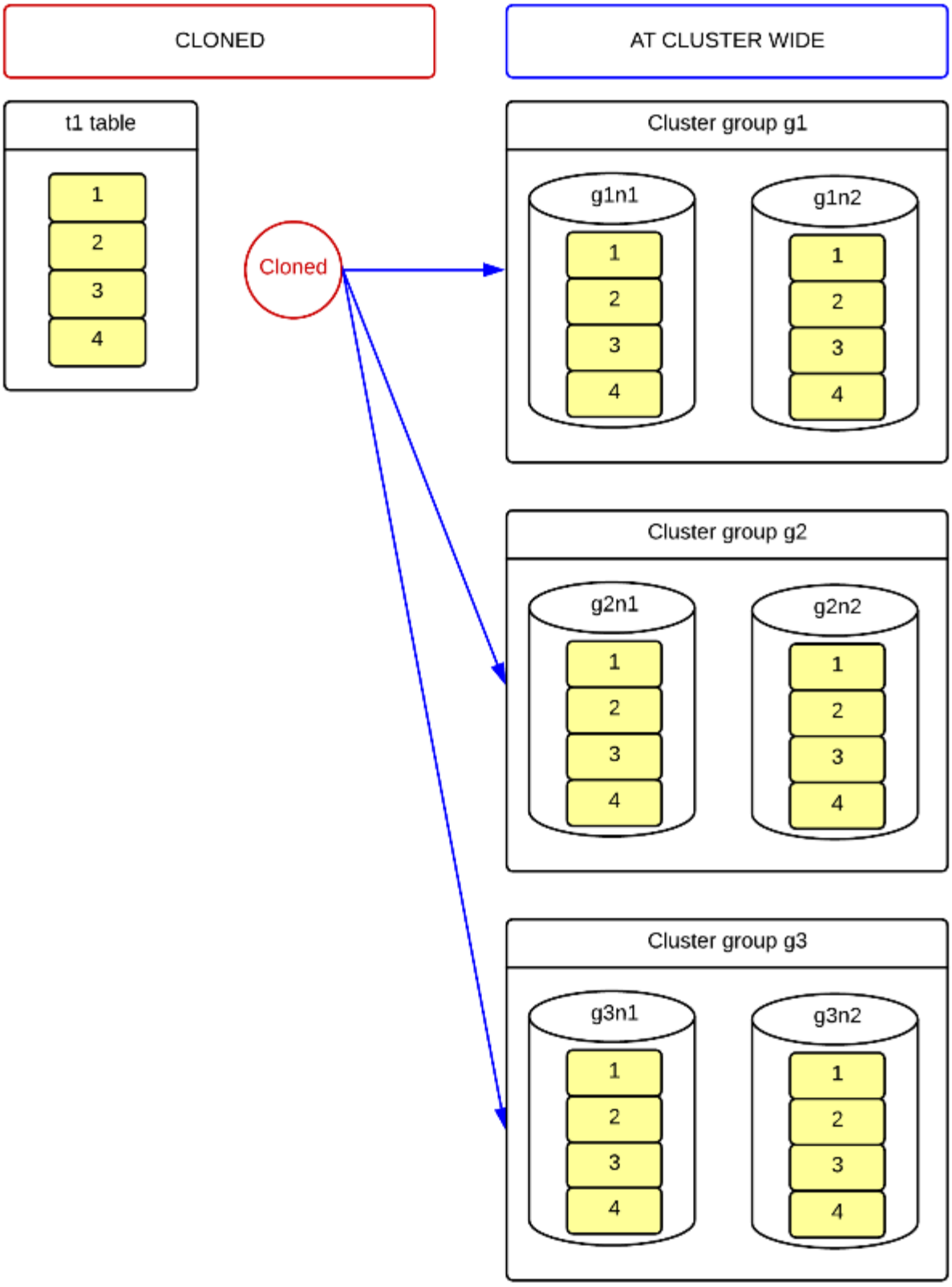
Cloned Table

A cloned table equally duplicates the table data and manages it.

The following is an example of creating a cluster-wide cloned table. All table data is equally duplicated an d placed in 3x2 cluster members.

```
CREATE TABLE t1 ( id INTEGER )
   CLONED
   AT CLUSTER WIDE
;
```

Figure 5 Cluster-wide cloned table

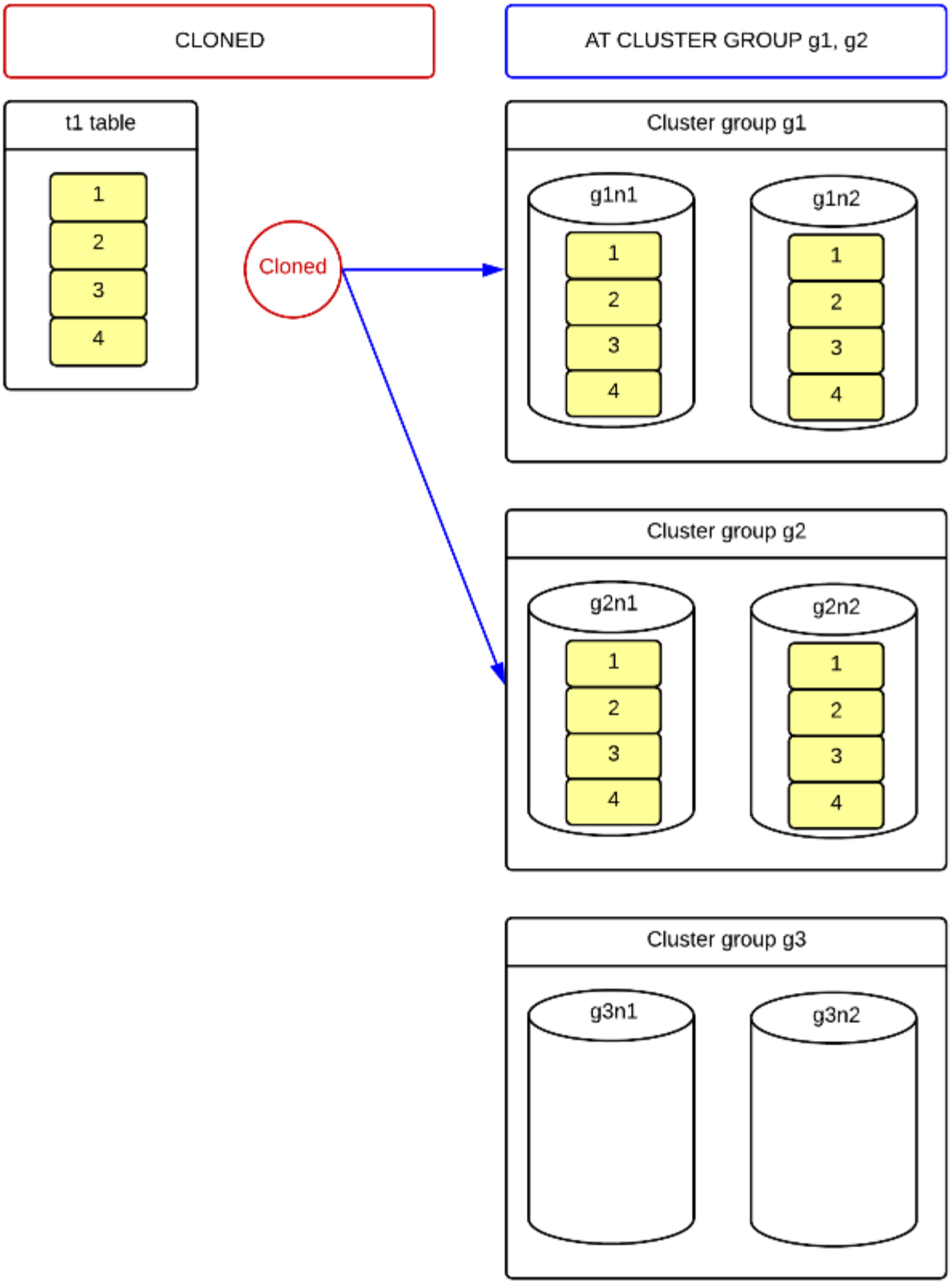


The following is an example of creating a group-specific cloned table. All table data is equally duplicated

and managed, but the duplicated table data exists only in cluster members of group g1 and g2 which are specified by a user, but it does not exist in a group g3.

```
CREATE TABLE t1 ( id INTEGER )
   CLONED
   AT CLUSTER GROUP g1, g2
;
```

Figure 6 Group-specific cloned table



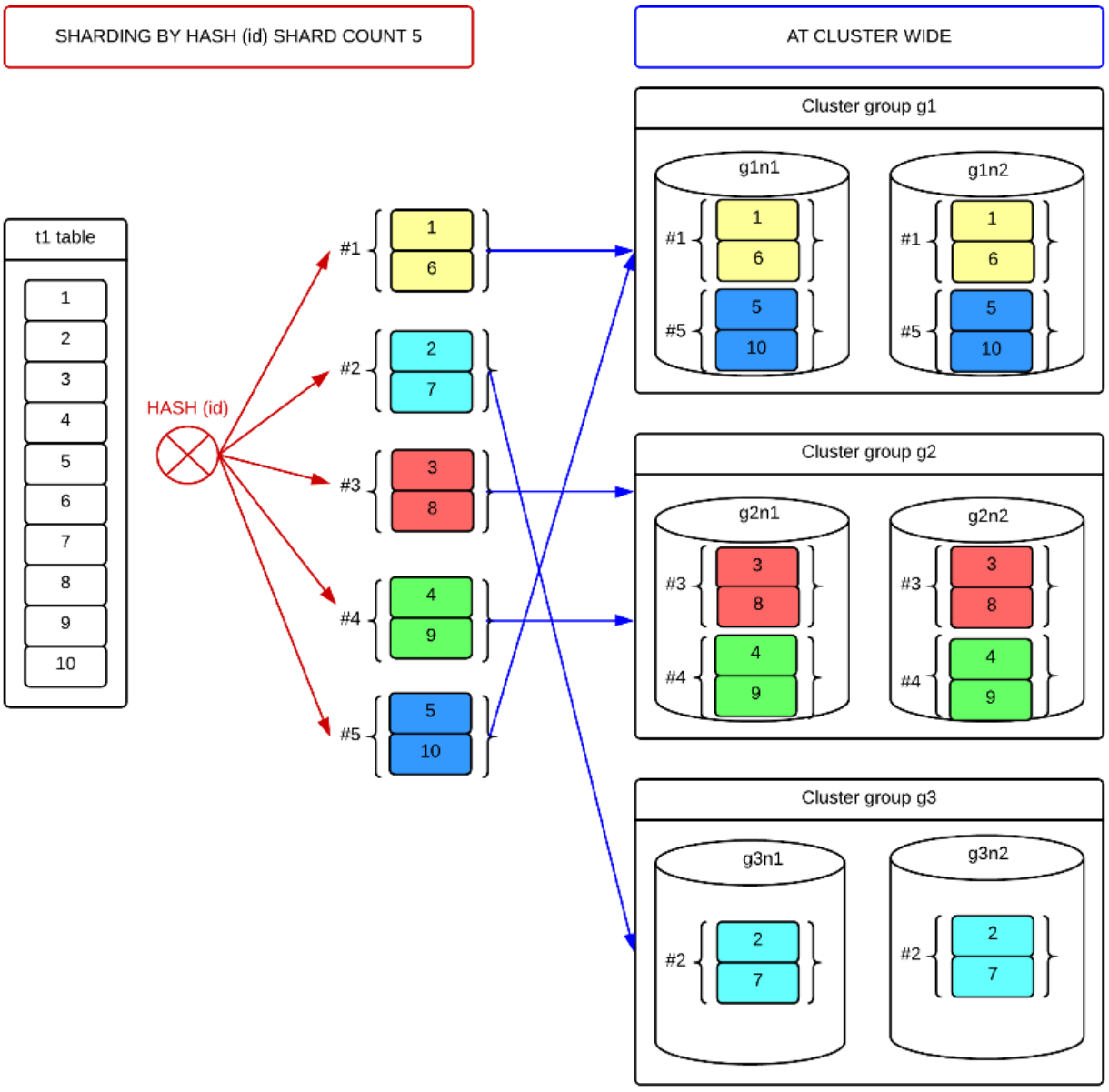
Hash-sharded table

A hash-sharded table divides the table data into several shards based on the hash value of a sharding key, then places them in the cluster system.

The following is an example of creating the cluster-wide hash-sharded table. When adding data to a tabl e, the hash value is created based on the ID column value, and the shard to place row is selected among f ive shards by using the hash value. Each shard is automatically placed. All rows with the same ID column value are included in the same shard, and placed in the same cluster group.

```
CREATE TABLE t1 ( id INTEGER )
   SHARDING BY HASH(id)
   SHARD COUNT 5
   AT CLUSTER WIDE
;
```

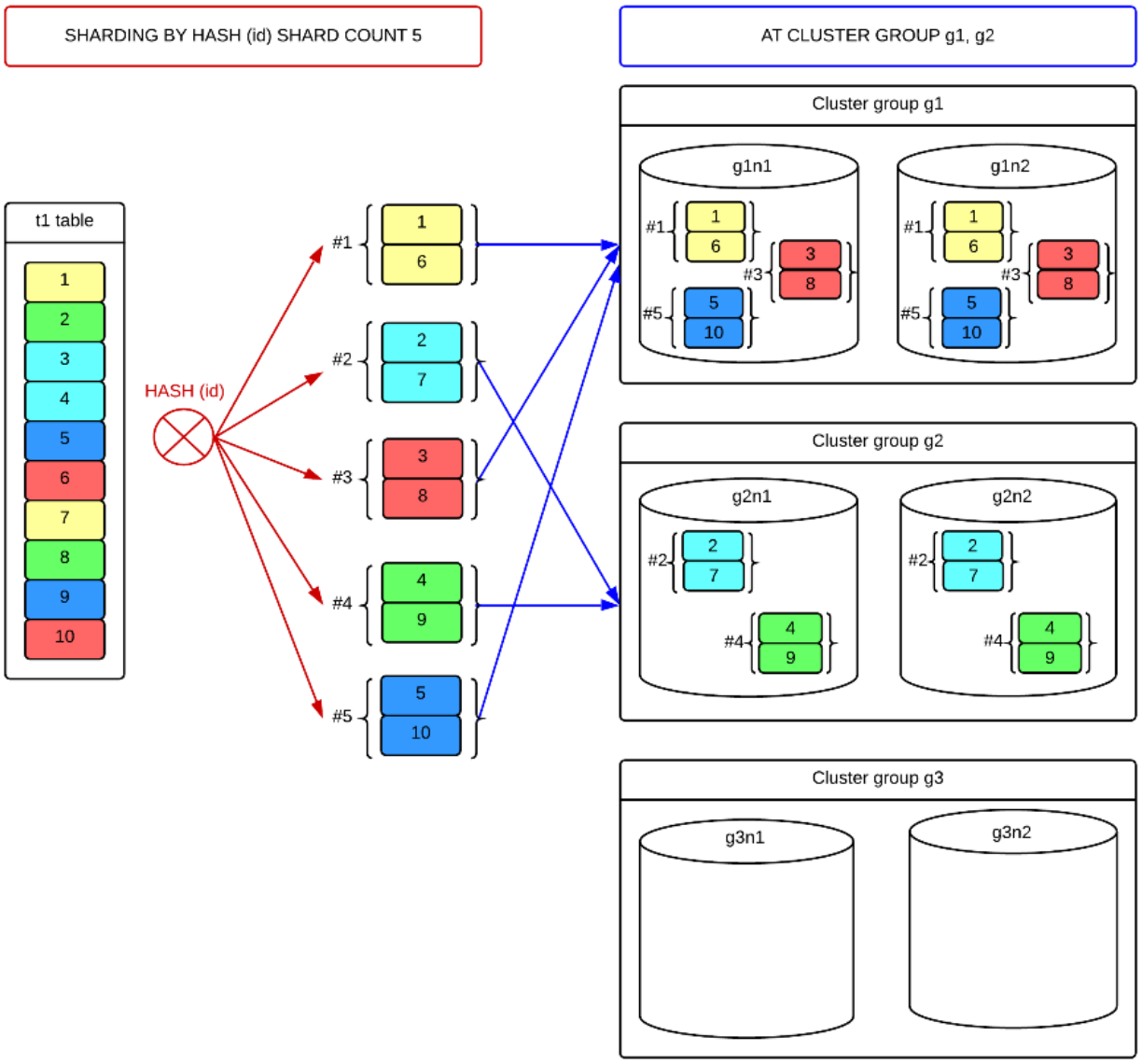
Figure 7 Cluster-wide hash-sharded table



The following is an example of creating the group-specific hash-sharded table. The hash value of an ID co lumn determines the shard, but each shard is placed only in the cluster group g1 and g2 which are specifi ed by a user.

```
CREATE TABLE t1 ( id INTEGER )
   SHARDING BY HASH(id)
   SHARD COUNT 5
   AT CLUSTER GROUP g1, g2
;
```

Figure 8 Group-specific hash-sharded table



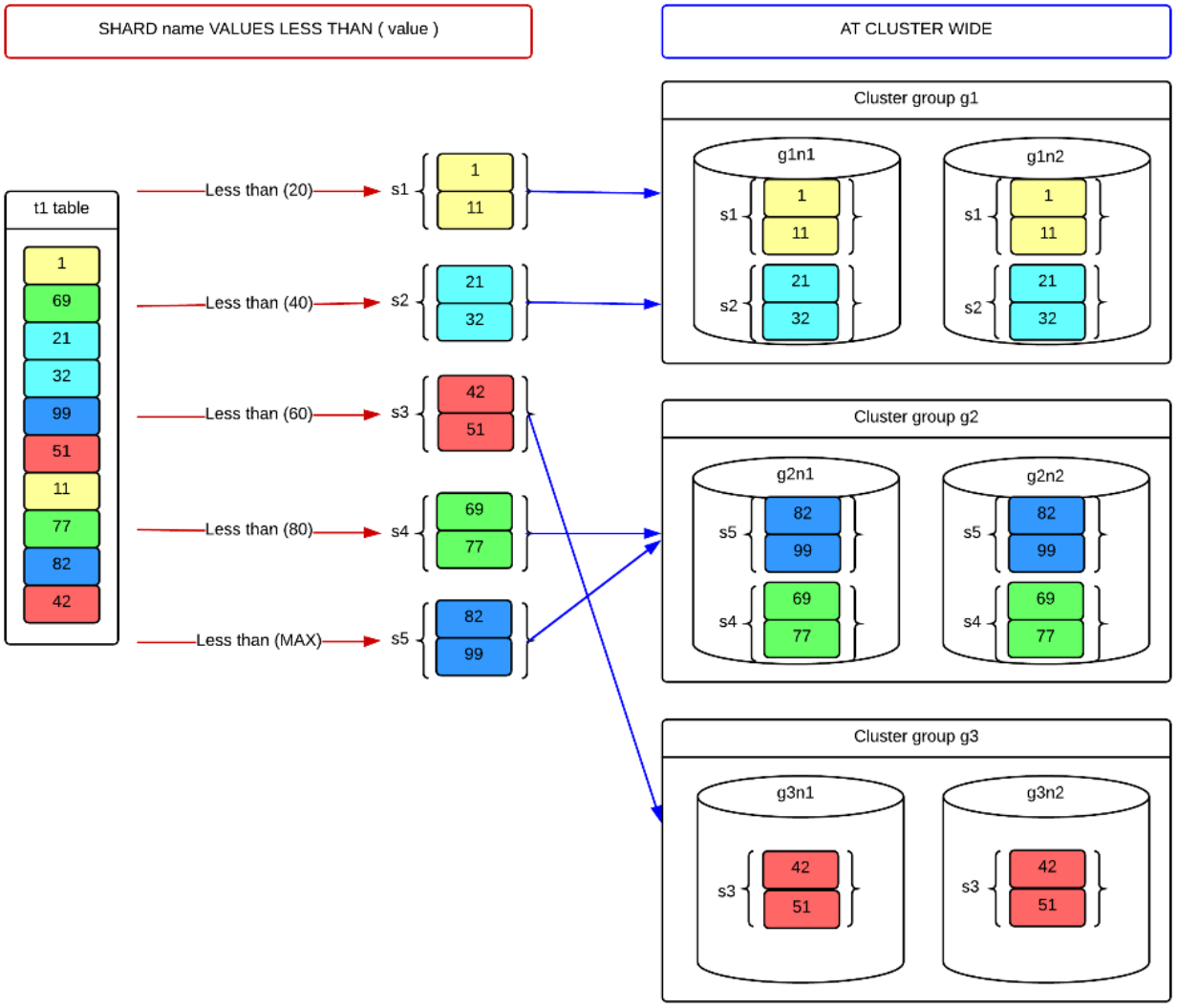
Range-sharded Table

A range-sharded table divides the table data into several shards based on the range value of a sharding key, then places them in the cluster system.

The following is an example of creating the cluster-wide range-sharded table. When adding data to a table, the shard to place row is selected among five shards based on the range value of the ID column. Each shard is automatically placed. All rows of the ID column within the same range are included in the same shard, and placed in the same cluster group.

```
CREATE TABLE t1 ( id INTEGER )
   SHARDING BY RANGE(id)
   AT CLUSTER WIDE
       SHARD s1 VALUES LESS THAN ( 20 ),
       SHARD s2 VALUES LESS THAN ( 40 ),
       SHARD s3 VALUES LESS THAN ( 60 ),
       SHARD s4 VALUES LESS THAN ( 80 ),
       SHARD s5 VALUES LESS THAN ( MAXVALUE )
;
```

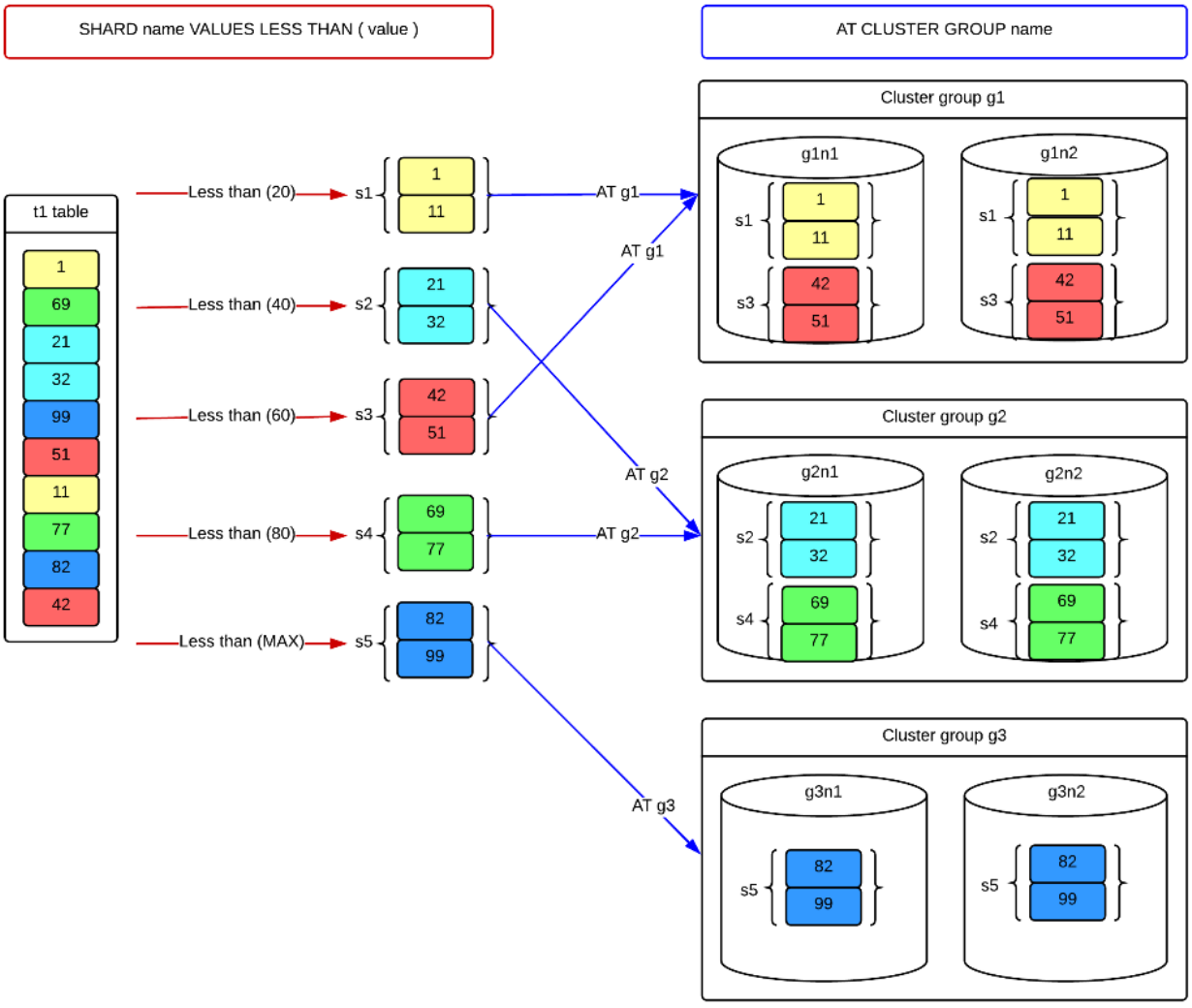
Figure 9 Cluster-wide range-sharded table



The following is an example of creating the group-specific range-sharded table. The range value of the ID column defines the shard, but each shard is placed in a cluster group specified by a user.

```
CREATE TABLE t1 ( id INTEGER )
   SHARDING BY RANGE(id)
       SHARD s1 VALUES LESS THAN ( 20 )       AT CLUSTER GROUP g1,
       SHARD s2 VALUES LESS THAN ( 40 )       AT CLUSTER GROUP g2,
       SHARD s3 VALUES LESS THAN ( 60 )       AT CLUSTER GROUP g1,
       SHARD s4 VALUES LESS THAN ( 80 )       AT CLUSTER GROUP g2,
       SHARD s5 VALUES LESS THAN ( MAXVALUE ) AT CLUSTER GROUP g3
;
```

Figure 10 Group-specific range-sharded table



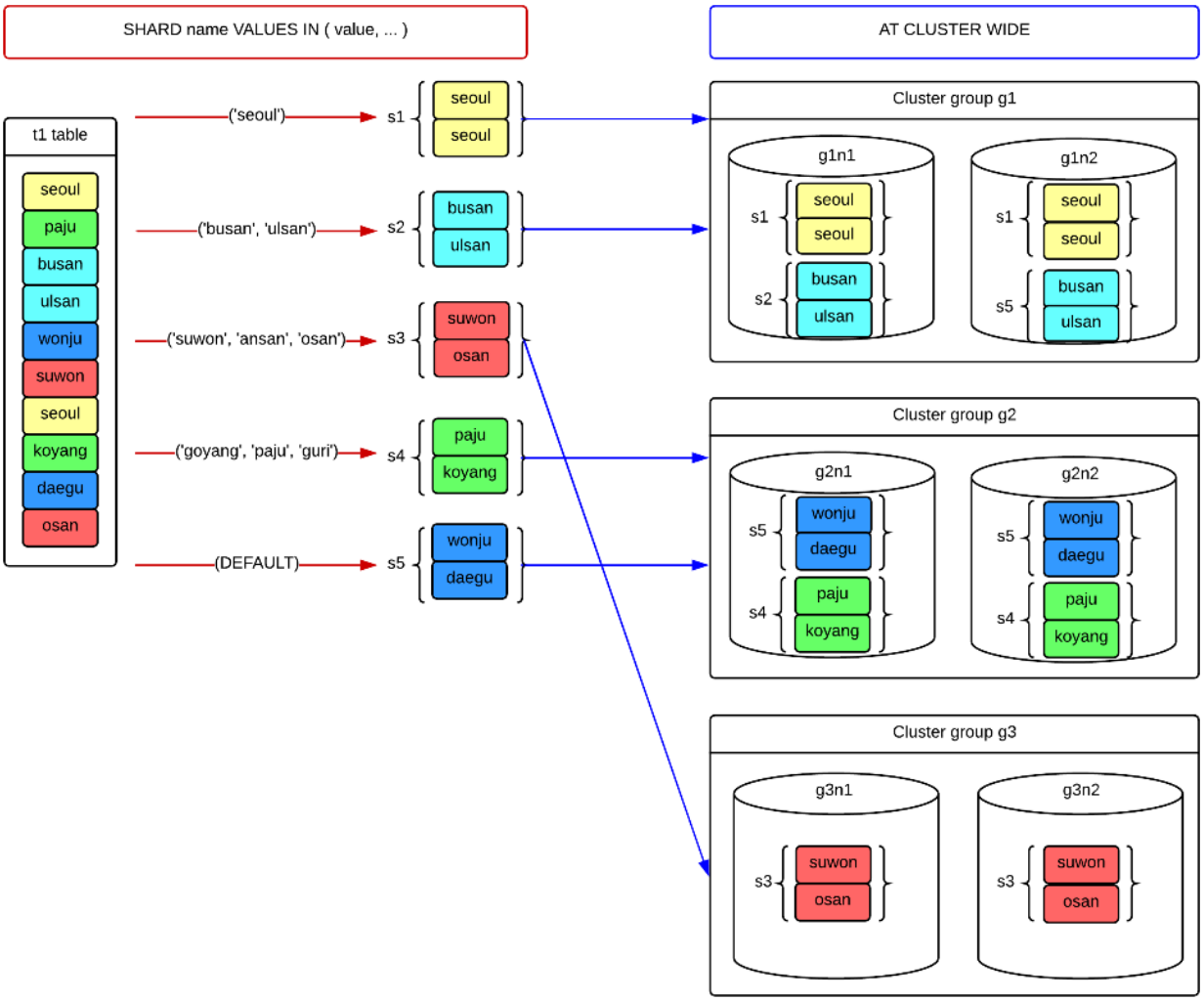
List-sharded Table

A list-sharded table divides the table data into several shards based on the list value of a sharding key, the n places them in the cluster system.

The following is an example of creating the cluster-wide list-sharded table. When adding data to a table, rows are placed in a shard with the list value as same as the CITY column value. Each shard is automatical ly placed.

```
CREATE TABLE t1 ( city VARCHAR(128) )
   SHARDING BY LIST (city)
      AT CLUSTER WIDE
      SHARD s1 VALUES IN ( 'seoul' ),
      SHARD s2 VALUES IN ( 'busan', 'ulsan' ),
      SHARD s3 VALUES IN ( 'suwon', 'ansan', 'osan' ),
      SHARD s4 VALUES IN ( 'goyang', 'paju', 'guri' ),
      SHARD s5 VALUES IN ( DEFAULT )
;
```

Figure 11 Cluster-wide list-sharded table

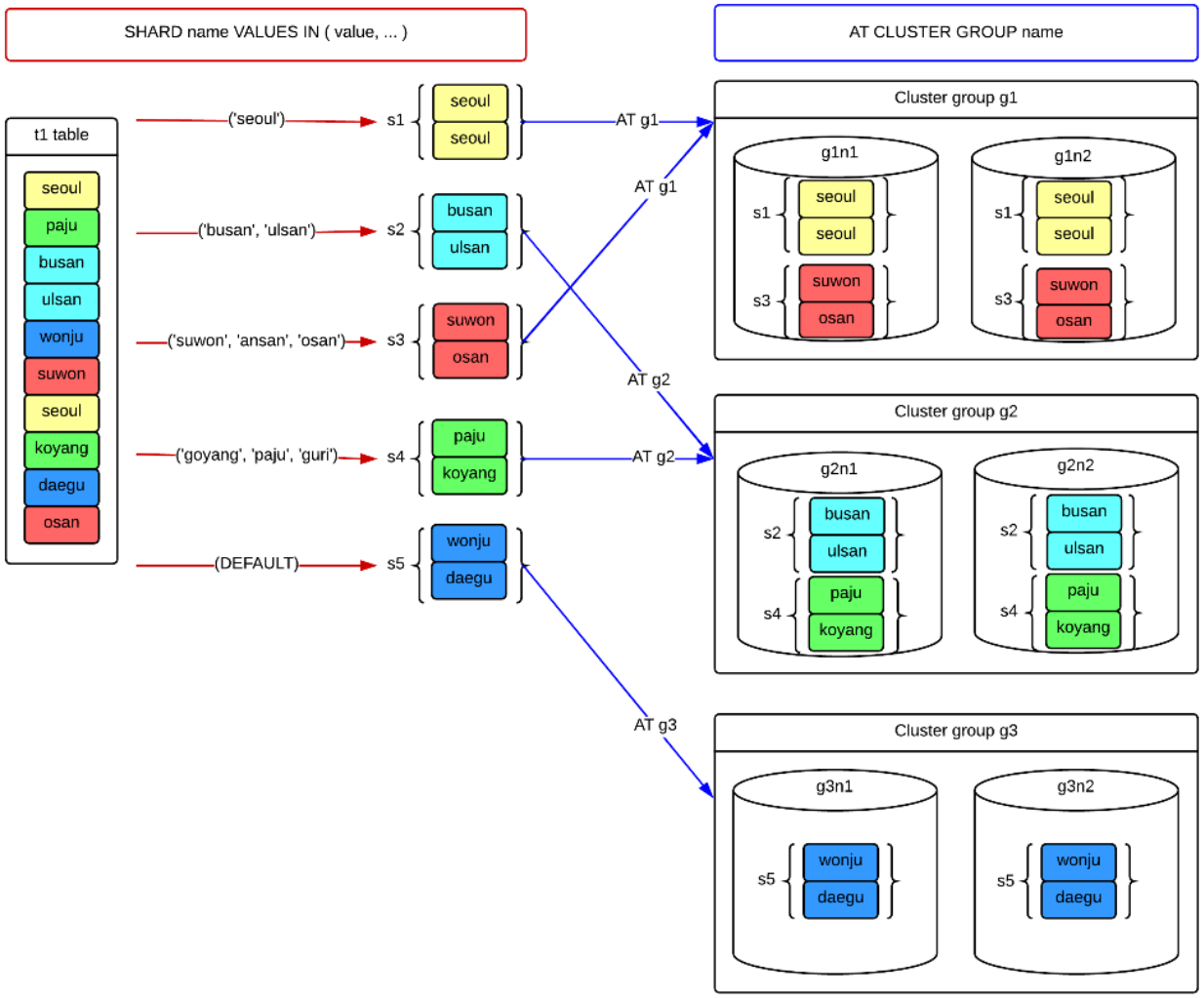


The following is an example of creating the group-specific list-sharded table. The shard is selected by the l

ist value of the CITY column, but each shard is placed in the cluster group specified by a user.

```
CREATE TABLE t1 ( city VARCHAR(128) )
   SHARDING BY LIST (city)
      SHARD s1 VALUES IN ( 'seoul' )                  AT CLUSTER GROUP g1,
      SHARD s2 VALUES IN ( 'busan', 'ulsan' )         AT CLUSTER GROUP g2,
      SHARD s3 VALUES IN ( 'suwon', 'ansan', 'osan' ) AT CLUSTER GROUP g1,
      SHARD s4 VALUES IN ( 'goyang', 'paju', 'guri' ) AT CLUSTER GROUP g2,
      SHARD s5 VALUES IN ( DEFAULT )                  AT CLUSTER GROUP g3
;
```

Figure 12 Group-specific list-sharded table



Rebalancing Cluster Table

Rebalance the data of the cluster table by using **ALTER TABLE name REBALANCE** statement.

Data of the cluster table is rebalanced in the following unit.

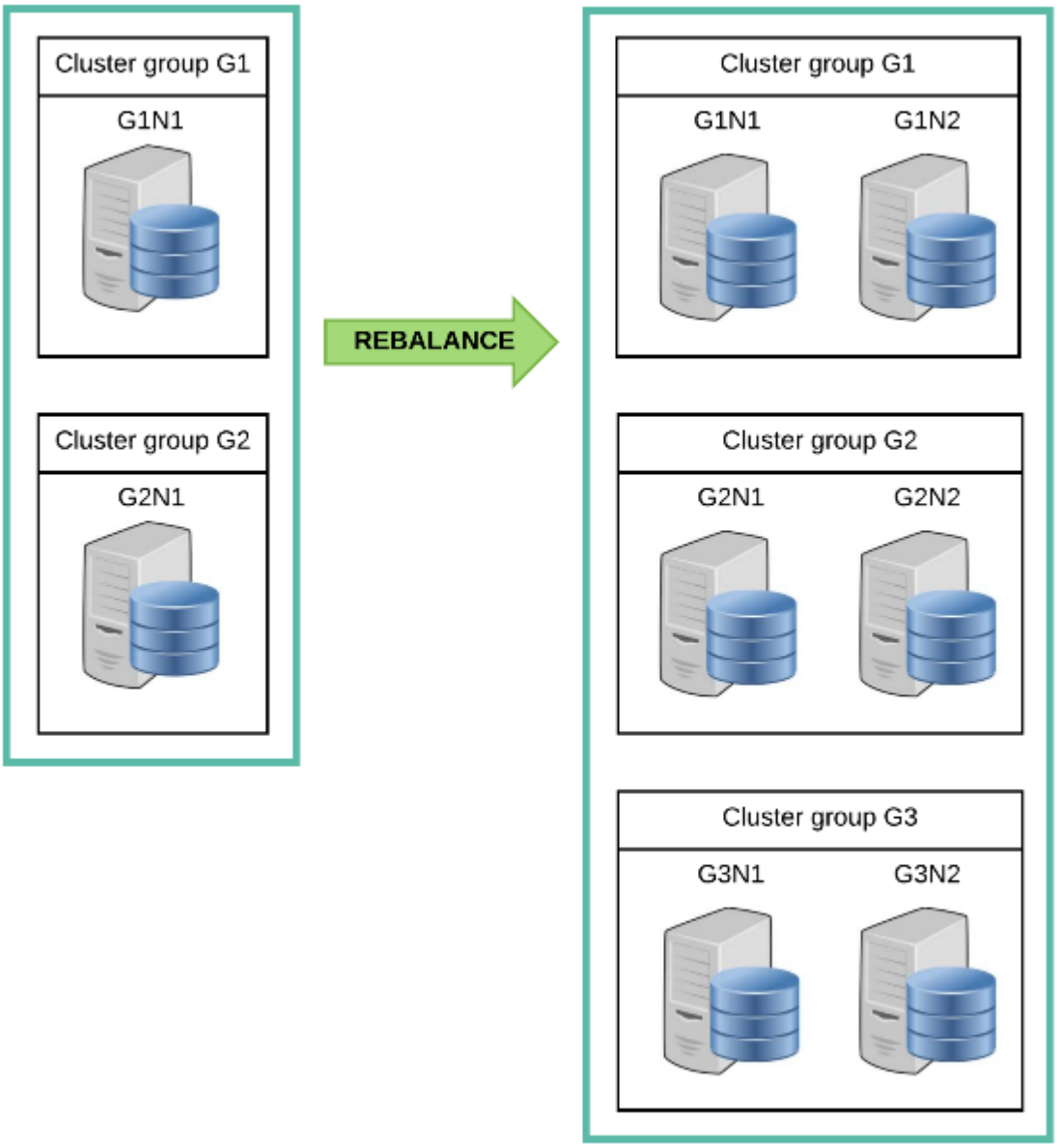
- Cloned table: The entire table
- Sharded table: shard unit

If the table is specified as AT CLUSTER WIDE, the shard is automatically rebalanced to the created cluster group, but if the cluster group to place the shard is specified by AT CLUSTER GROUP, the shard is not reb alanced to the created cluster group.

- When automatically rebalancing by using AT CLUSTER WIDE, then the data can be rebalanced to a n ew cluster group and a new cluster member.

```
CREATE TABLE region
(
    r_regionkey    INTEGER
  , r_name         CHAR(25)
  , r_comment      VARCHAR(152)
)
CLONED
AT CLUSTER WIDE;
```

Figure 13 Rebalancing the region table defined by AT CLUSTER WIDE

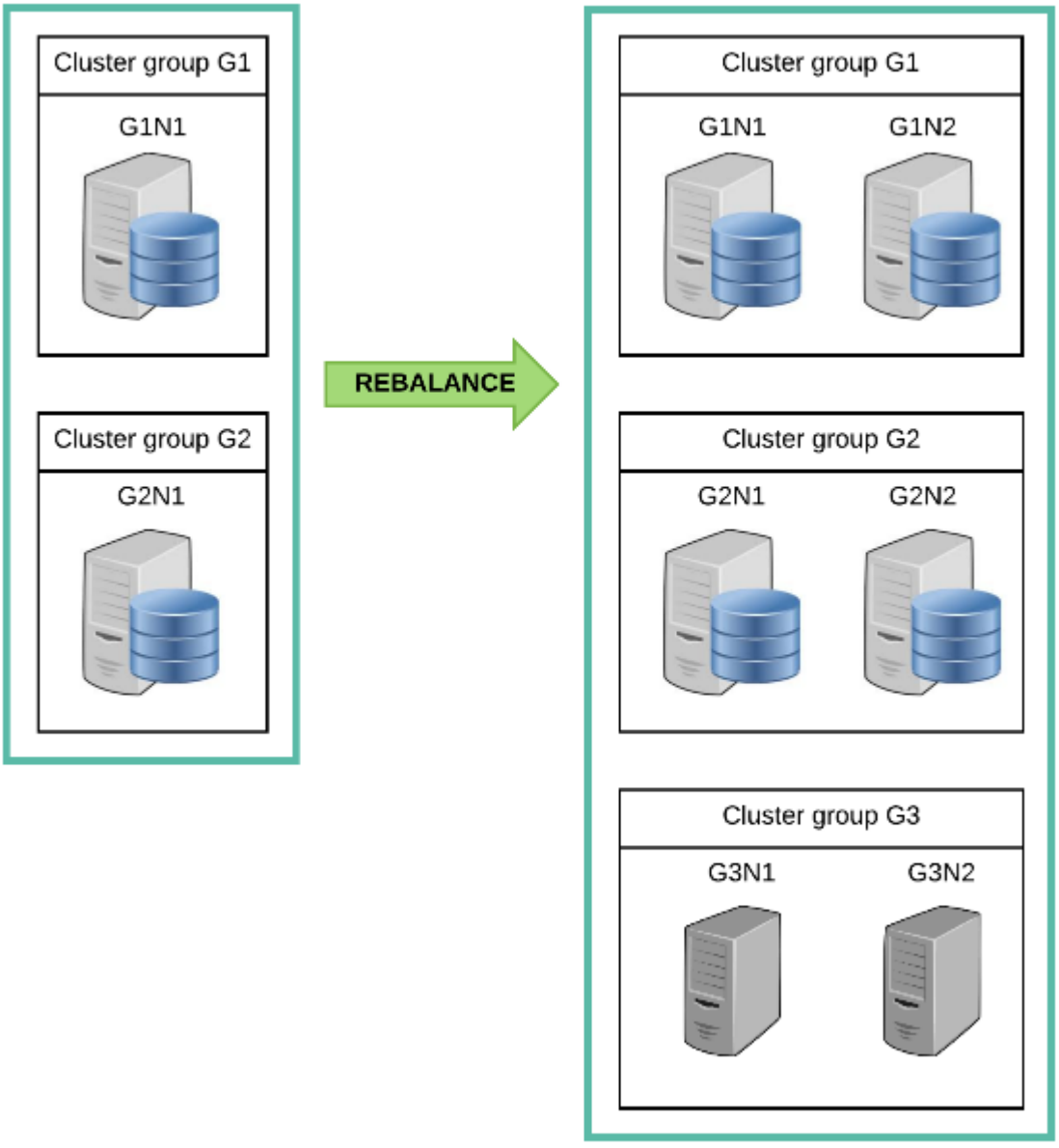


- When specifying the location to place the shard by using AT CLUSTER GROUP
  - The data is not rebalanced on the new cluster group.
  - The data can be rebalanced on the new cluster member which is added to the specified cluster gr oup.

```
CREATE TABLE nation
(
    n_nationkey    INTEGER
  , n_name         CHAR(25)
  , n_regionkey    INTEGER
  , n_comment      VARCHAR(152)
```

```
)
CLONED
AT CLUSTER GROUP g1, g2;
```

Figure 14 Rebalancing the nation table defined by AT CLUSTER GROUP



Information about the table placement can be retrieved through the following views.

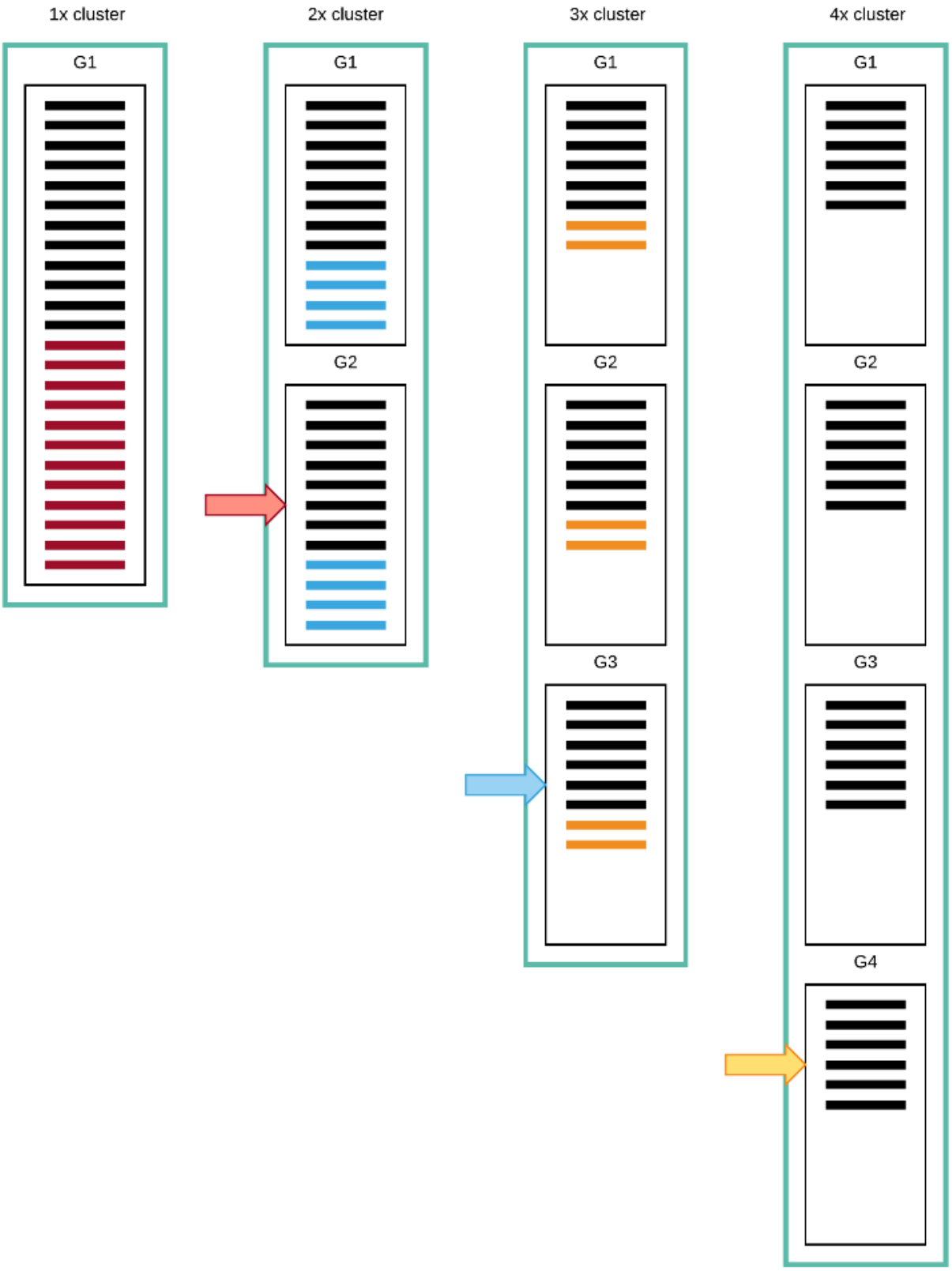
- USER_TAB_PLACE
- ALL_TAB_PLACE

```
gSQL>
SELECT group_name, member_name
  FROM user_tab_place
 WHERE table_name = 'REGION';
GROUP_NAME MEMBER_NAME
---------- -----------
G1         G1N1
G1         G1N2
G2         G2N1
G2         G2N2
G3         G3N1
G3         G3N2
6 rows selected.
```

The sharded table is rebalanced in shard unit, and rebalancing shards due to the increase of groups is as f ollows.

```
CREATE TABLE orders
(
    o_orderkey     INTEGER
  , o_custkey      INTEGER
  , o_orderstatus  CHAR(1)
  , o_totalprice   NUMERIC(12,2)
  , o_orderdate     DATE
  , o_orderpriority CHAR(15)
  , o_clerk        CHAR(15)
  , o_shippriority INTEGER
  , o_comment      VARCHAR(79)
)
SHARDING BY HASH( o_orderkey )
SHARD COUNT 24
AT CLUSTER WIDE
;
```

Figure 15 Rebalancing shards due to the increase of groups



In the example above, the data of orders table is divided into 24 shards and placed. All shards are placed i

n a single group of 1x cluster which has only one group. 12 shards are placed in each group of 2x cluster which has two groups.

When expanding 2x cluster to 3x cluster, shards are transferred from an old group to the new group, but the number of shards in each group is same (8). When expanding 3x cluster to 4x cluster, then each G1, G2, G3 rebalances two shards to the newly created group G4.

In other words, the shard rebalancing minimize the movement when shards are transferred from an old g roup to the new group, then data is equally rebalanced maintaining the same number of shards in each g roup.

Information about the shard placement of the sharded table can be retrieved through the following view s.

- USER_TAB_SHARDS
- ALL_TAB_SHARDS

```
gSQL>
SELECT shard_name, group_name
  FROM user_tab_shards
 WHERE table_name = 'ORDERS';
SHARD_NAME   GROUP_NAME
------------ ----------
SHARD_000000 G1
SHARD_000001 G1
SHARD_000002 G1
SHARD_000003 G1
SHARD_000004 G1
SHARD_000005 G1
SHARD_000006 G1
SHARD_000007 G1
SHARD_000008 G3
SHARD_000009 G3
SHARD_000010 G3
SHARD_000011 G3
SHARD_000012 G2
SHARD_000013 G2
SHARD_000014 G2
SHARD_000015 G2
SHARD_000016 G2
SHARD_000017 G2
SHARD_000018 G2
SHARD_000019 G2
```

```
SHARD_000020 G3
SHARD_000021 G3
SHARD_000022 G3
SHARD_000023 G3
24 rows selected.
```

When tables which have the same <sharding strategy> are completely rebalanced as follows, then the sha rd placement result is same. Rows with the same shard key is guaranteed to be placed in the same shard and the same group even when the tables are different.

- Table t1
  - It is created in the 2x cluster.
  - CREATE TABLE t1 (c1 INTEGER) SHARDING BY (c1);
  - It is rebalanced in the 4x cluster.
  - ALTER TABLE t1 REBALANCE:
- Table t2
  - It is created in the 3x cluster.
  - CREATE TABLE t2 (a1 INTEGER) SHARDING BY (a1);
  - It is rebalanced in the 4x cluster.
  - ALTER TABLE t2 REBALANCE:
- Table t3
  - It is created in the 4x cluster.
  - CREATE TABLE t3 (i1 INTEGER) SHARDING BY (i1);

In other words, the following query can access only to a single cluster member and process it.

```
SELECT COUNT(*)
  FROM t1, t2, t3
 WHERE t1.c1 = t2.a1
   AND t2.a1 = t3.i1
   AND t1.c1 = 1;
```

14.6 Global Secondary Index

Global Secondary Index Related Statements

For more information about creating, dropping, and altering a global secondary index, refer to the follow ings.

- Creating a global secondary index: **ALTER TABLE name ADD GLOBAL SECONDARY INDEX**
- Dropping a global secondary index: **ALTER TABLE name DROP GLOBAL SECONDARY INDEX**
- Altering a global secondary index: **ALTER TABLE name ALTER GLOBAL SECONDARY INDEX**

Information related to a global secondary index can be retrieved through the following views.

Table 14-6 Global secondary index related information

Schema	View	Description
DICTIONARY_SCHEMA	ALL_CLUSTER_TABLES	Existence of a global secondary index in user accessible table
	ALL_GLOBAL_SECONDARY_INDEXES	Object information of user accessible global secondary index
	ALL_GSI_PLACE	Placement information of user accessible global secondary index
	USER_CLUSTER_TABLES	Existence of a global secondary index in user owned table
	USER_GLOBAL_SECONDARY_INDEXES	Object information of user owned global secondary index
	USER_GSI_PLACE	Placement information of user owned global secondary index

Concepts of Global Secondary Index

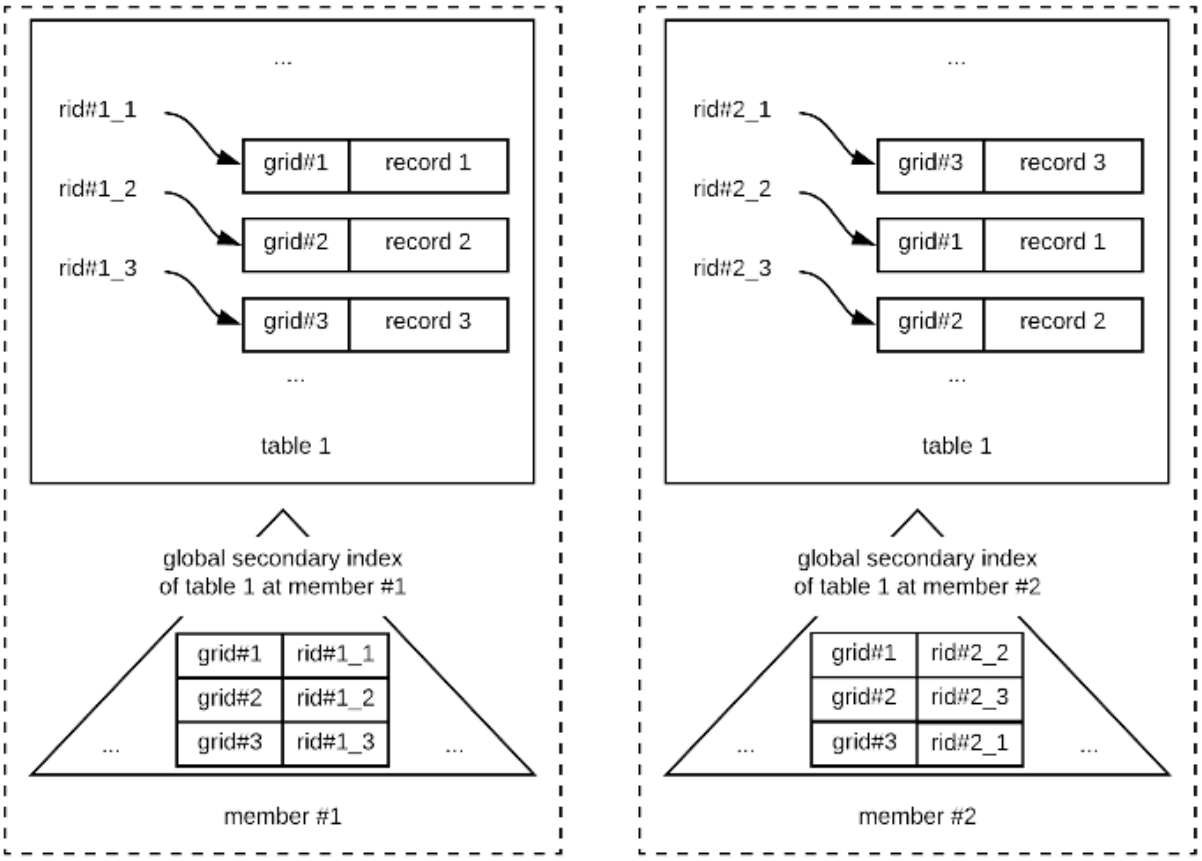
A global secondary index is a B-Tree index which configures the GRID (global row identifier) value as a key in each cluster member of the cluster environment.

Tables in the cluster environment is duplicated to all members in a group and the same records are applied and retrieved when performing DML or select. GRID is a unique value identifying the same records in cluster members, and it is allocated when the record is inserted for the first time, and it is spreaded over all members in a group, then stored together with the record.

When creating a table in a cluster environment, a global secondary index may or may not be created, and

it can be separately created or deleted after the table is created. A table may not have a global secondary index, or it may have maximum one global secondary index.

Figure 16 Structure of a global secondary index



A global secondary index is required to perform a non-deterministic query for a table. If the table dose no t have a global secondary index, then the non-deterministic query fails as follows.

```
gSQL> DELETE FROM T1 LIMIT 1;
ERR-42000(16423): does not support non-deterministic DML in the cluster system : global
secondary index expected
```

Refer to the dictionary such as ALL_GSI_PLACE, DBA_GSI_PLACE, and USER_GSI_PLACE to check if a tabl e has a global secondary index.

15.

SQL Tuning

15.1 SQL Tuning

Overview

SQL tuning is a process of analyzing and modifying a query statement to improve the performance.

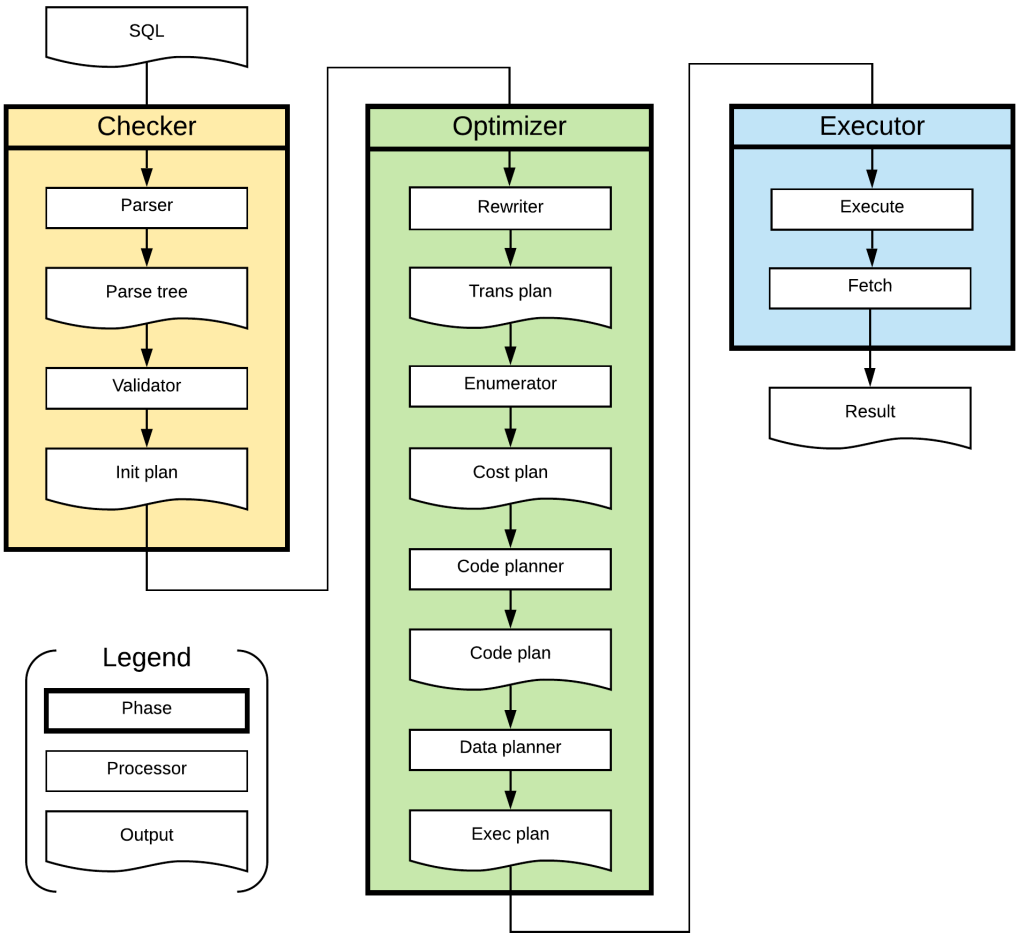
This process helps a query statement reach to the desired performance level by decreasing the response ti me of a query statement or by increasing the throughput.

The knowledge about SQL processing and an optimizer is required to perform SQL tuning, so this chapter describes them.

SQL Processing

The following figure describes the SQL processing.

Figure 1 SQL processing



The user query returns the result through a parser, a validator, a rewriter, an enumerator, a code planner, a data planner, an executor phases. The following paragraphs describes each phase.

Parser

A parser checks a grammatical error of SQL statement input by a user.

```
gSQL> SELECT * FORM customer;
ERR-42000(40000): syntax error
SELECT * FORM customer
.........^  ^
Error at line 1
```

If the SQL statement does not have a grammatical error, then it creates a parse tree. Then it becomes an i nput argument of a validator in the next phase.

Validator

A validator checks semantic error of a parse tree which was input.

For example, it checks whether an object such as a table or a column specified in a query exists, and if a u ser has s privilege to refer to those objects.

```
gSQL> SELECT * FROM customer;
ERR-42000(16040): table or view does not exist :
SELECT * FROM customer
              *
ERROR at line 1:
```

If a parse tree does not have a semantic error, then an init plan is created based on it.

Rewriter

A rewriter converts a SQL statement into a high performance SQL with the same meaning.

It analyzes an init plan, creates a trans plan, converts it to a trans plan in a form whose high performance is expected. The converted trans plan becomes an input argument of enumerator phase.

For more information about converting SQL statement, refer to **Rewriter**.

Enumerator

An enumerator calculates the cost of multiple plans based on the statistics information, and creates the b est cost plan.

For more information about various optimization methods such as an access path to a table a join orderin g and a join method determination, refer to **Enumerator**.

Code Planner

A code planner creates a code plan.

A code plan creates the plan finally selected by an enumerator into a execution plan form. An execution plan consists of tree structure nodes, and it includes the following information.

- How to access each table
- The sequence to refer to tables
- How perform the operation joining tables
- The information about the data filter
- The information about grouping and aggregating the data
- The information about sorting the data

Plan Cache

Code plans created by a code planner are registered in a plan cache. The registered plans are classified ac cording to whether plan cache parameters values match.

Table 15-1 Plan cache parameters

Parameter	Description
Query text	Case-sensitive query text
User information	User ID
Cursor property	Cursor property of a query requiring fetch
Bind parameter	The number of bind parameters and IN/ OUT property of each bind parameter
Enable atomic	Whether to use an atomic insertion
Enable hint error	Whether a validation error occurs in a hint

When a user query is input, then it checks whether the plan with the same plan cache parameters values exist in a plan cache. If exists, it omits parser-validator-rewriter-enumerator-code planner process, and us ed the plan registered in the plan cache.
When using the plan registered in the plan cache in this way, then the cost for parser-validator-rewriter-e numerator-code planner process decreases, so it improves the performance.

The following is an example of queries whose query texts values are different. Though they are same que

ries, but their capitalization is different, so the queries below are not recognized as a same plan.

```
"SELECT * FROM customer"
"select * from customer"
"Select * From customer"
"SELECT * FROM  customer"
```

If a schema object (table, index, view, sequence) referenced by a plan is not committed, then th e plan is not registered.

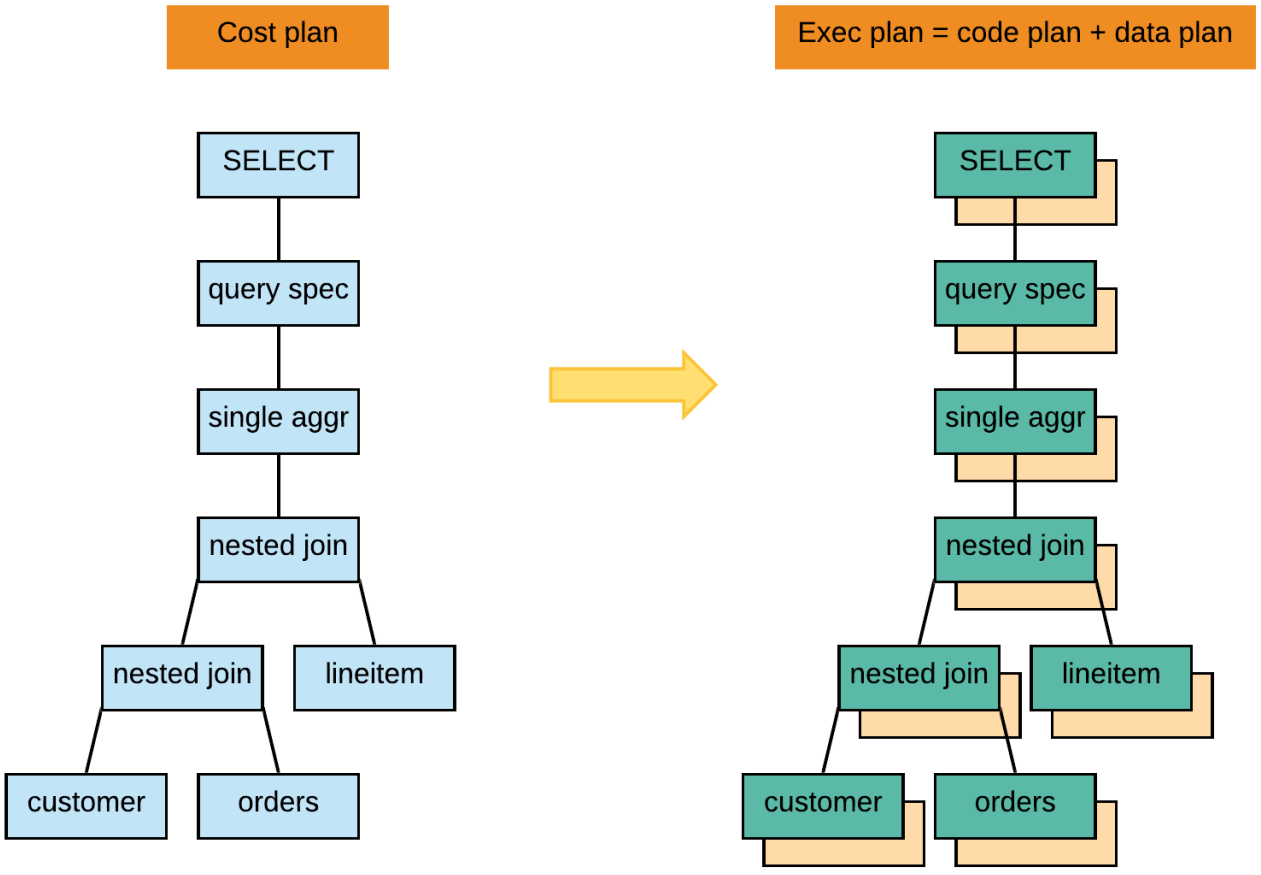
Data Planner

A data planner creates a data plan. A data plan has a space to store the intermediate result and a tempor ary space to store the expression result while performing code plans.

Executor

An executor returns the actual executed result of performing a code plan and a data plan.

Figure 2 Executor



Execution Plan

The execution plan consists of a code plan and a data plan. It is a tree form whose top node is INSERT, DELETE, UPDATE, SELECT statement.

The execution is a basic analysis tool of SQL tuning. The execution plan describes how the query is converted by a rewriter and which access path, join order, join method is selected by an enumerator.

The followings are a syntax and an example to output the execution plan.

Syntax

The following is a syntax to output the execution plan.

```
<explain plan> ::=
      \explain plan [ on ¦ only ] <sql statement>
<sql statement> ::=
        <query expression>
      ¦ <select for update statement>
      ¦ <select statement: single row>
      ¦ <insert statement>
      ¦ <update statement: searched>
      ¦ <delete statement: searched>
```

Invocation and Access Rules

The privilege to access <sql statement> statement is required to perform <explain plan> statement.

Syntax Rules and Parameters

```
\EXPLAIN PLAN ON
\EXPLAIN PLAN
```

When specifying as above, it performs the query then outputs the execution plan.

```
\EXPLAIN PLAN ONLY
```

When specifying as above, it does not perform the query but only outputs the execution plan.

Example

When executing as follows, it performs SQL statement, and outputs the query result and the execution plan together.

The following example is executed in a cluster system which consists of G1(G1N1, G1N2), G2(G2N1, G2

N2) and G3(G3N1, G3N2). The customer is a cloned table and orders is a sharded table whose data is div ided by do_orderkey.

Figure 3 Read plan

```
\EXPLAIN PLAN
  SELECT  c_name,
          o_orderdate,
          o_orderstatus
    FROM  customer,
          orders
   WHERE  c_custkey = o_custkey
     AND  o_orderdate >= date '1995-03-15'
     AND  o_orderdate < date '1995-03-15' + interval '1' month;
...

19343 rows selected.

>>>  start print plan

< Execution Plan >
```

Code plan

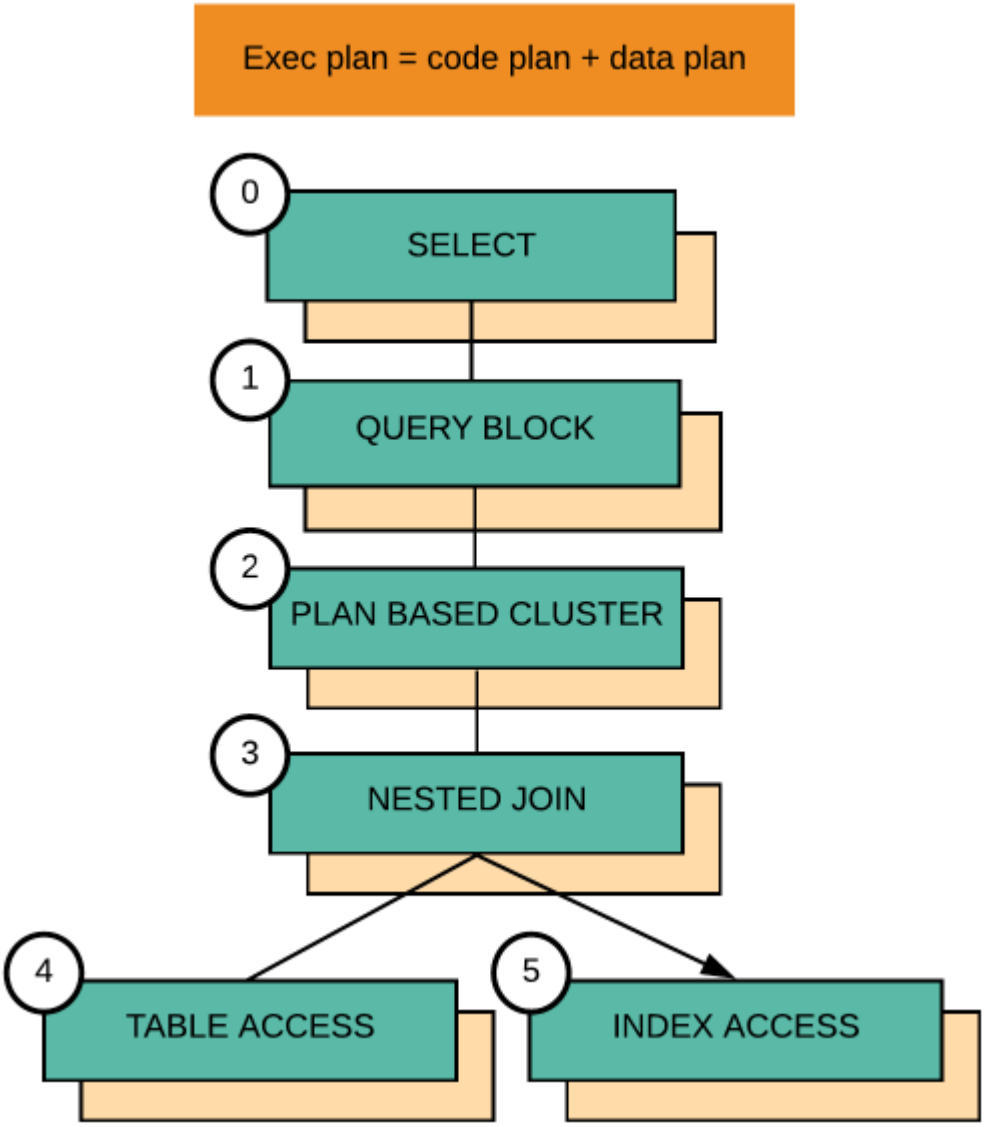
Data plan

IDX	NODE DESCRIPTION		ROWS
0	SELECT STATEMENT		19343
1	QUERY BLOCK ("$QB_IDX_2")		19343
2	PLAN BASED CLUSTER	LOCAL/REMOTE	19343
3	NESTED JOIN (INNER JOIN)		6389
4	TABLE ACCESS ("ORDERS")		6389
5	INDEX ACCESS ("CUSTOMER", "CUSTOMER_PK_INDEX")	(6389)	6389

```
     1  -  TARGET : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE
                            USE_NL_IN( _A1 )
                            FULL( _A2 )
                            INDEX( _A1, "PUBLIC"."CUSTOMER_PK_INDEX" )
                       */
                        "_A1"."C_NAME", "_A2"."O_ORDERDATE", "_A2"."O_ORDERSTATUS"
                  FROM ( "PUBLIC"."ORDERS"@LOCAL AS "_A2"
                         INNER JOIN
                         "PUBLIC"."CUSTOMER"@LOCAL AS "_A1" ON true
                        ) ALIAS "_A3"
                 WHERE "_A2"."O_ORDERDATE" < :_VO
                   AND "_A2"."O_ORDERDATE" >= :_V1
                   AND "_A1"."C_CUSTKEY" = "_A2"."O_CUSTKEY"
           TARGET DOMAIN : G1(G1N1,G1N2) 6389 rows, G2(G2N1,G2N2) 6462 rows, G3(G3N1,G3N2) 6492 rows
     3  -  JOINED COLUMN : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     4  -  HASH SHARD ( # 3 )
           READ COLUMN : ORDERS.O_CUSTKEY, ORDERS.O_ORDERSTATUS, ORDERS.O_ORDERDATE
             PHYSICAL FILTER : ORDERS.O_ORDERDATE < DATE'1995-03-15' + CAST( '1' AS INTERVAL(MONTH) )
                               AND ORDERS.O_ORDERDATE >= DATE'1995-03-15'
     5  -  CLONED
           READ INDEX COLUMN : CUSTOMER.C_CUSTKEY
           READ TABLE COLUMN : CUSTOMER.C_NAME
             MIN RANGE : CUSTOMER.C_CUSTKEY = {ORDERS.O_CUSTKEY}
             MAX RANGE : CUSTOMER.C_CUSTKEY = {ORDERS.O_CUSTKEY}
           FETCH ONE ROW

<<<  end print plan
```

The execution plan above is represented as the following tree. The execution starts from the bottom nod e.

Figure 4 Read plan tree



The execution tree above is performed as follows.

First, PLAN BASED CLUSTER(IDX 2) transfers the following SQL to G1, G2 and G3.

```
SELECT /*+ KEEP_JOINED_TABLE
           USE_NL_IN( _A1 )
           FULL( _A2 )
           INDEX( _A1, "PUBLIC"."CUSTOMER_PK_INDEX" )
       */
       "_A1"."C_NAME", "_A2"."O_ORDERDATE", "_A2"."O_ORDERSTATUS"
  FROM ( "PUBLIC"."ORDERS"@LOCAL AS "_A2"
         INNER JOIN
         "PUBLIC"."CUSTOMER"@LOCAL AS "_A1" ON true
       ) ALIAS "_A3"
```

```
WHERE "_A2"."O_ORDERDATE" < :_V0
  AND "_A2"."O_ORDERDATE" >= :_V1
  AND "_A1"."C_CUSTKEY" = "_A2"."O_CUSTKEY"
```

TABLE ACCESS(IDX:4) and INDEX ACCESS(IDX:5) read data from customer table and orders table and perform NESTED JOIN(IDX 3) in G1, G2, G3.

PLAN BASED CLUSTER brings all NESTED JOIN(IDX 3) results of G1, G2, G3 to local.

Then, it returns the result.

Execution Plan Information

The information about each column in the execution plan table is as follows.

- IDX
  - It is an identifier given to each plan node.
- NODE DESCRIPTION
  - It is the name of a plan node.
  - The contents in brackets are additional information which distinguishes plan nodes.
  - The indented plan node means the subordinate plan node.
    - The execution starts from the bottom plan node, and the result is transferred to the superordinate node.
- ROWS
  - It is the number of result records according to the plan node execution.

Each plan node contains the following optimization information.

- **Access Paths** to each table
- The sequence to process **Join** and join method
- The information about **Group By** processing
- The information about **Distinct** processing
- The information about **Single Row Aggregation** processing
- The information about **Order By** processing
- The information about **Cluster Puller** processing
- The information about **Cluster Pusher** processing

15.2 Rewriter

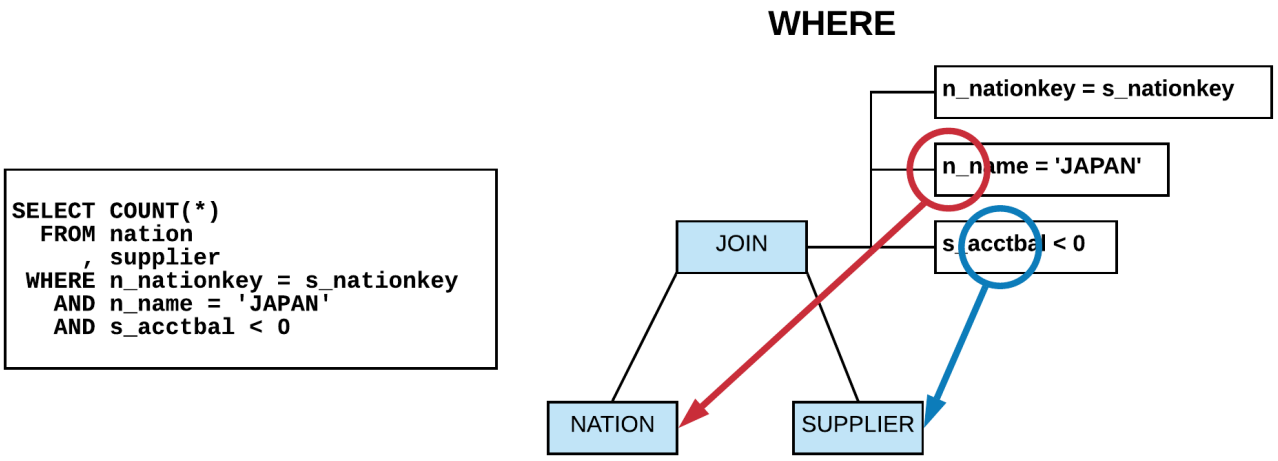
This chapter describes various query transformation methods processed by a rewriter.

Filter Push Down

It pushes down the filter as down as possible so that it reduces the intermediate result to be processed.

The following is an example of filter push down.

Figure 5 Filter push down



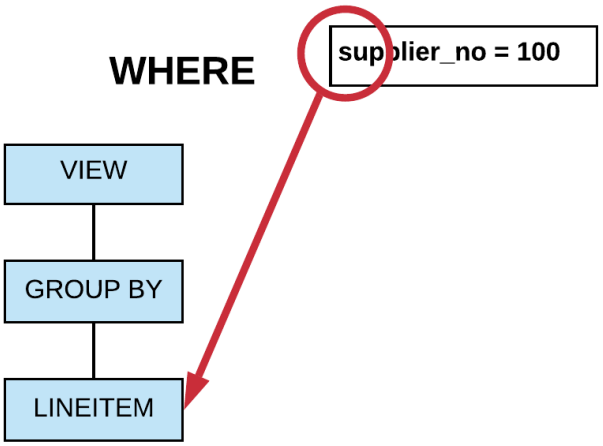
It filters rows satisfying n_name = JAPAN condition in NATION, and filters rows satisfying s_acctbal < 0 co ndition in SUPPLIER before performing join. In this case the number of join target rows decreases so it im proves the performance.

The following is an example of performing filter push down into the view.

Figure 6 Filter push down into view

```
CREATE VIEW revenue (supplier_no, total_revenue) AS
  SELECT l_suppkey,
         ROUND( sum(l_extendedprice * (1 - l_discount)), 2)
    FROM lineitem
   WHERE l_shipdate >= date '1996-01-01'
     AND l_shipdate < date '1996-01-01' + interval '3' month
GROUP BY l_suppkey;
```

```
SELECT total_revenue
  FROM revenue
     , supplier
 WHERE supplier_no = 100
```



When pushing it down to lineitem TABLE ACCESS node after converting supplier_no = 100 to l_suppkey = 100, then target rows of GROUP BY decreases so it improves the performance.

DISTINCT Elimination

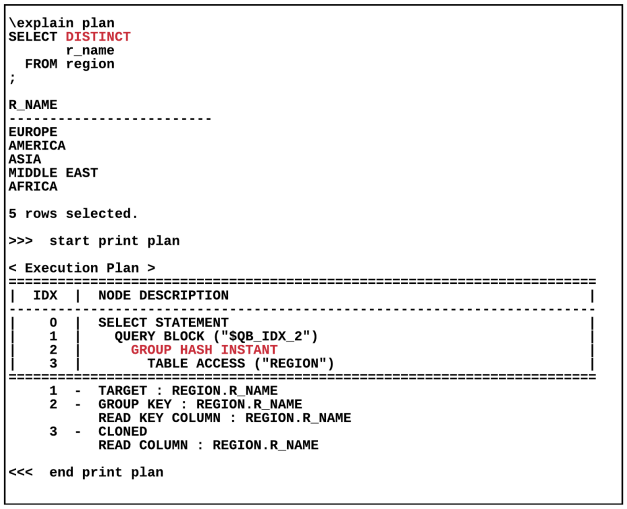
It eliminates the unnecessary DISTINCT.

DISTINCT is eliminated in the following cases.

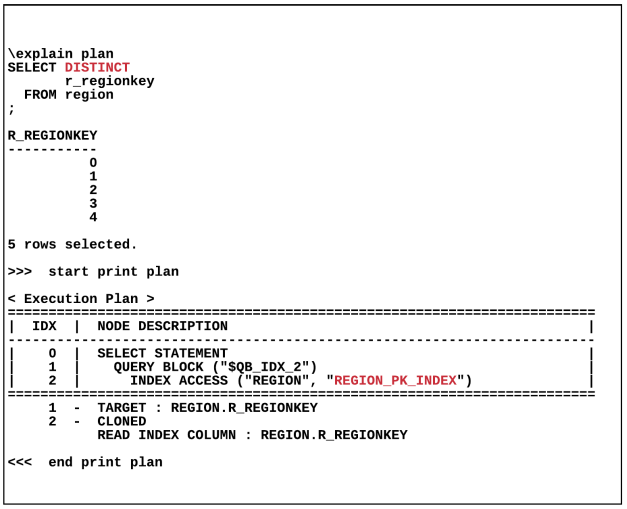
- The query result of a single-row aggregation is one, so a duplicate results does not occur even withou t DISTINCT.
- If *group by* exists and all key columns of *group by* exist in a select list, then the result is unique so a d uplicate results does not occur even without DISTINCT.
- If all key columns in a primary key exist in a select list, then the result is unique so a duplicate results d oes not occur even without DISTINCT.

The following is an example of eliminating DISTINCT.

Figure 7 DISTINCT elimination



```
\explain plan
SELECT DISTINCT
       r_name
  FROM region
;

R_NAME
-------------------------
EUROPE
AMERICA
ASIA
MIDDLE EAST
AFRICA

5 rows selected.

>>>  start print plan

< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("SQB_IDX_2")                                      |
|    2  |      GROUP HASH INSTANT                                           |
|    3  |        TABLE ACCESS ("REGION")                                    |
=============================================================================
     1  -  TARGET : REGION.R_NAME
     2  -  GROUP KEY : REGION.R_NAME
           READ KEY COLUMN : REGION.R_NAME
     3  -  CLONED
           READ COLUMN : REGION.R_NAME

<<<  end print plan
```



```
\explain plan
SELECT DISTINCT
       r_regionkey
  FROM region
;

R_REGIONKEY
-----------
          0
          1
          2
          3
          4

5 rows selected.

>>>  start print plan

< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("SQB_IDX_2")                                      |
|    2  |      INDEX ACCESS ("REGION", "REGION_PK_INDEX")                   |
=============================================================================
     1  -  TARGET : REGION.R_REGIONKEY
     2  -  CLONED
           READ INDEX COLUMN : REGION.R_REGIONKEY

<<<  end print plan
```

The left execution plan has GROUP HASH INSTANT node to process DISTINCT, but the right execution pla n does not have GROUP HASH INSTANT node to process DISTINCT.

r_regionkey is a primary key, so it is guaranteed that the result is distinct even when DISTINCT is eliminate d. Therefore, the unnecessary DISTINCT is eliminated.

ORDER BY Elimination

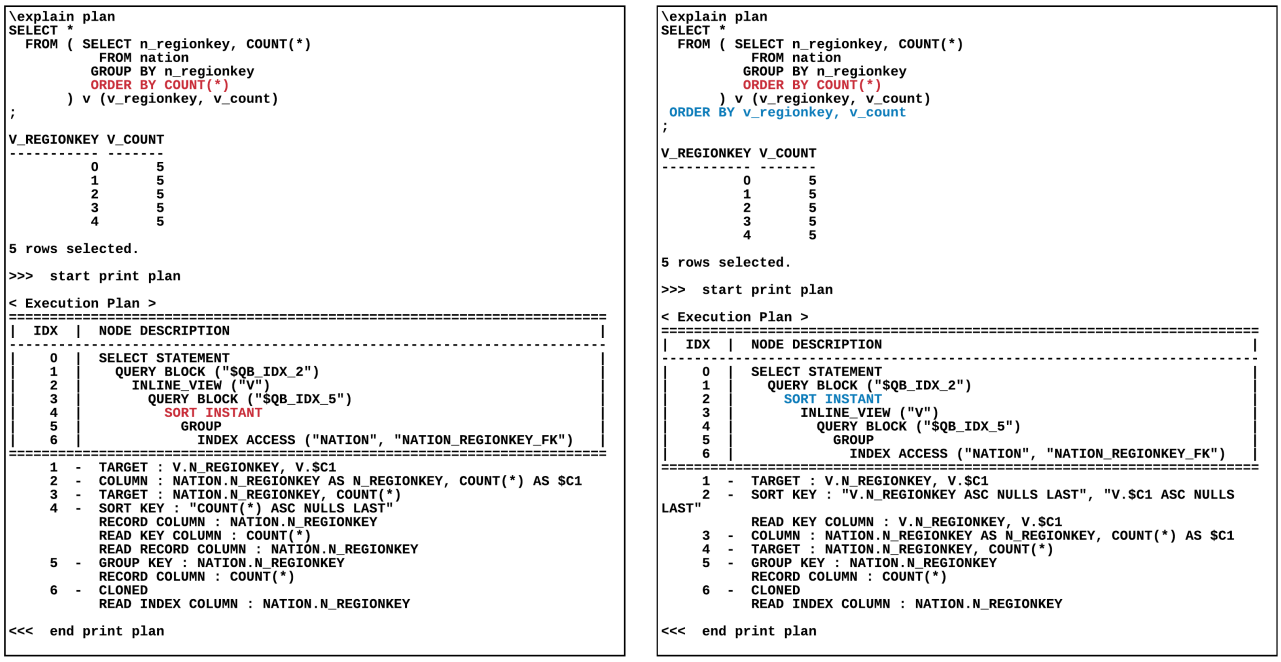
It eliminates unnecessary ORDER BY.

ORDER BY is not required in the following cases.

- When only a view exists in *from* clause, and *group by* clause, *distinct* clause or *order by* clause exists, then *order by* exists in a query block within a view is not necessary. It is because the sequence ordere d by *group by*, *distinct*, *order by* disappears even though it was ordered.
- *order by* exists in a a query block within a subquery is not necessary. It is because the subquery result i s used only when determining whether to return the row of an outer query.

The following is an example of eliminating ORDER BY.

Figure 8 ORDERBY elimination



```
\explain plan
SELECT *
  FROM ( SELECT n_regionkey, COUNT(*)
           FROM nation
         GROUP BY n_regionkey
         ORDER BY COUNT(*)
       ) v (v_regionkey, v_count)
;

V_REGIONKEY V_COUNT
----------- -------
          0       5
          1       5
          2       5
          3       5
          4       5

5 rows selected.

>>>  start print plan

< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
-----------------------------------------------------------------------------
|     0 |  SELECT STATEMENT                                                 |
|     1 |    QUERY BLOCK ("$QB_IDX_2")                                      |
|     2 |      INLINE_VIEW ("V")                                            |
|     3 |        QUERY BLOCK ("$QB_IDX_5")                                  |
|     4 |          SORT INSTANT                                             |
|     5 |            GROUP                                                  |
|     6 |              INDEX ACCESS ("NATION", "NATION_REGIONKEY_FK")       |
=============================================================================
     1  -  TARGET : V.N_REGIONKEY, V.$C1
     2  -  COLUMN : NATION.N_REGIONKEY AS N_REGIONKEY, COUNT(*) AS $C1
     3  -  TARGET : NATION.N_REGIONKEY, COUNT(*)
     4  -  SORT KEY : "COUNT(*) ASC NULLS LAST"
           RECORD COLUMN : NATION.N_REGIONKEY
           READ KEY COLUMN : COUNT(*)
           READ RECORD COLUMN : NATION.N_REGIONKEY
     5  -  GROUP KEY : NATION.N_REGIONKEY
           RECORD COLUMN : COUNT(*)
     6  -  CLONED
           READ INDEX COLUMN : NATION.N_REGIONKEY

<<<  end print plan
```

```
\explain plan
SELECT *
  FROM ( SELECT n_regionkey, COUNT(*)
           FROM nation
         GROUP BY n_regionkey
         ORDER BY COUNT(*)
       ) v (v_regionkey, v_count)
 ORDER BY v_regionkey, v_count
;

V_REGIONKEY V_COUNT
----------- -------
          0       5
          1       5
          2       5
          3       5
          4       5

5 rows selected.

>>>  start print plan

< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
-----------------------------------------------------------------------------
|     0 |  SELECT STATEMENT                                                 |
|     1 |    QUERY BLOCK ("$QB_IDX_2")                                      |
|     2 |      SORT INSTANT                                                 |
|     3 |        INLINE_VIEW ("V")                                          |
|     4 |          QUERY BLOCK ("$QB_IDX_5")                                |
|     5 |            GROUP                                                  |
|     6 |              INDEX ACCESS ("NATION", "NATION_REGIONKEY_FK")       |
=============================================================================
     1  -  TARGET : V.N_REGIONKEY, V.$C1
     2  -  SORT KEY : "V.N_REGIONKEY ASC NULLS LAST", "V.$C1 ASC NULLS
LAST"
           READ KEY COLUMN : V.N_REGIONKEY, V.$C1
     3  -  COLUMN : NATION.N_REGIONKEY AS N_REGIONKEY, COUNT(*) AS $C1
     4  -  TARGET : NATION.N_REGIONKEY, COUNT(*)
     5  -  GROUP KEY : NATION.N_REGIONKEY
           RECORD COLUMN : COUNT(*)
     6  -  CLONED
           READ INDEX COLUMN : NATION.N_REGIONKEY

<<<  end print plan
```

The left and right views are same in the figure above, but the right SQL has *order by* in a superordinate q uery of the view, so the *order by* within the view is eliminated.

Simple View Merging

It merges a simple view which does not include *group by*, *distinct*, *aggregation* into a superordinate quer y block.

Applying a simple view merging enables an optimizer to select various access path, join ordering, join me thod, so it can acquire the better execution plan.

A simple view merging can not be applied to the following cases.

- When a query block within the view includes the followings
  - Set operator
  - LIMIT, OFFSET
  - DISTINCT
  - GROUP BY
  - Single row aggregation
  - Full outer join
  - Natural join
  - ROWNUM
  - When a subquery expression exists in SELECT list
- When a view participates in the following query.

- A view participate in a full outer join.
- A view participate in a left outer join, and two or more tables exist within a view.

The following is an example of merging a simple view.

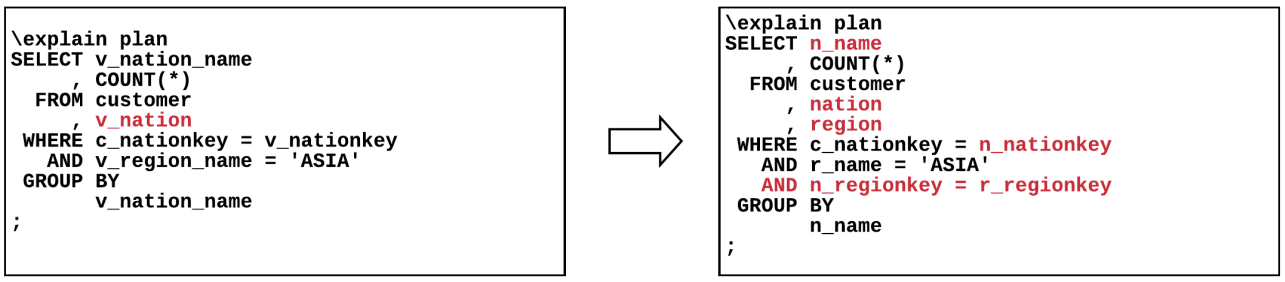
```
CREATE VIEW v_nation
(
    v_nationkey,
    v_nation_name,
    v_region_name
)
AS
SELECT n_nationkey,
       n_name,
       r_name
  FROM nation, region
 WHERE n_regionkey = r_regionkey;
```

```
\EXPLAIN PLAN
SELECT v_nation_name, COUNT(*)
  FROM customer, v_nation
 WHERE c_nationkey = v_nationkey
   AND v_region_name = 'ASIA'
GROUP BY v_nation_name;
V_NATION_NAME             COUNT(*)
------------------------- --------
CHINA                         6024
INDIA                         6042
INDONESIA                     6161
JAPAN                         5948
VIETNAM                       6008
5 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                     ¦
--------------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                    ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                       ¦
¦    2  ¦      GROUP HASH INSTANT                                            ¦
¦    3  ¦        NESTED JOIN (INNER JOIN)                                    ¦
```

```
|    4  |          NESTED JOIN (INNER JOIN)                                |
|    5  |            TABLE ACCESS ("REGION")                               |
|    6  |            INDEX ACCESS ("NATION", "NATION_REGIONKEY_FK")       |
|    7  |          INDEX ACCESS ("CUSTOMER", "CUSTOMER_NATIONKEY_FK")     |
=======================================================================

     1  -  TARGET : NATION.N_NAME, COUNT(*)
     2  -  GROUP KEY : NATION.N_NAME
           RECORD COLUMN : COUNT(*)
           READ KEY COLUMN : NATION.N_NAME
           READ RECORD COLUMN : COUNT(*)
     3  -  JOINED COLUMN : NATION.N_NAME
     4  -  JOINED COLUMN : NATION.N_NATIONKEY, NATION.N_NAME
     5  -  CLONED
           READ COLUMN : REGION.R_REGIONKEY, REGION.R_NAME
             PHYSICAL FILTER : REGION.R_NAME = 'ASIA'
     6  -  CLONED
           READ INDEX COLUMN : NATION.N_REGIONKEY
           READ TABLE COLUMN : NATION.N_NATIONKEY, NATION.N_NAME
             MIN RANGE : NATION.N_REGIONKEY = {REGION.R_REGIONKEY}
             MAX RANGE : NATION.N_REGIONKEY = {REGION.R_REGIONKEY}
     7  -  CLONED
           READ INDEX COLUMN : CUSTOMER.C_NATIONKEY
             MIN RANGE : CUSTOMER.C_NATIONKEY = {NATION.N_NATIONKEY}
             MAX RANGE : CUSTOMER.C_NATIONKEY = {NATION.N_NATIONKEY}
<<<  end print plan
```

A view does not exist in the execution plan above. The view is merged to an outer query, then it is perfor med as a converted query form form as follows.

Figure 9 Simple view merging



Outer Join Table Elimination

It eliminates an unnecessary outer join table.

Neither an outer join nor access to the right table is required in the following cases.

- It is a left outer join, and
- a predicate in a form of *left_table.col = right_table.col* exists in ON clause.
- a unique index for right_table.col of ON clause exists.
- a column in a right table is not used in any other clauses except for ON clause condition.

The following is an example of eliminating an outer join table.

The access to nation table exists only in ON clause, and n_nationkey is a primary key column in the followi ng example, so it is unique and even null data does not exist. Therefore, eliminating nation table does no t affect the result.

```
\EXPLAIN PLAN
SELECT COUNT(*)
  FROM supplier
       LEFT OUTER JOIN
       nation
       ON s_nationkey = n_nationkey
 WHERE s_acctbal < 0
;
COUNT(*)
--------
     886
1 row selected.
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                                              | ROWS |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                              |    1 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                   |    1 |
|    2  |      TABLE ACCESS ("SUPPLIER")                                 |    1 |
================================================================================
     1  -  TARGET : COUNT(*)
     2  -  CLONED
           READ COLUMN : SUPPLIER.S_ACCTBAL
           AGGREGATION : COUNT(*)
```

```
            PHYSICAL FILTER : SUPPLIER.S_ACCTBAL < 0
<<<  end print plan
```

All accesses to OUTER JOIN and right table are eliminated in the execution plan above.

Outer Join Operation Elimination

It eliminates an unnecessary outer join operation.

- For a left outer join,
  - If a conditional clause corresponding to the right table exists in where clause, and the conditional clause is not IS NULL,
    - It can be changed to an inner join
- For a full outer join,
  - If a conditional clause corresponding to the right table exists in where clause, and the conditional clause is not IS NULL,
    - It can be changed to a right outer join
  - If a conditional clause corresponding to the left table exists in where clause, and the conditional c lause is not IS NULL,
    - It can be changed to a left outer join.
  - If a conditional clause corresponding to both left and right tables exists in where clause, and the conditional clause is not IS NULL,
    - It can be changed to an inner join

The following is an example of eliminating an outer join operation.

```
\EXPLAIN PLAN
  SELECT MAX(COUNT(*))
    FROM customer
         LEFT OUTER JOIN
         orders
         ON  c_custkey = o_custkey
         AND o_comment LIKE '%special%requests%'
   WHERE o_orderpriority = '1-URGENT'
GROUP BY c_custkey;
MAX(COUNT(*))
-------------
            3
1 row selected.
>>>  start print plan
< Execution Plan >
```

```
================================================================================
|  IDX  |  NODE DESCRIPTION                                                    |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                    |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                         |
|    2  |      AGGREGATION BY HASH                                             |
|    3  |        GROUP                                                         |
|    4  |          HASH JOIN (INNER JOIN)                                      |
|    5  |            INDEX ACCESS ("CUSTOMER", "CUSTOMER_PK_INDEX")           |
|    6  |            HASH JOIN INSTANT                                         |
|    7  |              TABLE ACCESS ("ORDERS")                                 |
================================================================================
     1  -  TARGET : MAX( COUNT(*) )
     2  -  AGGREGATION : MAX( COUNT(*) )
     3  -  GROUP KEY : CUSTOMER.C_CUSTKEY
           RECORD COLUMN : COUNT(*)
     4  -  JOINED COLUMN : CUSTOMER.C_CUSTKEY
     5  -  READ INDEX COLUMN : CUSTOMER.C_CUSTKEY
     6  -  HASH KEY : ORDERS.O_CUSTKEY
           READ KEY COLUMN : ORDERS.O_CUSTKEY
             HASH FILTER : ORDERS.O_CUSTKEY = CUSTOMER.C_CUSTKEY
     7  -  READ COLUMN : ORDERS.O_CUSTKEY, ORDERS.O_ORDERPRIORITY,
                         ORDERS.O_COMMENT
             PHYSICAL FILTER : ORDERS.O_ORDERPRIORITY = '1-URGENT'
             LOGICAL FILTER : ORDERS.O_COMMENT LIKE '%special%requests%'
<<<  end print plan
```

o_orderpriority = '1-URGENT condition exists in WHERE clause in the example above. Therefore, rows all of whose data is NULL can not exist as a result of the right table due to this condition.

Therefore, the results are same even when converting a left outer join to an inner join.

EXISTS/NOT EXIST Operation Target Optimization

It decreases unnecessary expressions from SELECT list of a subquery which exists in EXISTS or NOT EXISTS, so that it improves the query processing performance.

EXISTS or NOT EXISTS operation is a operator which determines whether the result row of a subquery exi sts, so neither the number of expressions in SELECT list of a subquery nor does the processing result affec t the operation result. Therefore, it changes SELECT list of the subquery to TRUE (BOOLEAN constant).

The following is an example of optimizing EXISTS operation target.

```
\EXPLAIN PLAN
SELECT o_orderpriority,
       count(*) as order_count
  FROM orders
 WHERE o_orderdate = date '1993-07-01'
   AND EXISTS (
               SELECT /*+ NO_UNNEST */
                      *
                 FROM lineitem
                WHERE l_orderkey = o_orderkey
                  AND l_commitdate < l_receiptdate
             )
GROUP BY o_orderpriority
ORDER BY o_orderpriority;
O_ORDERPRIORITY ORDER_COUNT
--------------- -----------
1-URGENT                113
2-HIGH                  136
3-MEDIUM                112
4-NOT SPECIFIED         103
5-LOW                    97
5 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
¦IDX¦  NODE DESCRIPTION                                                        ¦
--------------------------------------------------------------------------------
¦ 0 ¦  SELECT STATEMENT                                                        ¦
¦ 1 ¦    QUERY BLOCK ("$QB_IDX_2")                                             ¦
¦ 2 ¦      SORT INSTANT                                                        ¦
¦ 3 ¦        GROUP HASH INSTANT                                                ¦
¦ 4 ¦          TABLE ACCESS ("ORDERS")                                         ¦
¦ 5 ¦          SUB QUERY LIST                                                  ¦
¦ 6 ¦            INLINE_VIEW ("$V6")                                           ¦
¦ 7 ¦              QUERY BLOCK ("$QB_IDX_6")                                   ¦
¦ 8 ¦                INDEX ACCESS ("LINEITEM", "LINEITEM_ORDERKEY_FK")         ¦
================================================================================
     1  -  TARGET : ORDERS.O_ORDERPRIORITY, COUNT(*) AS ORDER_COUNT
     2  -  SORT KEY : "ORDERS.O_ORDERPRIORITY ASC NULLS LAST"
```

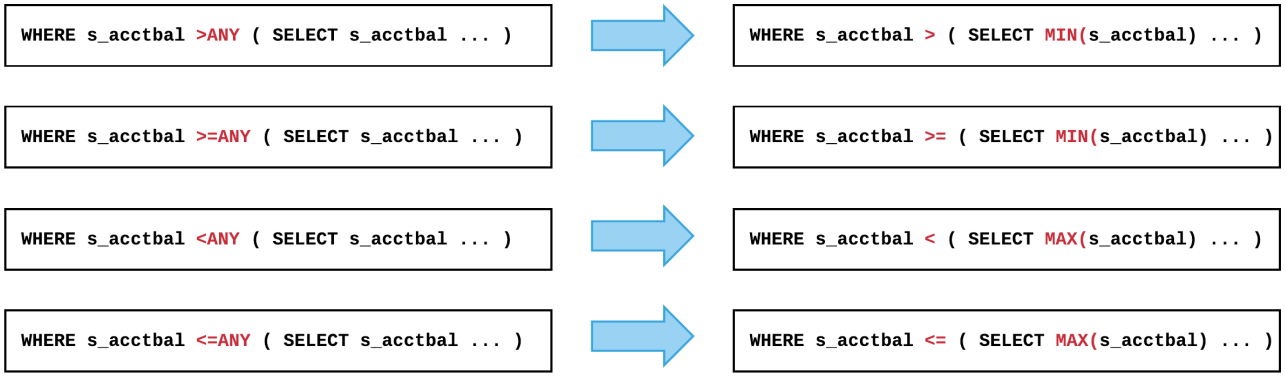
```
            RECORD COLUMN : COUNT(*)
            READ KEY COLUMN : ORDERS.O_ORDERPRIORITY
            READ RECORD COLUMN : COUNT(*)
      3   -  GROUP KEY : ORDERS.O_ORDERPRIORITY
            RECORD COLUMN : COUNT(*)
            READ KEY COLUMN : ORDERS.O_ORDERPRIORITY
            READ RECORD COLUMN : COUNT(*)
      4   -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_ORDERDATE,
                           ORDERS.O_ORDERPRIORITY
              PHYSICAL FILTER : ORDERS.O_ORDERDATE = DATE'1993-07-01'
              POST FILTER : EXISTS( ( $V6.DUMMY_COL ) )
      6   -  COLUMN : $V6.DUMMY_COL AS DUMMY_COL
      7   -  TARGET : NOTHING
      8   -  READ INDEX COLUMN : LINEITEM.L_ORDERKEY
            READ TABLE COLUMN : LINEITEM.L_COMMITDATE, LINEITEM.L_RECEIPTDATE
              MIN RANGE : LINEITEM.L_ORDERKEY = {ORDERS.O_ORDERKEY}
              MAX RANGE : LINEITEM.L_ORDERKEY = {ORDERS.O_ORDERKEY}
              PHYSICAL TABLE FILTER : LINEITEM.L_RECEIPTDATE > LINEITEM.L_COMMITDATE
<<<  end print plan
```

lineitem is a table with 16 columns in the example above. It is specified to read all columns in lineitem by using * in SELECT list within EXISTS subquery of a user query. However, it fetches only the information ab out whether the row satisfying the condition exists, but target column value does not fetch anything at t he time of execution.

Quantifier Elimination

It alters SQL as follows to eliminate ANY quantifier.

Figure 10 Quantifier elimination



The following is an example of eliminating a quantifier.

```
\EXPLAIN PLAN
SELECT COUNT(*)
  FROM supplier
 WHERE s_acctbal >ANY ( SELECT s_acctbal
                          FROM supplier, nation
                         WHERE s_nationkey = n_nationkey
                           AND n_name = 'CHINA' )
;
COUNT(*)
--------
    9950
1 row selected.
>>>  start print plan
< Execution Plan >
============================================================================
|  IDX  |  NODE DESCRIPTION                                                |
----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                     |
|    2  |      TABLE ACCESS ("SUPPLIER")                                   |
|    3  |  SUB QUERY LIST                                                  |
|    4  |    INLINE_VIEW ("$V4")                                           |
|    5  |      QUERY BLOCK ("$QB_IDX_6")                                   |
|    6  |        AGGREGATION BY HASH                                       |
|    7  |          NESTED JOIN (INNER JOIN)                                |
|    8  |            TABLE ACCESS ("NATION")                               |
|    9  |            INDEX ACCESS ("SUPPLIER", "SUPPLIER_NATIONKEY_FK")   |
============================================================================
     1  -  TARGET : COUNT(*)
     2  -  READ COLUMN : SUPPLIER.S_ACCTBAL
           AGGREGATION : COUNT(*)
             PHYSICAL FILTER : SUPPLIER.S_ACCTBAL > $V4.$C0
     4  -  COLUMN : MIN( SUPPLIER.S_ACCTBAL ) AS $C0
     5  -  TARGET : MIN( SUPPLIER.S_ACCTBAL )
     6  -  AGGREGATION : MIN( SUPPLIER.S_ACCTBAL )
     7  -  JOINED COLUMN : SUPPLIER.S_ACCTBAL
     8  -  READ COLUMN : NATION.N_NATIONKEY, NATION.N_NAME
             PHYSICAL FILTER : NATION.N_NAME = 'CHINA'
     9  -  READ INDEX COLUMN : SUPPLIER.S_NATIONKEY
```

```
            READ TABLE COLUMN : SUPPLIER.S_ACCTBAL
              MIN RANGE : SUPPLIER.S_NATIONKEY = {NATION.N_NATIONKEY}
              MAX RANGE : SUPPLIER.S_NATIONKEY = {NATION.N_NATIONKEY}
<<<  end print plan
```

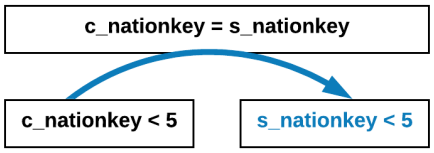
Transitive Closure

It creates a constant condition in another table by using a join condition. In this way, the throughput of jo in is decreased and the performance is improved.

The following is an example of a transitive closure.

Figure 11 Transitive closure

```
\explain plan
SELECT COUNT(*)
  FROM customer, supplier
 WHERE c_nationkey = s_nationkey
   AND c_nationkey < 5
;

COUNT(*)
--------
12305943

1 row selected.

>>>  start print plan

< Execution Plan >
=================================================================================================
|  IDX  |  NODE DESCRIPTION                                                    |              ROWS |
-------------------------------------------------------------------------------------------------
|     0  |  SELECT STATEMENT                                                   |                 1 |
|     1  |    QUERY BLOCK ("$QB_IDX_2")                                        |                 1 |
|     2  |      AGGREGATION BY HASH                                            |                 1 |
|     3  |        MERGE JOIN (INNER JOIN)                                      |          12305943 |
|     4  |          INDEX ACCESS ("CUSTOMER", "CUSTOMER_NATIONKEY_FK")         | (     29914)      29914 |
|     5  |          INDEX ACCESS ("SUPPLIER", "SUPPLIER_NATIONKEY_FK")         | (      2057)       2057 |
=================================================================================================
     1  -  TARGET : COUNT(*)
     2  -  AGGREGATION : COUNT(*)
     3  -  JOINED COLUMN : NOTHING
             ON FILTER (Equi) : CUSTOMER.C_NATIONKEY = SUPPLIER.S_NATIONKEY
     4  -  CLONED
           READ INDEX COLUMN : CUSTOMER.C_NATIONKEY
             MAX RANGE : CUSTOMER.C_NATIONKEY < 5
     5  -  CLONED
           READ INDEX COLUMN : SUPPLIER.S_NATIONKEY
             MAX RANGE : SUPPLIER.S_NATIONKEY < 5

<<<  end print plan
```



Join Transitive Closure

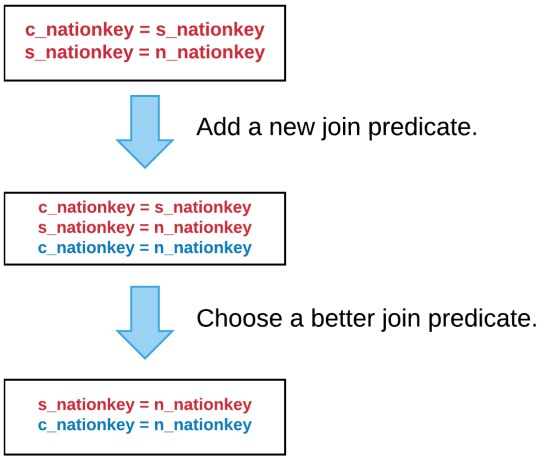
It creates a join condition in another table by using a join condition. Various join orderings and join meth ods are available, so a better execution plan can be made.

It is performed in a way of adding a join condition (A=C) to the other join condition (A = B AND B = C).

The following is an example of a join transitive closure.

Figure 12 Join transitive closure

```
\explain plan
select
    n_name,
    ROUND( sum(l_extendedprice * (1 - l_discount)), 2) as revenue
from
    customer,
    orders,
    lineitem,
    supplier,
    nation,
    region
where
      c_custkey = o_custkey
  and l_orderkey = o_orderkey
  and l_suppkey = s_suppkey
  and c_nationkey = s_nationkey
  and s_nationkey = n_nationkey
  and n_regionkey = r_regionkey
  and r_name = 'ASIA'
  and o_orderdate >= date '1994-01-01'
  and o_orderdate < date '1994-01-01' + interval '1' year
group by
    n_name
order by
    revenue desc;

N_NAME                         REVENUE
------------------------- -----------
INDONESIA                  55502041.17
VIETNAM                    55295087.00
CHINA                      53724494.26
INDIA                      52035512.00
JAPAN                      45410175.70

5 rows selected.
```




```
>>>  start print plan
< Execution Plan >
=============================================================================================
|  IDX  |  NODE DESCRIPTION                                                                 |
---------------------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                                      |
|    2  |      SORT INSTANT                                                                 |
|    3  |        SINGLE CLUSTER                                                             |
|    4  |          SELECT STATEMENT                                                         |
|    5  |            QUERY BLOCK ("$QB_IDX_2")                                              |
|    6  |              GROUP HASH INSTANT                                                   |
|    7  |                HASH JOIN (INNER JOIN)                                             |
|    8  |                  NESTED JOIN (INNER JOIN)                                         |
|    9  |                    NESTED JOIN (INNER JOIN)                                       |
|   10  |                      NESTED JOIN (INNER JOIN)                                     |
|   11  |                        NESTED JOIN (INNER JOIN)                                   |
|   12  |                          TABLE ACCESS ("REGION" AS _A6)                           |
|   13  |                          INDEX ACCESS ("NATION" AS _A5, "NATION_REGIONKEY_FK")    |
|   14  |                        INDEX ACCESS ("CUSTOMER" AS _A4, "CUSTOMER_NATIONKEY_FK")  |
|   15  |                      INDEX ACCESS ("ORDERS" AS _A3, "ORDERS_CUSTKEY_FK")          |
|   16  |                    INDEX ACCESS ("LINEITEM" AS _A2, "LINEITEM_ORDERKEY_FK")       |
|   17  |                  HASH JOIN INSTANT                                                |
|   18  |                    TABLE ACCESS ("SUPPLIER" AS _A1)                               |
=============================================================================================
     1  -  TARGET : NATION.N_NAME, ROUND(SUM(LINEITEM.L_EXTENDEDPRICE * (1 - LINEITEM.L_DISCOUNT)),2) AS REVENUE
     ...
     ...
     ...

    14  -  CLONED
           READ INDEX COLUMN : _A4.C_NATIONKEY
           READ TABLE COLUMN : _A4.C_CUSTKEY
             MIN RANGE : _A4.C_NATIONKEY = {_A5.N_NATIONKEY}
             MAX RANGE : _A4.C_NATIONKEY = {_A5.N_NATIONKEY}
    ...
    ...
    ...

    17  -  HASH KEY : _A1.S_SUPPKEY, _A1.S_NATIONKEY
           READ KEY COLUMN : _A1.S_SUPPKEY, _A1.S_NATIONKEY
             HASH FILTER : _A1.S_SUPPKEY = _A2.L_SUPPKEY AND _A1.S_NATIONKEY = _A5.N_NATIONKEY
           FETCH ONE ROW
    18  -  CLONED
           READ COLUMN : _A1.S_SUPPKEY, _A1.S_NATIONKEY
<<<  end print plan
```

Subquery Unnesting

A subquery unnesting converts a subquery in a conditional clause into a join statement guaranteeing the same result. It enables to select various access path, join method, join order, so a better execution plan ca n be made.

Subqueries are classified into the following two types.

- Nested subquery (Regular non-scalar subquery)
  - EXISTS/NOT EXIST subquery
  - Comparison operator (=, >,>=, <, <=, <>) ANY subquery
  - Comparison operator (=, >,>=, <, <=, <>) ALL subquery
  - IN/NOT IN subquery
- Scalar subquery: It is used in WHERE clause or a SELECT list, and returns only a single result.

Not all subqueries are unnested. The subquery can be unnested only when it satisfies the following constr aints.

- It should not include set operator.
- A scalar subquery is available only when it is specified on WHERE clause.
- It should include a correlated predicate.

A correlated predicate is a predicate including a column of an undefined outer query block in a subquery. In the example below, *c.cust_id* is a correlated column and *s.cust_id = c.cust_id* is a correlated predicate.

```
SELECT C.cust_last_name, C.country_id
FROM   customers C
WHERE  EXISTS (SELECT 1
                 FROM sales S
                WHERE S.quantity_sold > 1000
                  AND S.cust_id = C.cust_id);
```

Nested Subquery Unnesting

It converts a subquery into a semi join, an anti-join, or an inner join.

Figure 13 Nested subquery unnesting

```
SELECT COUNT(*)
  FROM customer
 WHERE c_custkey IN ( SELECT o_custkey
                        FROM orders
                       WHERE o_orderdate = DATE'1997-02-01'
                    )
```

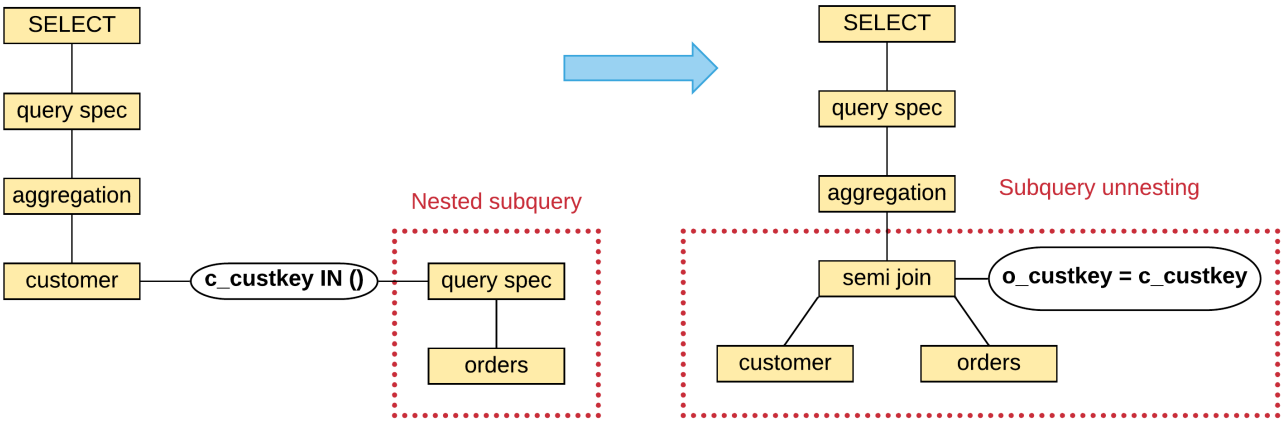


Figure 14 Nested subquery unnesting plan

```
>>>  start print plan
< Execution Plan >
=================================================================================
|  IDX  |  NODE DESCRIPTION                                                      |
---------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                      |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                           |
|    2  |      AGGREGATION BY HASH                                               |
|    3  |        MULTIPLE CLUSTER                                                |
|    4  |          SELECT STATEMENT                                              |
|    5  |            QUERY BLOCK ("$QB_IDX_2")                                   |
|    6  |              SORT INSTANT                                              |
|    7  |                NESTED JOIN (INVERTED SEMI)                             |
|    8  |                  SORT JOIN INSTANT (UNIQUE)                            |
|    9  |                    TABLE ACCESS ("ORDERS" AS _A2)                      |
|   10  |                  INDEX ACCESS ("CUSTOMER" AS _A1, "CUSTOMER_PK_INDEX")  |
=================================================================================
     1  -  TARGET : COUNT(*)
     2  -  AGGREGATION : COUNT(*)
     ...
     ...
     ...

    10  -  CLONED
            READ INDEX COLUMN : _A1.C_CUSTKEY
              MIN RANGE : _A1.C_CUSTKEY = {_A2.O_CUSTKEY}
              MAX RANGE : _A1.C_CUSTKEY = {_A2.O_CUSTKEY}
            FETCH ONE ROW
<<<  end print plan
```

Scalar Subquery Unnesting

It can unnest only a scalar subquery in WHERE clause, and the following conditions should be satisfied.

- It should be a single row aggregation

- It should include a correlated predicate.

The following is an example of unnesting a scalar subquery.

Figure 15 Scalar subquery unnesting

```
\explain plan
SELECT ps_suppkey
  FROM partsupp,
       part
 WHERE ps_partkey = p_partkey
   AND p_name like 'forest%'
   AND ps_availqty > ( SELECT 0.5 * sum(l_quantity)
                         FROM lineitem
                        WHERE l_partkey = ps_partkey
                          AND l_suppkey = ps_suppkey
                          AND l_shipdate >= date'1994-01-01'
                          AND l_shipdate < date'1994-01-01' + interval '1' year
                     )
;
```

```
\explain plan
SELECT ps_suppkey
  FROM partsupp,
       part,
       ( SELECT l_partkey,
                l_suppkey,
                0.5 * sum(l_quantity) as qty
           FROM lineitem
          WHERE l_shipdate >= date'1994-01-01'
            AND l_shipdate < date'1994-01-01' + interval '1' year
          GROUP BY l_partkey, l_suppkey
       )
 WHERE ps_partkey = p_partkey
   AND p_name like 'forest%'
   AND ps_availqty > v1.qty
   AND l_partkey = ps_partkey
   AND l_suppkey = ps_suppkey
;
```

Figure 16 Scalar subquery unnesting plan

```
>>>  start print plan
< Execution Plan >
=================================================================================
|  IDX  |  NODE DESCRIPTION                                                      |
---------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                      |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                           |
|    2  |      NESTED JOIN (INNER JOIN)                                          |
|    3  |        NESTED JOIN (INNER JOIN)                                        |
|    4  |          TABLE ACCESS ("PART")                                         |
|    5  |          INDEX ACCESS ("PARTSUPP", "PARTSUPP_PARTKEY_FK")              |
|    6  |        INLINE_VIEW                                                     |
|    7  |          QUERY BLOCK ("$QB_IDX_8")                                     |
|    8  |            GROUP                                                       |
|    9  |              INDEX ACCESS ("LINEITEM", "LINEITEM_PARTKEY_SUPPKEY_FK")  |
=================================================================================
     1  -  TARGET : PARTSUPP.PS_SUPPKEY
     2  -  JOINED COLUMN : PARTSUPP.PS_SUPPKEY
     3  -  JOINED COLUMN : PARTSUPP.PS_PARTKEY, PARTSUPP.PS_SUPPKEY, PARTSUPP.PS_AVAILQTY
     4  -  READ COLUMN : PART.P_PARTKEY, PART.P_NAME
             LOGICAL FILTER : PART.P_NAME LIKE 'forest%'
     5  -  READ INDEX COLUMN : PARTSUPP.PS_PARTKEY
           READ TABLE COLUMN : PARTSUPP.PS_SUPPKEY, PARTSUPP.PS_AVAILQTY
             MIN RANGE : PARTSUPP.PS_PARTKEY = {PART.P_PARTKEY}
             MAX RANGE : PARTSUPP.PS_PARTKEY = {PART.P_PARTKEY}
     6  -  COLUMN : LINEITEM.L_PARTKEY AS L_PARTKEY,
                    LINEITEM.L_SUPPKEY AS L_SUPPKEY,
                    0.5 * SUM( LINEITEM.L_QUANTITY ) AS $C2
     7  -  TARGET : LINEITEM.L_PARTKEY,
                    LINEITEM.L_SUPPKEY,
                    0.5 * SUM( LINEITEM.L_QUANTITY )
     8  -  GROUP KEY : LINEITEM.L_PARTKEY, LINEITEM.L_SUPPKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             LOGICAL FILTER : {PARTSUPP.PS_AVAILQTY} > ( 0.5 * SUM(LINEITEM.L_QUANTITY) )
     9  -  READ INDEX COLUMN : LINEITEM.L_PARTKEY, LINEITEM.L_SUPPKEY
           READ TABLE COLUMN : LINEITEM.L_QUANTITY, LINEITEM.L_SHIPDATE
             MIN RANGE : LINEITEM.L_PARTKEY = {PARTSUPP.PS_PARTKEY}
                         AND LINEITEM.L_SUPPKEY = {PARTSUPP.PS_SUPPKEY}
             MAX RANGE : LINEITEM.L_PARTKEY = {PARTSUPP.PS_PARTKEY}
                         AND LINEITEM.L_SUPPKEY = {PARTSUPP.PS_SUPPKEY}
             PHYSICAL TABLE FILTER : LINEITEM.L_SHIPDATE < DATE'1994-01-01' + CAST( '1'
AS INTERVAL(YEAR) ) AND LINEITEM.L_SHIPDATE >= DATE'1994-01-01'
<<<  end print plan

;
```

Complex View Merging

It merges a view including *group by* with a superordinate query block.
A complex view merging is useful when *group by* can not decrease the intermediate results a lot and the join filtering with a superordinate query block is effective. In other words, if a complex view merging is us ed when *group by* can decrease the intermediate results a lot, then it may degrade the performance.

A complex view merging can not be applied in the following cases.

- When a query block within a view includes the following item
  - SET operator
  - ROWNUM
  - LIMIT/OFFSET
  - Single row aggregation

  - ORDER BY
  - SELECT list including a subquery
  - FULL OUTER JOIN
  - NATURAL JOIN
- When a view participates in the following query
  - Join other than an inner join
  - An equi Join predicate does not exist

The following is an example of merging a complex view.

Figure 17 Complex view merging

```
\explain plan
SELECT ps_suppkey
  FROM partsupp,
       part,
       ( SELECT l_partkey,
                l_suppkey,
                0.5 * sum(l_quantity) qty
           FROM lineitem
          WHERE l_shipdate >= date'1994-01-01'
            AND l_shipdate < date'1994-01-01' + interval '1' year
          GROUP BY l_partkey, l_suppkey
       ) v1
 WHERE ps_partkey = p_partkey
   AND p_name like 'forest%'
   AND l_partkey = ps_partkey
   AND l_suppkey = ps_suppkey
   AND ps_availqty > v1.qty
;
```

```
\explain plan
SELECT ps_suppkey
  FROM ( SELECT MAX(ps_suppkey) as ps_suppkey
           FROM lineitem,
                partsupp,
                part
          WHERE ps_partkey = p_partkey
            AND p_name like 'forest%'
            AND l_partkey = ps_partkey
            AND l_suppkey = ps_suppkey
            AND l_shipdate >= date'1994-01-01'
            AND l_shipdate < date'1994-01-01' + interval '1' year
          GROUP BY l_partkey, l_suppkey, partsupp.rowid, part.rowid
          HAVING MAX(ps_availqty) > 0.5 * sum(l_quantity)
       ) v1;
;
```

Figure 18 Complex view merging plan

```
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                                                    |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                    |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                         |
|    2  |      INLINE_VIEW ("V1")                                              |
|    3  |        QUERY BLOCK ("$QB_IDX_9")                                     |
|    4  |          GROUP HASH INSTANT                                          |
|    5  |            NESTED JOIN (INNER JOIN)                                  |
|    6  |              NESTED JOIN (INNER JOIN)                                |
|    7  |                TABLE ACCESS ("PART")                                 |
|    8  |                INDEX ACCESS ("LINEITEM", "LINEITEM_PARTKEY_SUPPKEY_FK")   |
|    9  |              INDEX ACCESS ("PARTSUPP", "PARTSUPP_PK_INDEX")          |
================================================================================
     1  -  TARGET : V1.PS_SUPPKEY
     2  -  COLUMN : PS_SUPPKEY AS PS_SUPPKEY
     3  -  TARGET : MAX( PARTSUPP.PS_SUPPKEY ) AS PS_SUPPKEY
     4  -  GROUP KEY : LINEITEM.L_PARTKEY, LINEITEM.L_SUPPKEY, PARTSUPP.$PHYSICAL_ROWID, PART.$PHYSICAL_ROWID
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY ), MAX( PARTSUPP.PS_SUPPKEY )
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY ), MAX( PARTSUPP.PS_SUPPKEY )
             LOGICAL FILTER : PARTSUPP.PS_AVAILQTY > ( 0.5 * SUM( LINEITEM.L_QUANTITY ) )
     5  -  JOINED COLUMN : LINEITEM.L_PARTKEY, LINEITEM.L_SUPPKEY, PARTSUPP.$PHYSICAL_ROWID, PART.$PHYSICAL_ROWID,
PARTSUPP.PS_AVAILQTY, LINEITEM.L_QUANTITY, PARTSUPP.PS_SUPPKEY
     6  -  JOINED COLUMN : PART.P_PARTKEY, LINEITEM.L_PARTKEY, LINEITEM.L_SUPPKEY, PART.$PHYSICAL_ROWID,
LINEITEM.L_QUANTITY
     7  -  READ COLUMN : PART.P_PARTKEY, PART.P_NAME
             LOGICAL FILTER : PART.P_NAME LIKE 'forest%'
     8  -  READ INDEX COLUMN : LINEITEM.L_PARTKEY, LINEITEM.L_SUPPKEY
           READ TABLE COLUMN : LINEITEM.L_QUANTITY, LINEITEM.L_SHIPDATE
             MIN RANGE : LINEITEM.L_PARTKEY = {PART.P_PARTKEY}
             MAX RANGE : LINEITEM.L_PARTKEY = {PART.P_PARTKEY}
             PHYSICAL TABLE FILTER : LINEITEM.L_SHIPDATE < DATE'1994-01-01' + CAST( '1' AS INTERVAL(YEAR) ) AND
LINEITEM.L_SHIPDATE >= DATE'1994-01-01'
     9  -  READ INDEX COLUMN : PARTSUPP.PS_PARTKEY, PARTSUPP.PS_SUPPKEY
           READ TABLE COLUMN : PARTSUPP.PS_AVAILQTY
             MIN RANGE : PARTSUPP.PS_PARTKEY = {PART.P_PARTKEY} AND
                         PARTSUPP.PS_PARTKEY = {LINEITEM.L_PARTKEY} AND
                         PARTSUPP.PS_SUPPKEY = {LINEITEM.L_SUPPKEY}
             MAX RANGE : PARTSUPP.PS_PARTKEY = {PART.P_PARTKEY} AND
                         PARTSUPP.PS_PARTKEY = {LINEITEM.L_PARTKEY} AND
                         PARTSUPP.PS_SUPPKEY = {LINEITEM.L_SUPPKEY}
           FETCH ONE ROW
<<<  end print plan
```

15.3 Enumerator

An enumerator calculates the cost based on statistics information to find the most efficient plan.

It receives a trans plan and creates various cost plans about it, then calculates the cost per each cost plan based on statistics information to select the plan of the lowest cost.

Access Paths

An access path is a method to access a single table, and there are following four types.

- Table access
- Index access
- Rowid access
- Index concat

It calculates the cost according to the methods above, and select an access of the lowest cost as an execu tion plan.

Table Access

A table access reads all rows stored in a table in a way as they were stored.

A table access is selected in the following cases.

- When an index does not exist
- When a predicate for using an index does not exist even though an index exists (e.g. WHERE col1 + 1 = 10)
- When the table accessing cost is big because a condition for the first key column of an index does no t exist
  (e.g. WHERE col2 > 3 when only a condition for the second column of composite index (col1, col2) e xists)
- When a table access cost is smaller than an index access cost because the table data is small
- When a user gives a table access hint (e.g. FULL(t1))
- When an index selectivity is poor or the data is extremely disproportionately distributed

The following is an example of using an table access.

```
\EXPLAIN PLAN
SELECT r_regionkey, r_name
```

```
  FROM region;
R_REGIONKEY R_NAME
----------- -------------------------
          0 AFRICA
          1 AMERICA
          2 ASIA
          3 EUROPE
          4 MIDDLE EAST
5 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                     ¦
--------------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                     ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                        ¦
¦    2  ¦      TABLE ACCESS ("REGION")                                        ¦
================================================================================
     1  -  TARGET : REGION.R_REGIONKEY, REGION.R_NAME
     2  -  READ COLUMN : REGION.R_REGIONKEY, REGION.R_NAME
<<<  end print plan
```

Index Access

An index access uses an index to read a table.

Index access types are an index full scan, an index unique scan, an index range scan and an in key range s can, and the best type is selected by the cost estimation.

Index Full Scan

It fully scans the index.

The following is an example of an index full scan.

```
\EXPLAIN PLAN
SELECT p_partkey
  FROM part;
...
200000 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
```

```
¦  IDX  ¦  NODE DESCRIPTION                                          ¦
-------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                          ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                             ¦
¦    2  ¦      INDEX ACCESS ("PART", "PART_PK_INDEX")              ¦
=========================================================================
     1  -  TARGET : PART.P_PARTKEY
     2  -  READ INDEX COLUMN : PART.P_PARTKEY
<<<  end print plan
```

It retrieves p_partkey which is a key column of PART_PK_INDEX only, so making the result by the index fu ll scan costs less than the table full scan.

Index Unique Scan

It fetches only one row through the index.

The following is an example of an index unique scan.

```
\EXPLAIN PLAN
SELECT *
  FROM part
 WHERE p_partkey = 1;
...
1 row selected.
>>>  start print plan
< Execution Plan >
=========================================================================
¦  IDX  ¦  NODE DESCRIPTION                                          ¦
-------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                          ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                             ¦
¦    2  ¦      INDEX ACCESS ("PART", "PART_PK_INDEX")              ¦
=========================================================================
     1  -  TARGET : PART.P_PARTKEY, PART.P_NAME, PART.P_MFGR, PART.P_BRAND,
                    PART.P_TYPE, PART.P_SIZE, PART.P_CONTAINER,
                    PART.P_RETAILPRICE, PART.P_COMMENT
     2  -  READ INDEX COLUMN : PART.P_PARTKEY
           READ TABLE COLUMN : PART.P_NAME, PART.P_MFGR, PART.P_BRAND,
                               PART.P_TYPE, PART.P_SIZE, PART.P_CONTAINER,
                               PART.P_RETAILPRICE, PART.P_COMMENT
             MIN RANGE : PART.P_PARTKEY = 1
```

```
            MAX RANGE : PART.P_PARTKEY = 1
          FETCH ONE ROW
<<<  end print plan
```

Index Range Scan

It reads rows in the range satisfying a predicate condition through an index. The result rows are sorted fo r an index key column because they are read through the index.

The following is an example of an index range scan.

```
\EXPLAIN PLAN
SELECT p_partkey, p_brand, p_type
  FROM part
 WHERE p_partkey >= 10
   AND p_partkey < 20;
P_PARTKEY P_BRAND    P_TYPE
--------- ---------- -------------------------
       10 Brand#54   LARGE BURNISHED STEEL
       11 Brand#25   STANDARD BURNISHED NICKEL
       12 Brand#33   MEDIUM ANODIZED STEEL
       13 Brand#55   MEDIUM BURNISHED NICKEL
       14 Brand#13   SMALL POLISHED STEEL
       15 Brand#15   LARGE ANODIZED BRASS
       16 Brand#32   PROMO PLATED TIN
       17 Brand#43   ECONOMY BRUSHED STEEL
       18 Brand#11   SMALL BURNISHED STEEL
       19 Brand#23   SMALL ANODIZED NICKEL
10 rows selected.
>>>  start print plan
< Execution Plan >
------------------------------------------------------------------------
|  IDX  |  NODE DESCRIPTION                                            |
------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                            |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                 |
|    2  |      INDEX ACCESS ("PART", "PART_PK_INDEX")                  |
------------------------------------------------------------------------
     1  -  TARGET : PART.P_PARTKEY, PART.P_BRAND, PART.P_TYPE
     2  -  READ INDEX COLUMN : PART.P_PARTKEY
           READ TABLE COLUMN : PART.P_BRAND, PART.P_TYPE
             MIN RANGE : PART.P_PARTKEY >= 10
```

```
            MAX RANGE : PART.P_PARTKEY IS NOT NULL AND PART.P_PARTKEY < 20
<<<  end print plan
```

In Key Range Scan

An in key range scan is available when the following predicate exists.

```
( col1, col2 ) IN ( (val1, val2), (val3, val4) )
```

- IN or =ANY list function filter exists in WHERE clause.
- col1 and col2 should be base columns. (It should be a column without an operation or a function.)
- (val1, val3) corresponding to col1 should be convertible to a single data type.
- (val2, val4) corresponding to col2 should be convertible to a single data type.

The following is an example of an in key range scan.

```
\EXPLAIN PLAN
  SELECT o_orderstatus
    FROM orders
   WHERE o_custkey IN ( 1, 10, 100, 1000, 10000 );
...
91 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                      ¦
--------------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                     ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                        ¦
¦    2  ¦      INDEX ACCESS ("ORDERS", "ORDERS_CUSTKEY_FK")                   ¦
================================================================================
     1  -  TARGET : ORDERS.O_ORDERSTATUS
     2  -  READ INDEX COLUMN : ORDERS.O_CUSTKEY
           READ TABLE COLUMN : ORDERS.O_ORDERSTATUS
           IN KEY RANGE
             MIN RANGE : ORDERS.O_CUSTKEY = ?
             MAX RANGE : ORDERS.O_CUSTKEY = ?
<<<  end print plan
```

Rowid Access

A rowid access directly accesses to the corresponding page by using a rowid.

A predicate for the rowid should exist when using the rowid access. Generally, a rowid access is faster th an other access paths, so an estimator most likely to select a rowid access as the best access path as long as a predicate for a rowid exists.

```
gSQL> \EXPLAIN PLAN
SELECT p_brand, p_type
  FROM part
 WHERE rowid = 'AAAAAAAAYe8AACAAAEMJAAA';
P_BRAND    P_TYPE
---------- ------------------------
Brand#33   STANDARD POLISHED COPPER
1 row selected.
>>>  start print plan
< Execution Plan >
================================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                     ¦
--------------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                     ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                        ¦
¦    2  ¦      ROWID ACCESS ("PART")                                          ¦
================================================================================
     1  -  TARGET : PART.P_BRAND, PART.P_TYPE
     2  -  READ COLUMN : PART.P_BRAND, PART.P_TYPE
             ROWID FILTER : PART.ROWID = 'AAAAAAAAYe8AACAAAEMJAAA'
<<<  end print plan
```

Index Concat

An index concat intergrates results of index access, and makes it into a single result. Therefore, it can be used only when OR predicate exists and each predicate can use an index access.

If OR predicate exists, an estimator calculates the cost of index concat, and selects the method when the cost is smaller than that of other access path.

The following is an example of using an index concat.

```
gSQL> \EXPLAIN PLAN
SELECT p_brand, p_type
  FROM part
```

```
 WHERE p_partkey = 1 OR p_partkey = 20;
P_BRAND    P_TYPE
---------- ----------------------
Brand#13   PROMO BURNISHED COPPER
Brand#12   LARGE POLISHED NICKEL
2 rows selected.
>>>  start print plan
< Execution Plan >
============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
|    2  |      CONCAT (Compare Nothing)                                     |
|    3  |        INDEX ACCESS ("PART", "PART_PK_INDEX")                     |
|    4  |        INDEX ACCESS ("PART", "PART_PK_INDEX")                     |
============================================================================
     1  -  TARGET : PART.P_BRAND, PART.P_TYPE
     2  -  CONCAT COLUMN : PART.P_BRAND, PART.P_TYPE
     3  -  READ INDEX COLUMN : PART.P_PARTKEY
           READ TABLE COLUMN : PART.P_BRAND, PART.P_TYPE
             MIN RANGE : PART.P_PARTKEY = 1
             MAX RANGE : PART.P_PARTKEY = 1
           FETCH ONE ROW
     4  -  READ INDEX COLUMN : PART.P_PARTKEY
           READ TABLE COLUMN : PART.P_BRAND, PART.P_TYPE
             MIN RANGE : PART.P_PARTKEY = 20
             MAX RANGE : PART.P_PARTKEY = 20
           FETCH ONE ROW
<<<  end print plan
```

Join

Join combines two or more tables to make them into a single result set.

In this case, a join condition defines the relation between tables. If a join condition does not exist, then m ultiplying rows of every table compose a new result set.

An estimator creates various cost plans considering join order, join method, and access path according to the join type, and calculates a cost, then selects the best cost plan. **Access Paths** are described in a chapte

r above and this chapter describes a join type, a join method and a join order.

Join Type

Cross Join

A join condition does not exist. Therefore, a multiplication set of two tables composes a new result of joi n.

The following is an example of a cross join.

```
gSQL> \EXPLAIN PLAN SELECT r_regionkey, n_nationkey FROM region, nation;
R_REGIONKEY N_NATIONKEY
----------- -----------
          0           0
          0           1
          0           2
          0           3
          0           4
          0           5
          0           6
          0           7
          0           8
          0           9
          0          10
          0          11
          0          12
          0          13
          0          14
          0          15
          0          16
          0          17
          0          18
          0          19
R_REGIONKEY N_NATIONKEY
----------- -----------
          0          20
          0          21
          0          22
          0          23
          0          24
          1           0
```

```
          1           1
         ...         ...
          4          24
125 rows selected.
>>>  start print plan
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
|    2  |      NESTED JOIN (INNER JOIN)                                     |
|    3  |        INDEX ACCESS ("REGION", "REGION_PK_INDEX")                 |
|    4  |        INDEX ACCESS ("NATION", "NATION_PK_INDEX")                 |
=============================================================================
     1  -  TARGET : REGION.R_REGIONKEY, NATION.N_NATIONKEY
     2  -  JOINED COLUMN : REGION.R_REGIONKEY, NATION.N_NATIONKEY
     3  -  READ INDEX COLUMN : REGION.R_REGIONKEY
     4  -  READ INDEX COLUMN : NATION.N_NATIONKEY
<<<  end print plan
```

Inner Join

Only the rows satisfying a join condition from a multiplication set of two tables configure a result set.

The following is an example of an inner join.

```
gSQL> \EXPLAIN PLAN
SELECT r_regionkey, n_nationkey
  FROM region, nation
 WHERE r_regionkey = n_nationkey;
R_REGIONKEY N_NATIONKEY
----------- -----------
          0           0
          1           1
          2           2
          3           3
          4           4
5 rows selected.
>>>  start print plan
< Execution Plan >
=============================================================================
```

```
¦  IDX  ¦  NODE DESCRIPTION                                                ¦
------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                   ¦
¦    2  ¦      NESTED JOIN (INNER JOIN)                                  ¦
¦    3  ¦        INDEX ACCESS ("REGION", "REGION_PK_INDEX")              ¦
¦    4  ¦        INDEX ACCESS ("NATION", "NATION_PK_INDEX")              ¦
========================================================================
     1  -  TARGET : REGION.R_REGIONKEY, NATION.N_NATIONKEY
     2  -  JOINED COLUMN : REGION.R_REGIONKEY, NATION.N_NATIONKEY
     3  -  READ INDEX COLUMN : REGION.R_REGIONKEY
     4  -  READ INDEX COLUMN : NATION.N_NATIONKEY
             MIN RANGE : NATION.N_NATIONKEY = {REGION.R_REGIONKEY}
             MAX RANGE : NATION.N_NATIONKEY = {REGION.R_REGIONKEY}
           FETCH ONE ROW
<<<  end print plan
```

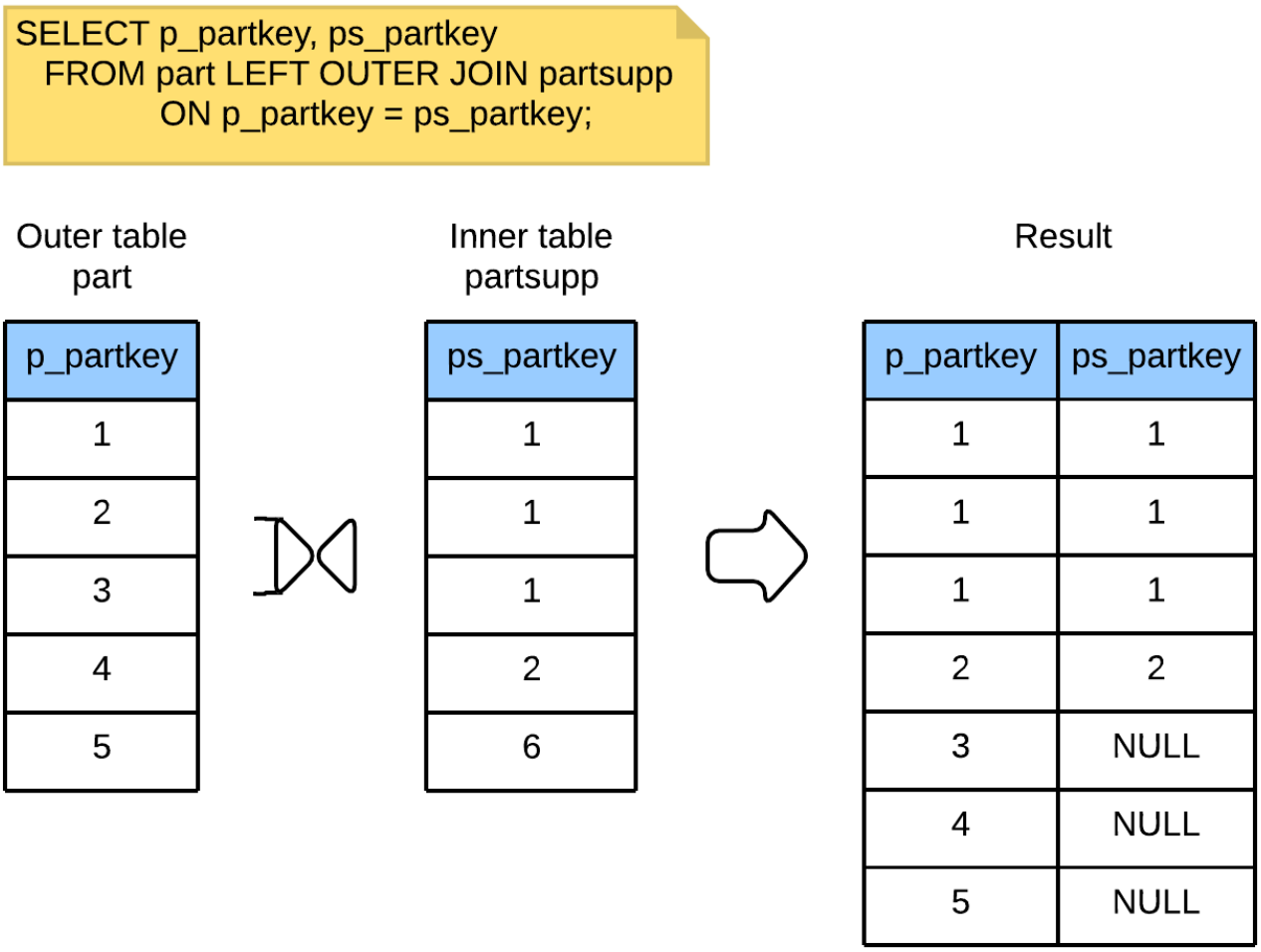
Outer Join

It returns rows satisfying a join condition from a multiplication set of two tables as result, and it also retur ns rows in an outer table as result even though the rows do not satisfy the join condition. In other words, an outer join is used to output even the rows which do not satisfy the join condition as well.

In this case, values corresponding to an inner table are NULL padded.

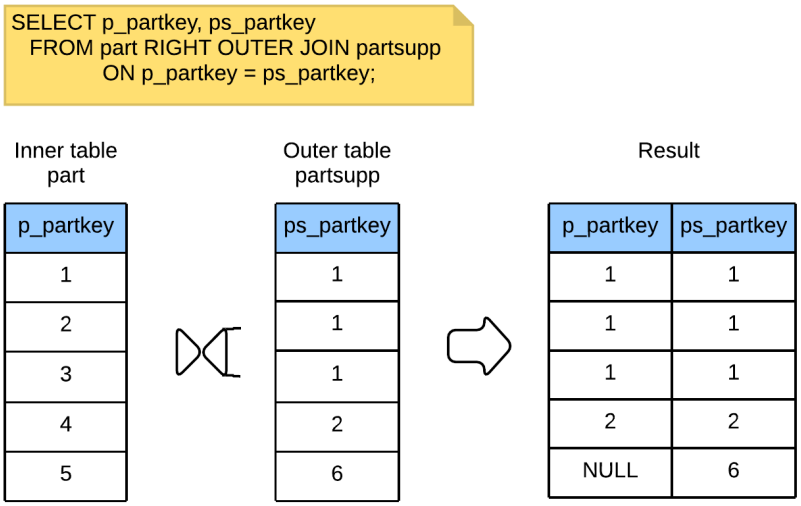
The left table becomes an outer table in a left outer join. Therefore, *part* which is a left table becomes an outer table so it outputs even the rows which do not satisfy the join condition as well in the example belo w. In this case, *partsupp* value which is an inner table is NULL padded.

Figure 19 Left outer join



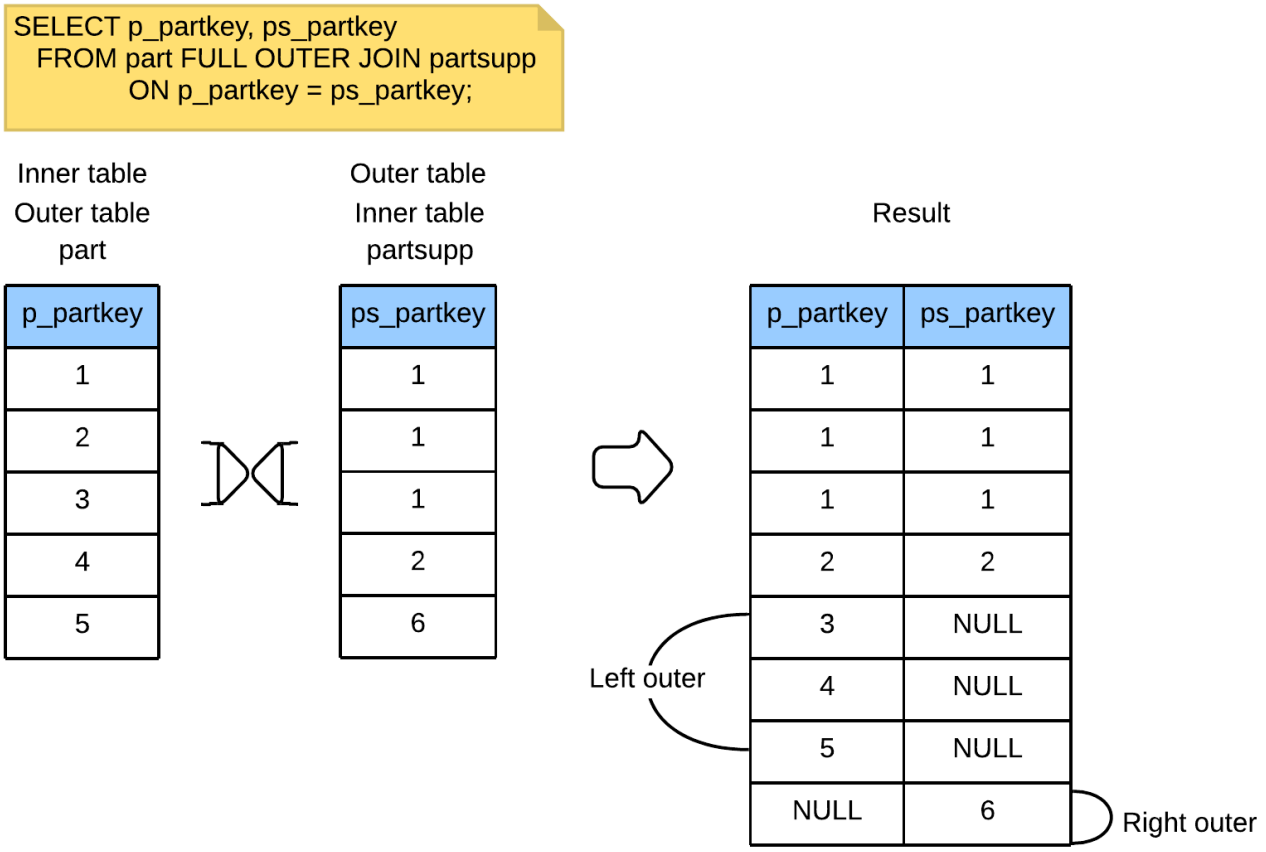
The right table becomes an outer table in a right outer join.
Therefore, *partsupp* which is a right table becomes an outer table so it outputs even the rows which do n ot satisfy the join condition as well in the example below. In this case, *parts* value which is an inner table i s NULL padded.

Figure 20 Right outer join



A full outer join join outputs rows satisfying the join condition, then it performs left outer join and a right outer join so that it outputs all rows.

Figure 21 Full outer join



Left Outer Join

It returns all rows satisfying a join condition as result, and also returns rows in a left table as result even th ough the rows do not satisfy the join condition.

The following is an example of a left outer join.

```
gSQL> \EXPLAIN PLAN
SELECT r_name, n_name
  FROM region
       LEFT OUTER JOIN
       nation
       ON  r_regionkey = n_regionkey
       AND n_nationkey > 20;
R_NAME                    N_NAME
------------------------- -------------------------
AFRICA                    null
AMERICA                   UNITED STATES
```

```
ASIA                    VIETNAM
EUROPE                  UNITED KINGDOM
EUROPE                  RUSSIA
MIDDLE EAST             null
6 rows selected.
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                  ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                  ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                     ¦
¦    2  ¦      HASH JOIN (LEFT OUTER JOIN)                                 ¦
¦    3  ¦        TABLE ACCESS ("REGION")                                   ¦
¦    4  ¦        HASH JOIN INSTANT                                         ¦
¦    5  ¦          INDEX ACCESS ("NATION", "NATION_PK_INDEX")              ¦
=============================================================================
     1  -  TARGET : REGION.R_NAME, NATION.N_NAME
     2  -  JOINED COLUMN : REGION.R_NAME, NATION.N_NAME
     3  -  READ COLUMN : REGION.R_REGIONKEY, REGION.R_NAME
     4  -  HASH KEY : NATION.N_REGIONKEY
           RECORD COLUMN : NATION.N_NAME
           READ KEY COLUMN : NATION.N_REGIONKEY, NATION.N_NAME
             HASH FILTER : NATION.N_REGIONKEY = REGION.R_REGIONKEY
     5  -  READ INDEX COLUMN : NATION.N_NATIONKEY
           READ TABLE COLUMN : NATION.N_NAME, NATION.N_REGIONKEY
             MIN RANGE : NATION.N_NATIONKEY > 20
             MAX RANGE : NATION.N_NATIONKEY IS NOT NULL
<<<  end print plan
```

Right Outer Join

It returns all rows satisfying a join condition as result, and also returns rows in a right table as result even though the rows do not satisfy the join condition.

The following is an example of a right outer join.

```
gSQL> \EXPLAIN PLAN
SELECT r_name, n_name
  FROM region RIGHT OUTER JOIN nation ON  r_regionkey = n_regionkey
                                      AND n_nationkey > 20;
R_NAME N_NAME
```

```
------ -------------------------
null                     ALGERIA
null                     ARGENTINA
null                     BRAZIL
null                     CANADA
null                     EGYPT
null                     ETHIOPIA
null                     FRANCE
null                     GERMANY
null                     INDIA
null                     INDONESIA
null                     IRAN
null                     IRAQ
null                     JAPAN
null                     JORDAN
null                     KENYA
null                     MOROCCO
null                     MOZAMBIQUE
null                     PERU
null                     CHINA
null                     ROMANIA
null                     SAUDI ARABIA
ASIA                     VIETNAM
EUROPE                   RUSSIA
EUROPE                   UNITED KINGDOM
AMERICA                  UNITED STATES
25 rows selected.
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                  ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                  ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                     ¦
¦    2  ¦      HASH JOIN (LEFT OUTER JOIN)                                 ¦
¦    3  ¦        TABLE ACCESS ("NATION")                                   ¦
¦    4  ¦        HASH JOIN INSTANT                                         ¦
¦    5  ¦          TABLE ACCESS ("REGION")                                 ¦
=============================================================================
     1  -  TARGET : REGION.R_NAME, NATION.N_NAME
     2  -  JOINED COLUMN : REGION.R_NAME, NATION.N_NAME
```

```
     3  -  READ COLUMN : NATION.N_NATIONKEY, NATION.N_NAME, NATION.N_REGIONKEY
     4  -  HASH KEY : REGION.R_REGIONKEY
           RECORD COLUMN : REGION.R_NAME
           READ KEY COLUMN : REGION.R_REGIONKEY, REGION.R_NAME
             HASH FILTER : REGION.R_REGIONKEY = NATION.N_REGIONKEY
             LOGICAL FILTER : {NATION.N_NATIONKEY} > 20
           FETCH ONE ROW
     5  -  READ COLUMN : REGION.R_REGIONKEY, REGION.R_NAME
<<<  end print plan
```

Full Outer Join

It returns all rows satisfying a join condition as result, and also returns all rows in both of a left table and a right table as result even though the rows do not satisfy the join condition.

The following is an example of a full outer join.

```
gSQL> \EXPLAIN PLAN
SELECT r_name, n_name
  FROM region FULL OUTER JOIN nation ON  r_regionkey = n_regionkey
                                     AND n_nationkey > 20;
R_NAME N_NAME
------ -------------------------
null                      ALGERIA
null                      ARGENTINA
null                      BRAZIL
null                      CANADA
null                      EGYPT
null                      ETHIOPIA
null                      FRANCE
null                      GERMANY
null                      INDIA
null                      INDONESIA
null                      IRAN
null                      IRAQ
null                      JAPAN
null                      JORDAN
null                      KENYA
null                      MOROCCO
null                      MOZAMBIQUE
null                      PERU
null                      CHINA
```

```
null                      ROMANIA
null                      SAUDI ARABIA
ASIA                      VIETNAM
EUROPE                    RUSSIA
EUROPE                    UNITED KINGDOM
AMERICA                   UNITED STATES
AFRICA                    null
MIDDLE EAST               null
27 rows selected.
>>>  start print plan
< Execution Plan >
=========================================================================
¦  IDX  ¦  NODE DESCRIPTION                                              ¦
-------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                              ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                  ¦
¦    2  ¦      HASH JOIN (FULL OUTER JOIN)                              ¦
¦    3  ¦        TABLE ACCESS ("NATION")                                ¦
¦    4  ¦        HASH JOIN INSTANT                                      ¦
¦    5  ¦          TABLE ACCESS ("REGION")                              ¦
=========================================================================
     1  -  TARGET : REGION.R_NAME, NATION.N_NAME
     2  -  JOINED COLUMN : REGION.R_NAME, NATION.N_NAME
     3  -  READ COLUMN : NATION.N_NATIONKEY, NATION.N_NAME, NATION.N_REGIONKEY
     4  -  HASH KEY : REGION.R_REGIONKEY
           RECORD COLUMN : REGION.R_NAME
           READ KEY COLUMN : REGION.R_REGIONKEY, REGION.R_NAME
             HASH FILTER : REGION.R_REGIONKEY = NATION.N_REGIONKEY
             LOGICAL FILTER : {NATION.N_NATIONKEY} > 20
     5  -  READ COLUMN : REGION.R_REGIONKEY, REGION.R_NAME
<<<  end print plan
```

Semi Join

A semi join can not be explicitly specified by using SQL statement. When a user used a subquery together with a quantifier such as IN, EXISTS, =ANY, then a rewriter converts it to a semi join while unnesting the s ubquery.

When a row satisfying the join condition exists, then it returns the row of a main query as result.

The following is an example of a semi join.

```
\EXPLAIN PLAN
SELECT o_orderpriority,
       count(*) as order_count
  FROM orders
 WHERE o_orderdate = date '1993-07-01'
   AND EXISTS (
               SELECT *
                 FROM lineitem
                WHERE l_orderkey = o_orderkey
                  AND l_commitdate < l_receiptdate
             )
GROUP BY o_orderpriority
ORDER BY o_orderpriority;
O_ORDERPRIORITY ORDER_COUNT
--------------- -----------
1-URGENT                113
2-HIGH                  136
3-MEDIUM                112
4-NOT SPECIFIED         103
5-LOW                    97
5 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                                                    |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                    |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                         |
|    2  |      SORT INSTANT                                                    |
|    3  |        GROUP HASH INSTANT                                            |
|    4  |          NESTED JOIN (SEMI)                                          |
|    5  |            TABLE ACCESS ("ORDERS")                                   |
|    6  |            INDEX ACCESS ("LINEITEM", "LINEITEM_ORDERKEY_FK")         |
================================================================================
     1  -  TARGET : ORDERS.O_ORDERPRIORITY, COUNT(*) AS ORDER_COUNT
     2  -  SORT KEY : "ORDERS.O_ORDERPRIORITY ASC NULLS LAST"
           RECORD COLUMN : COUNT(*)
           READ KEY COLUMN : ORDERS.O_ORDERPRIORITY
           READ RECORD COLUMN : COUNT(*)
     3  -  GROUP KEY : ORDERS.O_ORDERPRIORITY
           RECORD COLUMN : COUNT(*)
```

```
           READ KEY COLUMN : ORDERS.O_ORDERPRIORITY
           READ RECORD COLUMN : COUNT(*)
     4  -  JOINED COLUMN : ORDERS.O_ORDERPRIORITY
     5  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_ORDERDATE, ORDERS.O_ORDERPRIORITY
             PHYSICAL FILTER : ORDERS.O_ORDERDATE = DATE'1993-07-01'
     6  -  READ INDEX COLUMN : LINEITEM.L_ORDERKEY
           READ TABLE COLUMN : LINEITEM.L_COMMITDATE, LINEITEM.L_RECEIPTDATE
             MIN RANGE : LINEITEM.L_ORDERKEY = {ORDERS.O_ORDERKEY}
             MAX RANGE : LINEITEM.L_ORDERKEY = {ORDERS.O_ORDERKEY}
             PHYSICAL TABLE FILTER :
                         LINEITEM.L_RECEIPTDATE > LINEITEM.L_COMMITDATE
<<<  end print plan
```

Anti Semi Join

An anti semi join can not be explicitly specified by using SQL statement. When a user used a subquery tog ether with a quantifier such as NOT IN, NOT EXISTS, !=ALL, =ALL, then a rewriter converts it to an anti se mi join while unnesting the subquery.

When a row satisfying the join condition does not exist, then it returns the row of a main query as result.

When a nullable column exists in the join condition, then it is performed as a null-aware anti-semi join. O therwise, it is performed as an anti-semi join.

The following is an example of an anti semi join.
Both p_partkey and ps_partkey are primary key columns. Therefore, all of them are not null columns.

```
\EXPLAIN PLAN
SELECT p_name, p_brand
  FROM part
 WHERE p_partkey NOT IN ( SELECT ps_partkey
                            FROM partsupp
                           WHERE ps_availqty > 5000 );
...
12511 rows selected.
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                  ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                  ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                     ¦
¦    2  ¦      HASH JOIN (ANTI SEMI)                                       ¦
```

```
|    3  |        TABLE ACCESS ("PART")                                  |
|    4  |        HASH JOIN INSTANT (UNIQUE)                             |
|    5  |          TABLE ACCESS ("PARTSUPP")                            |
=================================================================
     1  -  TARGET : PART.P_NAME, PART.P_BRAND
     2  -  JOINED COLUMN : PART.P_NAME, PART.P_BRAND
     3  -  READ COLUMN : PART.P_PARTKEY, PART.P_NAME, PART.P_BRAND
     4  -  HASH KEY : PARTSUPP.PS_PARTKEY
           READ KEY COLUMN : PARTSUPP.PS_PARTKEY
             HASH FILTER : PARTSUPP.PS_PARTKEY = PART.P_PARTKEY
           FETCH ONE ROW
     5  -  READ COLUMN : PARTSUPP.PS_PARTKEY, PARTSUPP.PS_AVAILQTY
             PHYSICAL FILTER : PARTSUPP.PS_AVAILQTY > 5000
<<<  end print plan
```

The following is an example of a null-aware anti-semi join.
p_partkey is not null because it is a primary key column, but l_partkey is nullable.

```
\EXPLAIN PLAN
SELECT p_name, p_brand
  FROM part
 WHERE p_partkey NOT IN ( SELECT l_partkey
                            FROM lineitem
                           WHERE l_quantity > 30 );
no rows selected.
>>>  start print plan
< Execution Plan >
=================================================================
|  IDX  |  NODE DESCRIPTION                                       |
-----------------------------------------------------------------
|    0  |  SELECT STATEMENT                                       |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                            |
|    2  |      HASH JOIN (ANTI SEMI NA)                           |
|    3  |        TABLE ACCESS ("PART")                            |
|    4  |        HASH JOIN INSTANT (UNIQUE)                       |
|    5  |          TABLE ACCESS ("LINEITEM")                      |
=================================================================
     1  -  TARGET : PART.P_NAME, PART.P_BRAND
     2  -  JOINED COLUMN : PART.P_NAME, PART.P_BRAND
     3  -  READ COLUMN : PART.P_PARTKEY, PART.P_NAME, PART.P_BRAND
     4  -  HASH KEY : LINEITEM.L_PARTKEY
```

```
            READ KEY COLUMN : LINEITEM.L_PARTKEY
              HASH FILTER : LINEITEM.L_PARTKEY = PART.P_PARTKEY
            FETCH ONE ROW
      5  -  READ COLUMN : LINEITEM.L_PARTKEY, LINEITEM.L_QUANTITY
              PHYSICAL FILTER : LINEITEM.L_QUANTITY > 30
<<<  end print plan
```

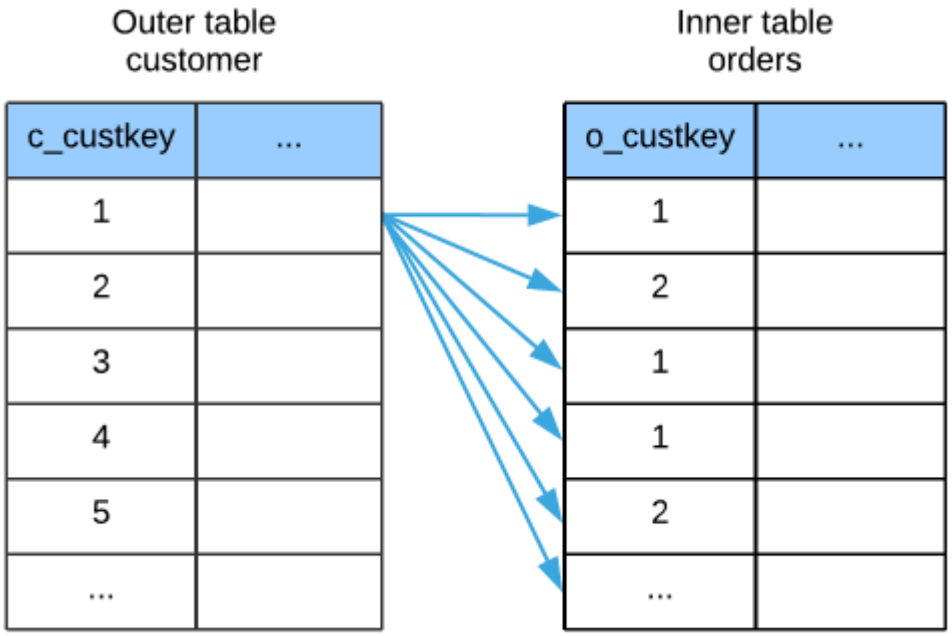
Join Method

Join operation methods between two tables are a nested loops join, a sort merge join and a hash join. An enumerator calculates the cost for those join methods, then select the join operation of the lowest cost.

Nested Loops Join

It scans all rows in an inner table for each row of an outer table, then retrieves results satisfying the join c ondition.

Figure 22 Nested loop join



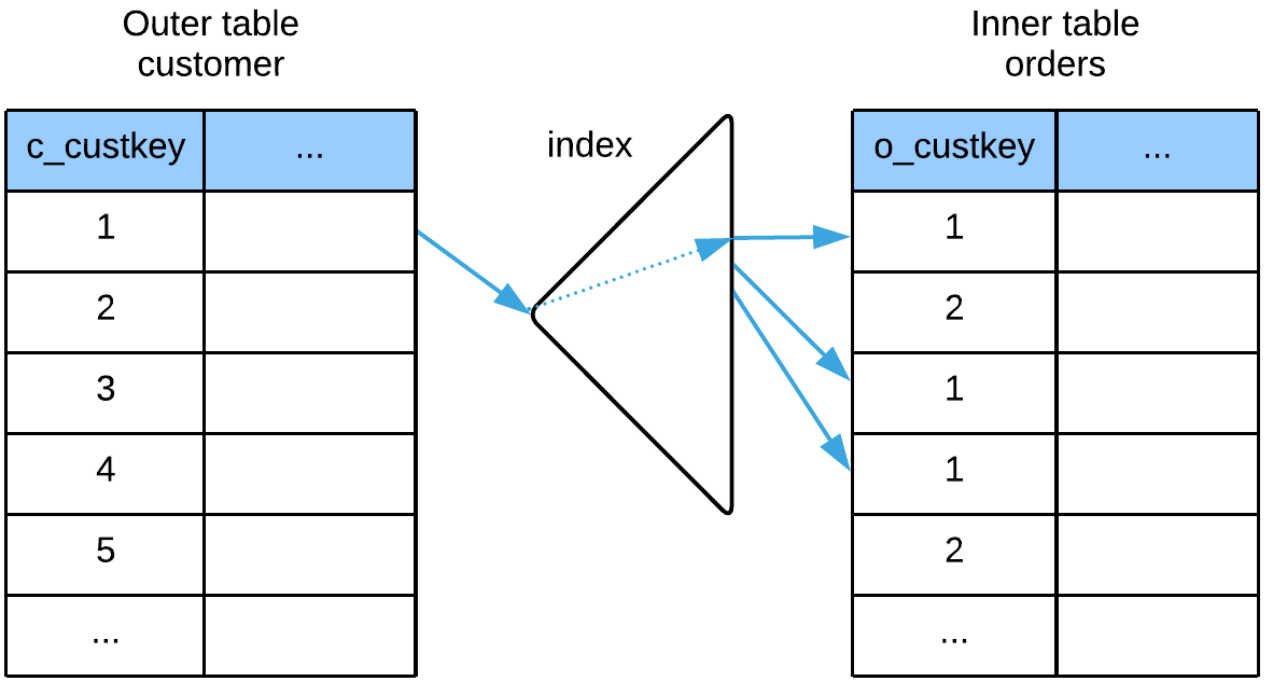
It performs the full scan for the inner table as many as the number of rows in an outer table, so the less r ows in an outer table the better.

Even the join without a join condition can return the execution result to a cartesian product through a ne sted loop join. Therefore, a nested loop join is available even when a hash join and a sort merge join are not available.

Index Nested Loops Join

It performs an index nested loop join when it can retrieve the row satisfying the join condition by using a n index in an Inner table. An index access accesses only to necessary rows, so the performance is improve d.

Figure 23 Index nested loop join



The following is an example of an index nested loop join.

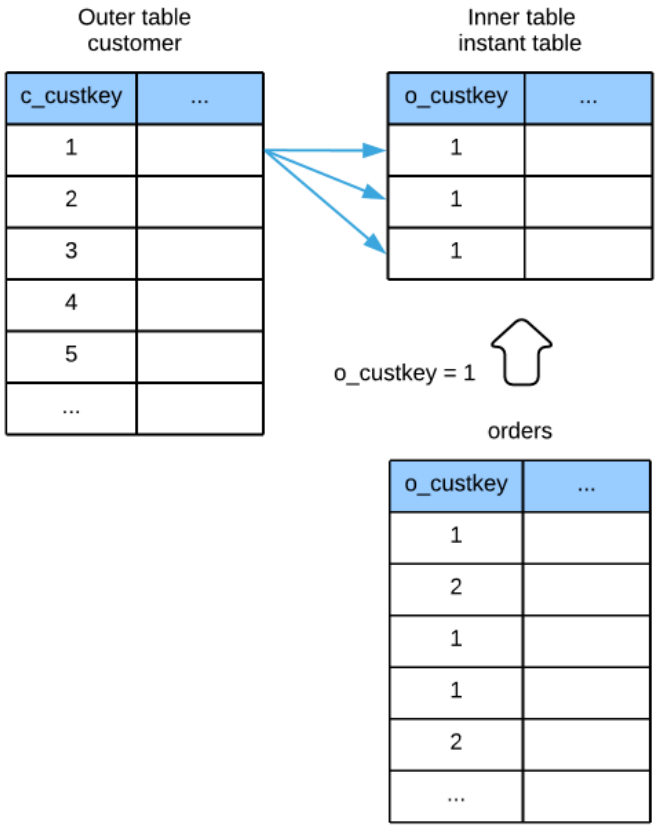
```
\EXPLAIN PLAN
  SELECT c_custkey, count(o_orderkey)
    FROM customer, orders
   WHERE c_custkey = o_custkey
     AND c_comment like '%special%requests%'
GROUP BY c_custkey;
...
2265 rows selected.
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                  ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                  ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                     ¦
¦    2  ¦      GROUP HASH INSTANT                                          ¦
```

```
|    3  |        NESTED JOIN (INNER JOIN)                                  |
|    4  |          TABLE ACCESS ("CUSTOMER")                               |
|    5  |          INDEX ACCESS ("ORDERS", "ORDERS_CUSTKEY_FK")            |
=======================================================================
     1  -  TARGET : CUSTOMER.C_CUSTKEY, COUNT( ORDERS.O_ORDERKEY )
     2  -  GROUP KEY : CUSTOMER.C_CUSTKEY
           RECORD COLUMN : COUNT( ORDERS.O_ORDERKEY )
           READ KEY COLUMN : CUSTOMER.C_CUSTKEY
           READ RECORD COLUMN : COUNT( ORDERS.O_ORDERKEY )
     3  -  JOINED COLUMN : CUSTOMER.C_CUSTKEY, ORDERS.O_ORDERKEY
     4  -  READ COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_COMMENT
             LOGICAL FILTER : CUSTOMER.C_COMMENT LIKE '%special%requests%'
     5  -  READ INDEX COLUMN : ORDERS.O_CUSTKEY
           READ TABLE COLUMN : ORDERS.O_ORDERKEY
             MIN RANGE : ORDERS.O_CUSTKEY = {CUSTOMER.C_CUSTKEY}
             MAX RANGE : ORDERS.O_CUSTKEY = {CUSTOMER.C_CUSTKEY}
<<<  end print plan
```

Instant Nested Loops Join

It performs a nested loop join after loading the intermediate result of an inner table on an instant table.

Figure 24 Instant nested loop join



When the condition such as o_custkey = 1 exists as the example above, a nested loop join is available by l oading the intermediate result of *orders* on an instant table.

The following is an example of an instant nested loop join.

```
\EXPLAIN PLAN
  SELECT c_custkey, count(o_orderkey)
    FROM customer, orders
   WHERE c_comment like '%special%requests%'
     AND o_orderdate = date '1995-03-15'
GROUP BY c_custkey;
...
3380 rows selected.
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                  ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                  ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                     ¦
¦    2  ¦      GROUP HASH INSTANT                                          ¦
¦    3  ¦        NESTED JOIN (INNER JOIN)                                  ¦
¦    4  ¦          TABLE ACCESS ("ORDERS")                                 ¦
¦    5  ¦          FLAT JOIN INSTANT                                       ¦
¦    6  ¦            TABLE ACCESS ("CUSTOMER")                             ¦
=============================================================================
     1  -  TARGET : CUSTOMER.C_CUSTKEY, COUNT( ORDERS.O_ORDERKEY )
     2  -  GROUP KEY : CUSTOMER.C_CUSTKEY
           RECORD COLUMN : COUNT( ORDERS.O_ORDERKEY )
           READ KEY COLUMN : CUSTOMER.C_CUSTKEY
           READ RECORD COLUMN : COUNT( ORDERS.O_ORDERKEY )
     3  -  JOINED COLUMN : CUSTOMER.C_CUSTKEY, ORDERS.O_ORDERKEY
     4  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_ORDERDATE
             PHYSICAL FILTER : ORDERS.O_ORDERDATE = DATE'1995-03-15'
     5  -  RECORD COLUMN : CUSTOMER.C_CUSTKEY
           READ COLUMN : CUSTOMER.C_CUSTKEY
     6  -  READ COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_COMMENT
             LOGICAL FILTER : CUSTOMER.C_COMMENT LIKE '%special%requests%'
<<<  end print plan
```

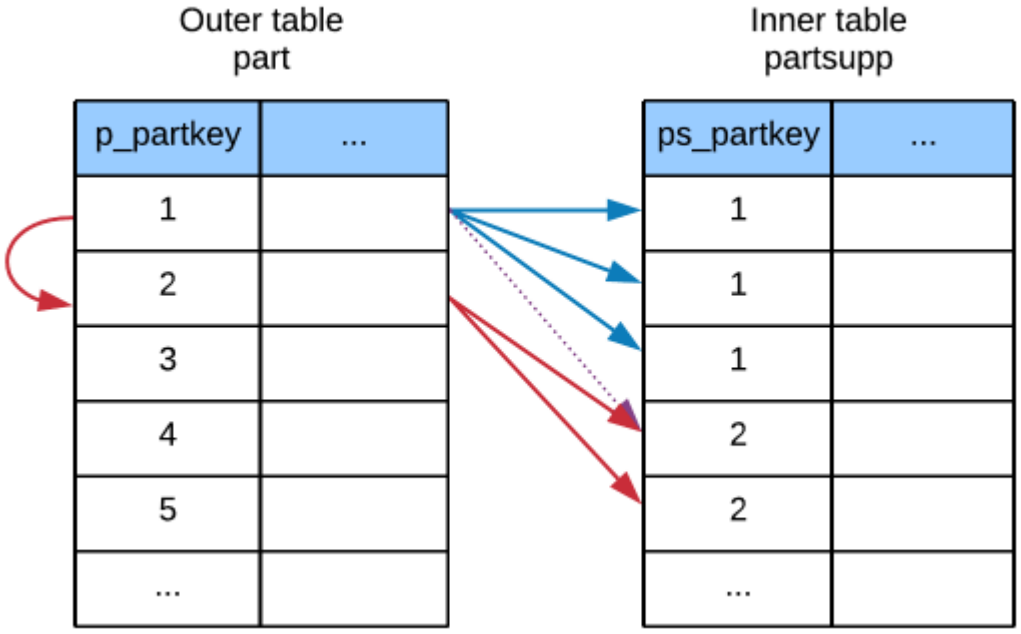
Sort Merge Join

It sorts the intermediate results of an outer table and an inner table, then sequentially compares them, ch ecks whether they satisfy the join condition and returns the join result.

When an index available exists in either an outer table or an inner table, then the table obtains the sorted intermediate results by using an index instead of using a sort instant.

One or more equi join conditions are required to perform a sort merge join.

Figure 25 Sort merge join



The following is an example of a sort merge join.

```
\EXPLAIN PLAN
SELECT p_name, p_brand, p_type
  FROM part, partsupp
 WHERE p_partkey = ps_partkey
   AND p_partkey < 10;
...
36 rows selected.
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                  ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                  ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                     ¦
```

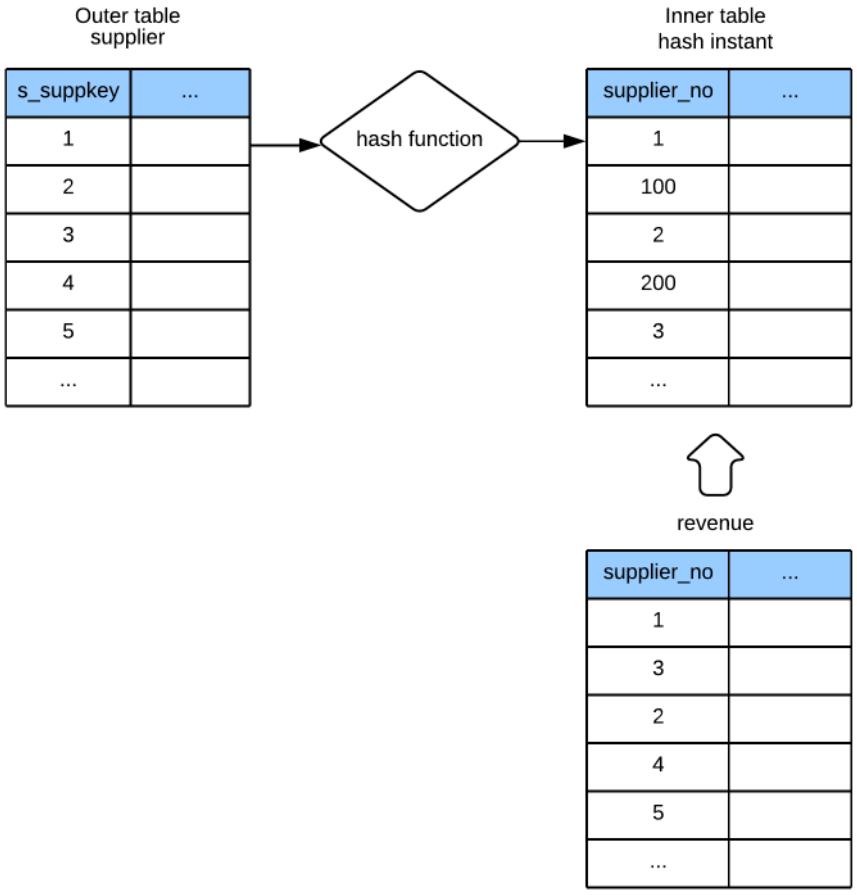
```
|    2  |      MERGE JOIN (INNER JOIN)                                  |
|    3  |        INDEX ACCESS ("PART", "PART_PK_INDEX")                 |
|    4  |        INDEX ACCESS ("PARTSUPP", "PARTSUPP_PARTKEY_FK")       |
=========================================================================
     1  -  TARGET : PART.P_NAME, PART.P_BRAND, PART.P_TYPE
     2  -  JOINED COLUMN : PART.P_NAME, PART.P_BRAND, PART.P_TYPE
             ON FILTER (Equi) : PART.P_PARTKEY = PARTSUPP.PS_PARTKEY
     3  -  READ INDEX COLUMN : PART.P_PARTKEY
           READ TABLE COLUMN : PART.P_NAME, PART.P_BRAND, PART.P_TYPE
             MAX RANGE : PART.P_PARTKEY < 10
     4  -  READ INDEX COLUMN : PARTSUPP.PS_PARTKEY
             MAX RANGE : PARTSUPP.PS_PARTKEY < 10
<<<  end print plan
```

Hash Join

It creates a hash instant in an inner table, then returns the join result satisfying the join condition by using a hash.

One or more equi join conditions are required to perform a hash join.

Figure 26 Hash join



The following is an example of a hash join.

```
\EXPLAIN PLAN
SELECT s_suppkey,
       s_name,
       total_revenue
  FROM supplier,
         (
            SELECT l_suppkey,
                   ROUND( sum(l_extendedprice * (1 - l_discount)), 2)
              FROM lineitem
             WHERE l_shipdate >= date '1996-01-01'
               AND l_shipdate < date '1996-01-01' + interval '3' month
             GROUP BY l_suppkey
       ) revenue(supplier_no, total_revenue)
 WHERE
      s_suppkey = supplier_no;
...
10000 rows selected.
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                  ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                  ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                     ¦
¦    2  ¦      HASH JOIN (INNER JOIN)                                      ¦
¦    3  ¦        TABLE ACCESS ("SUPPLIER")                                 ¦
¦    4  ¦        HASH JOIN INSTANT                                         ¦
¦    5  ¦          INLINE_VIEW ("REVENUE")                                 ¦
¦    6  ¦            QUERY BLOCK ("$QB_IDX_7")                             ¦
¦    7  ¦              GROUP HASH INSTANT                                  ¦
¦    8  ¦                TABLE ACCESS ("LINEITEM")                         ¦
=============================================================================
     1  -  TARGET : SUPPLIER.S_SUPPKEY, SUPPLIER.S_NAME, REVENUE.$C1
     2  -  JOINED COLUMN : SUPPLIER.S_SUPPKEY, SUPPLIER.S_NAME, REVENUE.$C1
     3  -  READ COLUMN : SUPPLIER.S_SUPPKEY, SUPPLIER.S_NAME
     4  -  HASH KEY : REVENUE.L_SUPPKEY
           RECORD COLUMN : REVENUE.$C1
           READ KEY COLUMN : REVENUE.L_SUPPKEY, REVENUE.$C1
             HASH FILTER : REVENUE.L_SUPPKEY = SUPPLIER.S_SUPPKEY
```

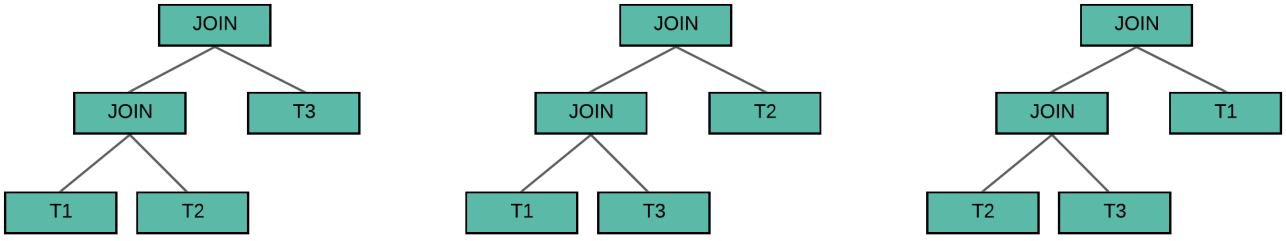
```
     5  -  COLUMN : LINEITEM.L_SUPPKEY AS L_SUPPKEY, ROUND(SUM( LINEITEM.L_EXTENDEDPRICE * ( 1
- LINEITEM.L_DISCOUNT ) ),2) AS $C1
     6  -  TARGET : LINEITEM.L_SUPPKEY, ROUND(SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 -
LINEITEM.L_DISCOUNT ) ),2)
     7  -  GROUP KEY : LINEITEM.L_SUPPKEY
           RECORD COLUMN : SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 - LINEITEM.L_DISCOUNT ) )
           READ KEY COLUMN : LINEITEM.L_SUPPKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 - LINEITEM.L_DISCOUNT ) )
     8  -  READ COLUMN : LINEITEM.L_SUPPKEY, LINEITEM.L_EXTENDEDPRICE, LINEITEM.L_DISCOUNT,
LINEITEM.L_SHIPDATE
             PHYSICAL FILTER : LINEITEM.L_SHIPDATE < DATE'1996-01-01' + CAST( '3' AS
INTERVAL(MONTH) ) AND LINEITEM.L_SHIPDATE >= DATE'1996-01-01'
<<<  end print plan
```

Join Order

When joining three or more tables, the join order should be determined. The join order is determined in t he way of joining two tables, then joining the intermediate result and the next table.

When there are three table, then there are various join orders as follows.

Figure 27 Join order



An enumerator creates sets of various execution plans according to join orders, join methods and access paths available, then it determines the join ordering by selecting the plan whose intermediate results and cost is low.

Group By

It processes *group by*.

Generally, it processes *group by* by creating GROUP HASH INSTANT.

If the intermediate result is sorted for *group by key column* and ascends from the subordinate node, then it can be processed without accumulating separate hash instants.

The following is an example of processing *group by* by creating GROUP HASH INSTANT.

```
\EXPLAIN PLAN
  SELECT c_custkey, count(o_orderkey)
    FROM customer, orders
 WHERE c_custkey = o_custkey
     AND c_comment like '%special%requests%'
     AND o_orderdate = date '1995-03-15'
GROUP BY c_custkey;
...
12 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                      ¦
--------------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                     ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                        ¦
¦    2  ¦      GROUP HASH INSTANT                                             ¦
¦    3  ¦        NESTED JOIN (INNER JOIN)                                     ¦
¦    4  ¦          TABLE ACCESS ("CUSTOMER")                                  ¦
¦    5  ¦          INDEX ACCESS ("ORDERS", "ORDERS_CUSTKEY_FK")               ¦
================================================================================
     1  -  TARGET : CUSTOMER.C_CUSTKEY, COUNT( ORDERS.O_ORDERKEY )
     2  -  GROUP KEY : CUSTOMER.C_CUSTKEY
           RECORD COLUMN : COUNT( ORDERS.O_ORDERKEY )
           READ KEY COLUMN : CUSTOMER.C_CUSTKEY
           READ RECORD COLUMN : COUNT( ORDERS.O_ORDERKEY )
     3  -  JOINED COLUMN : CUSTOMER.C_CUSTKEY, ORDERS.O_ORDERKEY
     4  -  READ COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_COMMENT
             LOGICAL FILTER : CUSTOMER.C_COMMENT LIKE '%special%requests%'
     5  -  READ INDEX COLUMN : ORDERS.O_CUSTKEY
           READ TABLE COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_ORDERDATE
             MIN RANGE : ORDERS.O_CUSTKEY = {CUSTOMER.C_CUSTKEY}
             MAX RANGE : ORDERS.O_CUSTKEY = {CUSTOMER.C_CUSTKEY}
             PHYSICAL TABLE FILTER : ORDERS.O_ORDERDATE = DATE'1995-03-15'
<<<  end print plan
```

The following is an example of processing *group by* by using the sorted intermediate results on the subor dinate node.

```
\EXPLAIN PLAN
  SELECT c_custkey, count(o_orderkey)
    FROM customer, orders
   WHERE c_custkey = o_custkey
     AND c_custkey >= 10
     AND c_custkey < 20
     AND c_comment like '%special%requests%'
     AND o_orderdate = date '1995-03-15'
GROUP BY c_custkey;
...
>>>  start print plan
< Execution Plan >
=========================================================================
¦  IDX  ¦  NODE DESCRIPTION                                              ¦
-------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                              ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                 ¦
¦    2  ¦      GROUP                                                   ¦
¦    3  ¦        NESTED JOIN (INNER JOIN)                              ¦
¦    4  ¦          INDEX ACCESS ("CUSTOMER", "CUSTOMER_PK_INDEX")      ¦
¦    5  ¦          INDEX ACCESS ("ORDERS", "ORDERS_CUSTKEY_FK")        ¦
=========================================================================
     1  -  TARGET : CUSTOMER.C_CUSTKEY, COUNT( ORDERS.O_ORDERKEY )
     2  -  GROUP KEY : CUSTOMER.C_CUSTKEY
           RECORD COLUMN : COUNT( ORDERS.O_ORDERKEY )
     3  -  JOINED COLUMN : CUSTOMER.C_CUSTKEY, ORDERS.O_ORDERKEY
     4  -  READ INDEX COLUMN : CUSTOMER.C_CUSTKEY
           READ TABLE COLUMN : CUSTOMER.C_COMMENT
             MIN RANGE : CUSTOMER.C_CUSTKEY >= 10
             MAX RANGE : CUSTOMER.C_CUSTKEY IS NOT NULL AND CUSTOMER.C_CUSTKEY < 20
             LOGICAL TABLE FILTER : CUSTOMER.C_COMMENT LIKE '%special%requests%'
     5  -  READ INDEX COLUMN : ORDERS.O_CUSTKEY
           READ TABLE COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_ORDERDATE
             MIN RANGE : ORDERS.O_CUSTKEY = {CUSTOMER.C_CUSTKEY} AND ORDERS.O_CUSTKEY >= 10
             MAX RANGE : ORDERS.O_CUSTKEY = {CUSTOMER.C_CUSTKEY} AND ORDERS.O_CUSTKEY < 20
             LOGICAL KEY FILTER : ORDERS.O_CUSTKEY LIKE '%special%requests%'
             PHYSICAL TABLE FILTER : ORDERS.O_ORDERDATE = DATE'1995-03-15'
<<<  end print plan
```

Distinct

It processes distinct.

Generally, it processes distinct by creating GROUP HASH INSTANT.

If the intermediate result is sorted for distinct key column and ascends from the subordinate node, then it can be processed without accumulating separate hash instants.

The following is an example of processing distinct by creating GROUP HASH INSTANT.

```
\EXPLAIN PLAN
SELECT DISTINCT c_nationkey
  FROM customer
 WHERE c_comment like '%special%requests%'
;
...
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                  ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                  ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                     ¦
¦    2  ¦      GROUP HASH INSTANT                                          ¦
¦    3  ¦        TABLE ACCESS ("CUSTOMER")                                 ¦
=============================================================================
     1  -  TARGET : CUSTOMER.C_NATIONKEY
     2  -  GROUP KEY : CUSTOMER.C_NATIONKEY
           READ KEY COLUMN : CUSTOMER.C_NATIONKEY
     3  -  READ COLUMN : CUSTOMER.C_NATIONKEY, CUSTOMER.C_COMMENT
             LOGICAL FILTER : CUSTOMER.C_COMMENT LIKE '%special%requests%'
<<<  end print plan
```

The following is an example of processing distinct by using the sorted intermediate results on the subordi nate node.

```
\EXPLAIN PLAN
  SELECT DISTINCT c_nationkey
    FROM customer
   WHERE c_nationkey >= 15
     AND c_nationkey < 20
```

```
     AND c_comment like '%special%requests%'
;
...
>>>  start print plan
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
|    2  |      GROUP                                                        |
|    3  |        INDEX ACCESS ("CUSTOMER", "CUSTOMER_NATIONKEY_FK")         |
=============================================================================
     1  -  TARGET : CUSTOMER.C_NATIONKEY
     2  -  GROUP KEY : CUSTOMER.C_NATIONKEY
     3  -  READ INDEX COLUMN : CUSTOMER.C_NATIONKEY
           READ TABLE COLUMN : CUSTOMER.C_COMMENT
             MIN RANGE : CUSTOMER.C_NATIONKEY >= 15
             MAX RANGE : CUSTOMER.C_NATIONKEY IS NOT NULL AND CUSTOMER.C_NATIONKEY < 20
             LOGICAL TABLE FILTER : CUSTOMER.C_COMMENT LIKE '%special%requests%'
<<<  end print plan
```

Single Row Aggregation

It processes single row aggregation.

Generally, it performs aggregation by using a hash.

If it is a simple query acquiring MIN(), MAX(), then it may use an index.

The following is an example of processing single row aggregation by using a hash.

```
\EXPLAIN PLAN
SELECT count(o_orderkey)
  FROM customer, orders
 WHERE c_custkey = o_custkey
   AND c_comment like '%special%requests%';
COUNT(O_ORDERKEY)
-----------------
            33526
1 row selected.
```

```
>>>  start print plan
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
|    2  |      AGGREGATION BY HASH                                          |
|    3  |        NESTED JOIN (INNER JOIN)                                   |
|    4  |          TABLE ACCESS ("CUSTOMER")                                |
|    5  |          INDEX ACCESS ("ORDERS", "ORDERS_CUSTKEY_FK")             |
=============================================================================
     1  -  TARGET : COUNT( ORDERS.O_ORDERKEY )
     2  -  AGGREGATION : COUNT( ORDERS.O_ORDERKEY )
     3  -  JOINED COLUMN : ORDERS.O_ORDERKEY
     4  -  READ COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_COMMENT
          LOGICAL FILTER : CUSTOMER.C_COMMENT LIKE '%special%requests%'
     5  -  READ INDEX COLUMN : ORDERS.O_CUSTKEY
          READ TABLE COLUMN : ORDERS.O_ORDERKEY
            MIN RANGE : ORDERS.O_CUSTKEY = {CUSTOMER.C_CUSTKEY}
            MAX RANGE : ORDERS.O_CUSTKEY = {CUSTOMER.C_CUSTKEY}
<<<  end print plan
```

The following is an example of processing single row aggregation by using an index.

```
\EXPLAIN PLAN SELECT MIN(c_custkey), MAX(c_custkey) FROM customer;
MIN(C_CUSTKEY) MAX(C_CUSTKEY)
-------------- --------------
             1         150000
1 row selected.
>>>  start print plan
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
|    2  |      INDEX ACCESS ("CUSTOMER", "CUSTOMER_PK_INDEX")               |
=============================================================================
     1  -  TARGET : MIN( CUSTOMER.C_CUSTKEY ), MAX( CUSTOMER.C_CUSTKEY )
     2  -  READ INDEX COLUMN : CUSTOMER.C_CUSTKEY
```

```
           AGGREGATION : MIN( CUSTOMER.C_CUSTKEY ), MAX( CUSTOMER.C_CUSTKEY )
<<<  end print plan
```

Order By

It processes *order by*.

Generally, it processes order by by creating SORT INSTANT.

The following two methods are used to process *order by* of SORT INSTANT node.

- Sorting by using sort instant table
- Sorting by using limit sort (If *order by* is used together with *limit,* limit sort method is applied.)

If the intermediate result is sorted for order by key column and ascends from the subordinate node, then i t can be processed without accumulating separate sort instants.

The following is an example of processing *order by* by creating SORT INSTANT.

```
\EXPLAIN PLAN
  SELECT c_nationkey
    FROM customer
   WHERE c_comment like '%special%requests%'
ORDER BY c_nationkey;
...
>>>  start print plan
< Execution Plan >
================================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                      ¦
--------------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                     ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                        ¦
¦    2  ¦      SORT INSTANT                                                   ¦
¦    3  ¦        TABLE ACCESS ("CUSTOMER")                                    ¦
================================================================================
     1  -  TARGET : CUSTOMER.C_NATIONKEY
     2  -  SORT KEY : "CUSTOMER.C_NATIONKEY ASC NULLS LAST"
           READ KEY COLUMN : CUSTOMER.C_NATIONKEY
     3  -  READ COLUMN : CUSTOMER.C_NATIONKEY, CUSTOMER.C_COMMENT
             LOGICAL FILTER : CUSTOMER.C_COMMENT LIKE '%special%requests%'
<<<  end print plan
```

The following is an example of processing *order by* by using LIMIT SORT method. It is performed within S ORT INSTANT node.

```
\EXPLAIN PLAN
  SELECT c_nationkey
    FROM customer
   WHERE c_comment like '%special%requests%'
ORDER BY c_nationkey
   LIMIT 3;
...
>>>  start print plan
< Execution Plan >
=============================================================================================
====
¦  IDX  ¦  NODE DESCRIPTION                                          ¦
ROWS ¦
---------------------------------------------------------------------------------------------
----
¦    0  ¦  SELECT STATEMENT                                          ¦
0 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                             ¦
0 ¦
¦    2  ¦      SORT INSTANT                                        ¦
0 ¦
¦    3  ¦        TABLE ACCESS ("CUSTOMER")                         ¦
0 ¦
=============================================================================================
====
     1  -  TARGET : CUSTOMER.C_NATIONKEY
     2  -  LIMIT SORT
           SORT KEY : "CUSTOMER.C_NATIONKEY ASC NULLS LAST"
           READ KEY COLUMN : CUSTOMER.C_NATIONKEY
     3  -  READ COLUMN : CUSTOMER.C_NATIONKEY, CUSTOMER.C_COMMENT
             LOGICAL FILTER : CUSTOMER.C_COMMENT LIKE '%special%requests%'
<<<  end print plan
```

The following is an example of processing *order by* by using the sorted intermediate results on the subord inate node. The *order by* processing is omitted.

```
>>>  start print plan
\EXPLAIN PLAN
```

```
  SELECT c_nationkey
    FROM customer
   WHERE c_comment like '%special%requests%'
     AND c_nationkey >= 15
     AND c_nationkey < 20
ORDER BY c_nationkey;
...
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                  ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                  ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                     ¦
¦    2  ¦      INDEX ACCESS ("CUSTOMER", "CUSTOMER_NATIONKEY_FK")          ¦
=============================================================================
     1  -  TARGET : CUSTOMER.C_NATIONKEY
     2  -  READ INDEX COLUMN : CUSTOMER.C_NATIONKEY
           READ TABLE COLUMN : CUSTOMER.C_COMMENT
             MIN RANGE : CUSTOMER.C_NATIONKEY >= 15
             MAX RANGE : CUSTOMER.C_NATIONKEY IS NOT NULL AND CUSTOMER.C_NATIONKEY < 20
             LOGICAL TABLE FILTER : CUSTOMER.C_COMMENT LIKE '%special%requests%'
<<<  end print plan
```

15.4 Cluster

This chapter describes optimizing the cluster query in the cluster system.
For more information about the cluster system, refer to **GOLDILOCKS Cluster System Architecture**.

Table Sharding Strategy

There are two types of table sharding strategies as follows.

- Cloned table
- Sharded table

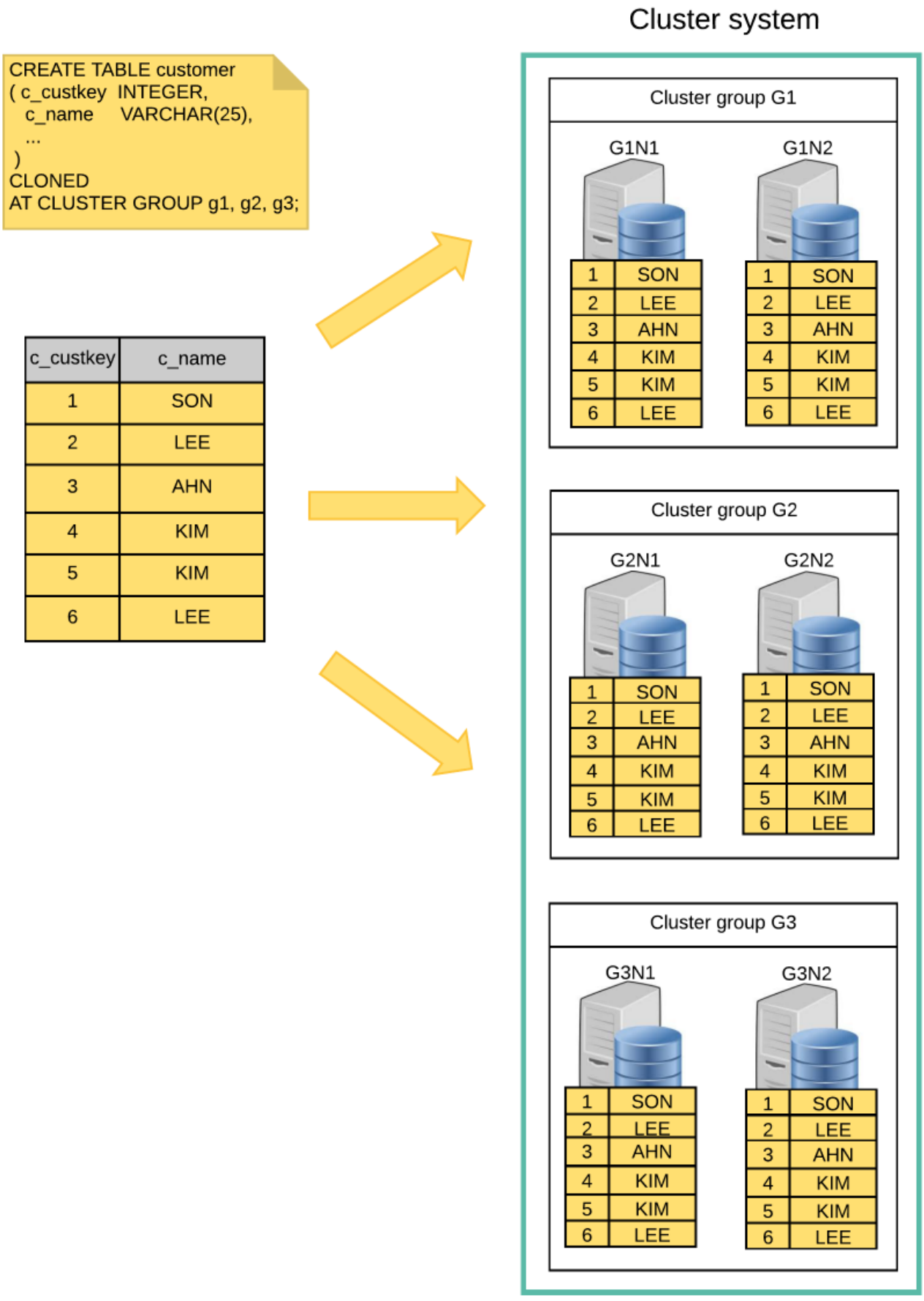
The description and examples of table sharding strategies are as follows.

Cloned Table

All table data of every node in every group are stored same in the cloned table. Therefore it is appropriate for the table which is rarely updated and whose data is little.

The following is an example of creating a customer table with a cloned table.

Figure 28 Cloned table



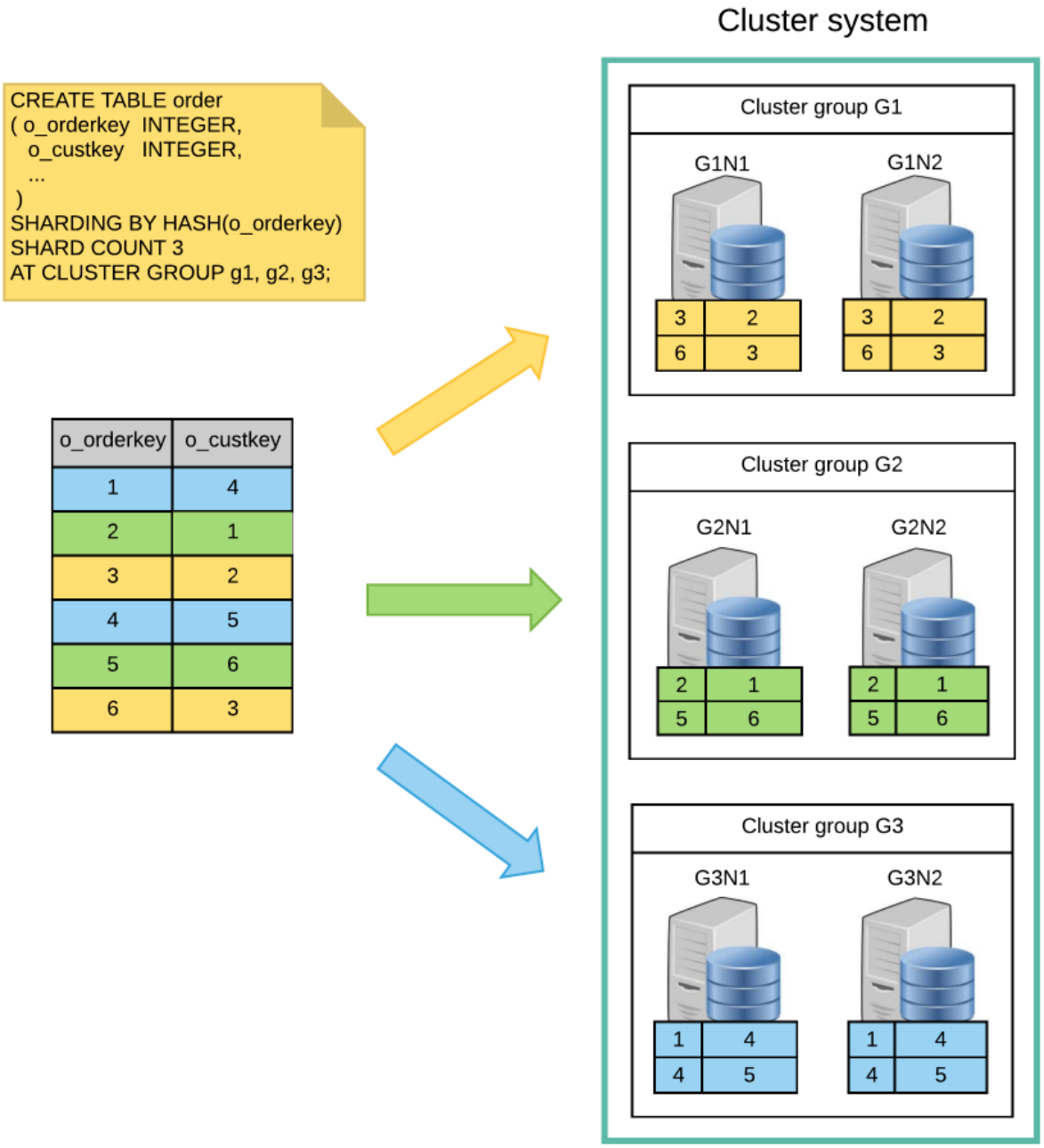
Sharded Table

The data is divided in a group by a shard key, then stored. Nodes in a single group have the same data. T herefore, it is it is appropriate for the table which should be divided due to many data. There are three ty pes of shards according to the sharding strategies as follows.

- Hash shard
- Range shard
- List shard

The following is an example of creating an order table by using hash shards.

Figure 29 Sharded table

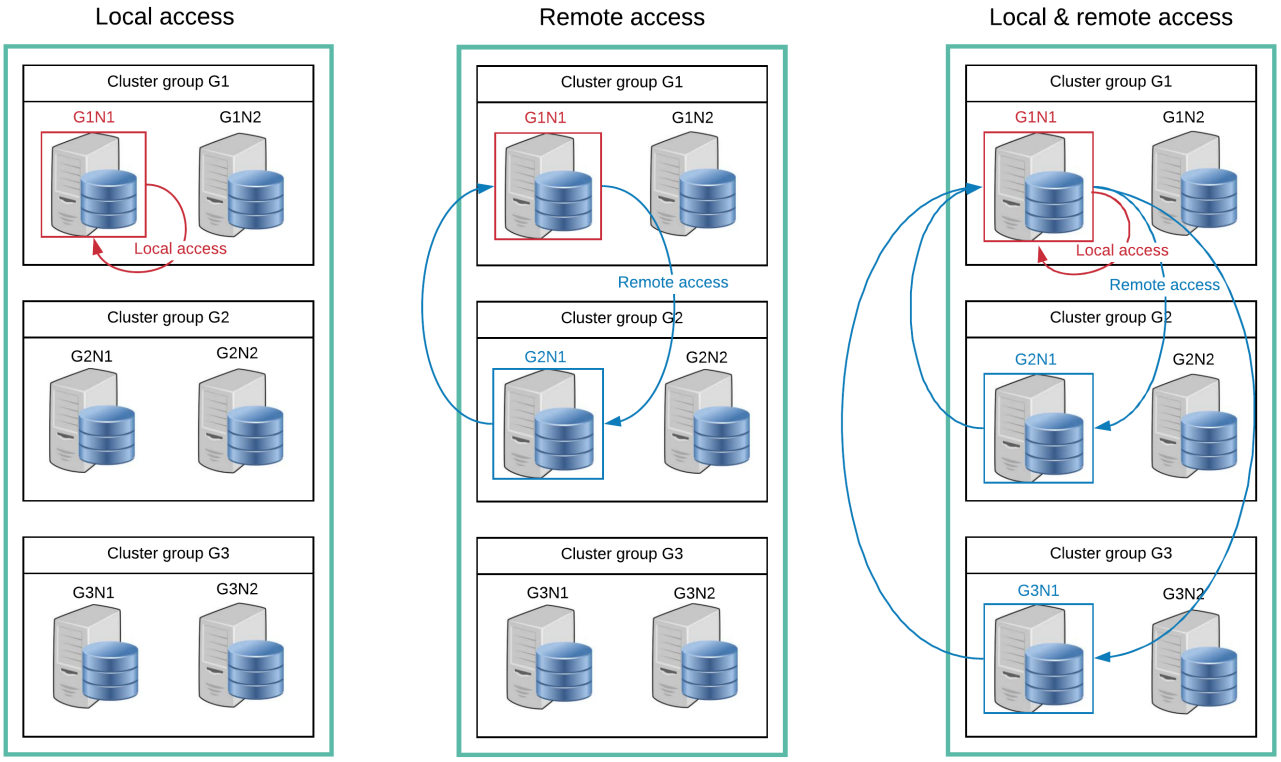


Access

This paragraph describes the case of which a cluster query access to a single table.

The following figure describes a local access and a remote access of when the current server is G1N1.

Figure 30 Cluster access

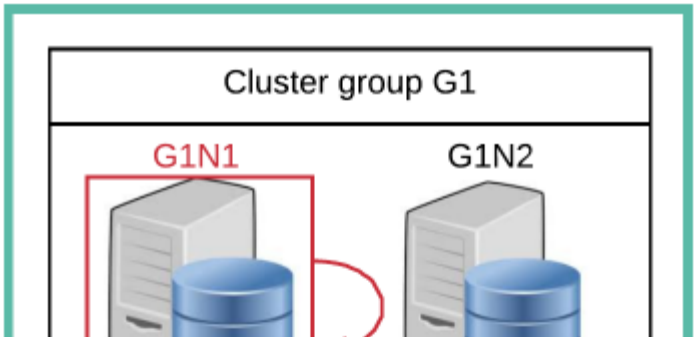


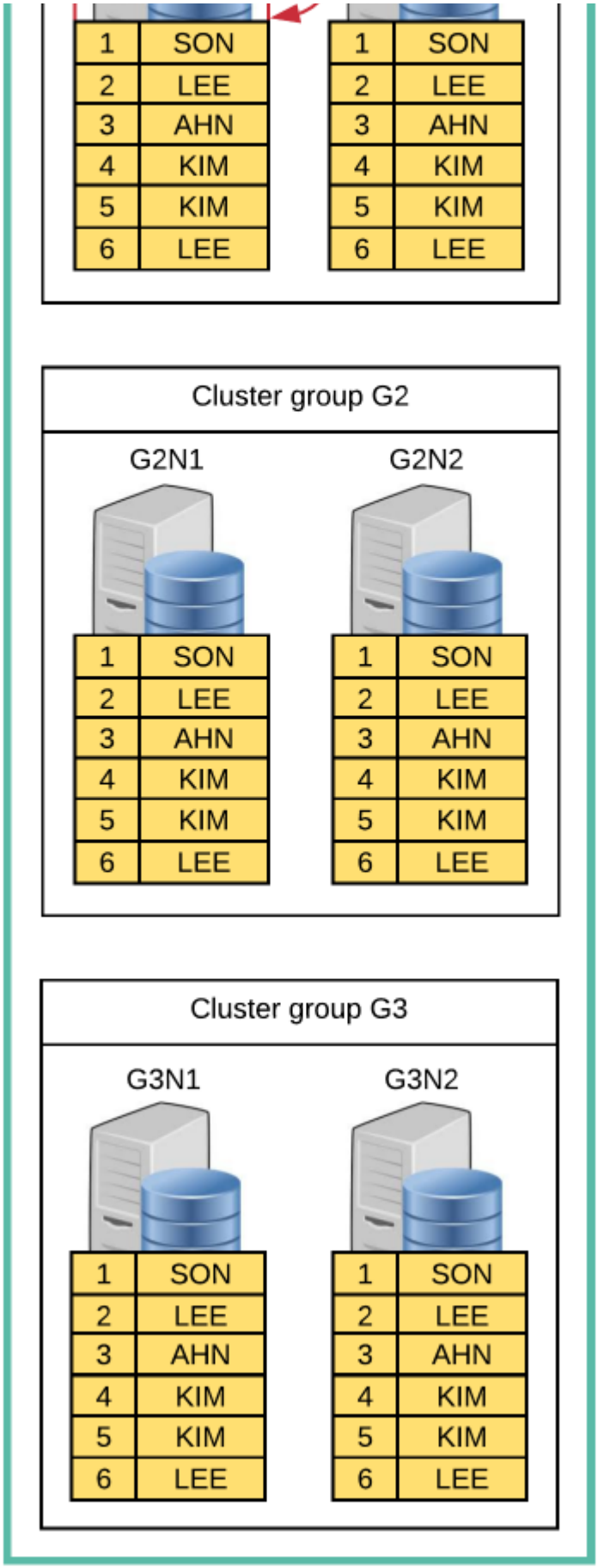
Local Access

It performs the operation only on the current server in driver aspect.

If enquiring about the cloned table as follows, then it only needs to be performed in the current server be cause data in every node in all group are the same.

Figure 31 Local access (cloned table)

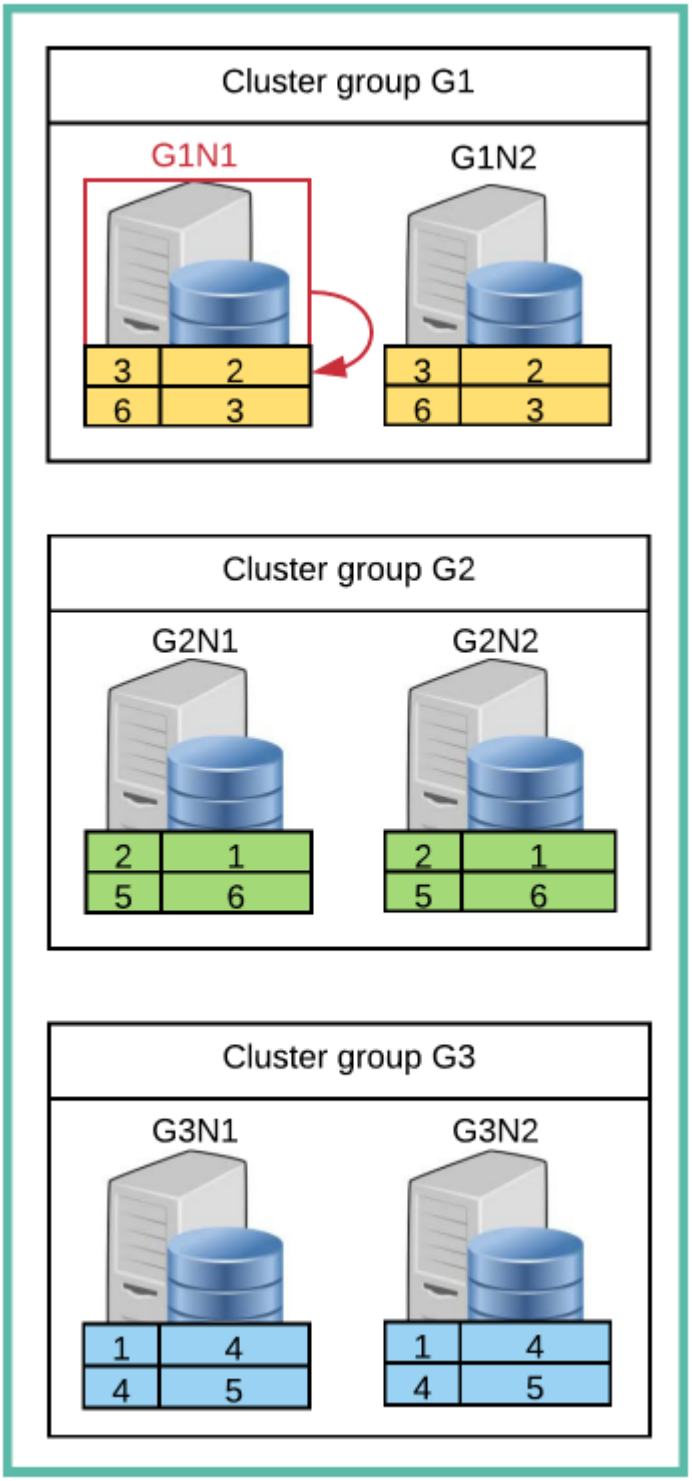



1 SON
2 LEE
3 AHN
4 KIM
5 KIM
6 LEE
1 SON
2 LEE
3 AHN
4 KIM
5 KIM
6 LEE
Cluster group G2
G2N1
G2N2
1 SON
2 LEE
3 AHN
4 KIM
5 KIM
6 LEE
1 SON
2 LEE
3 AHN
4 KIM
5 KIM
6 LEE
Cluster group G3
G3N1
G3N2
1 SON
2 LEE
3 AHN
4 KIM
5 KIM
6 LEE
1 SON
2 LEE
3 AHN
4 KIM
5 KIM
6 LEE

```
\EXPLAIN PLAN SELECT * FROM customer;
C_CUSTKEY C_NAME
--------- ------
        1 SON
        2 LEE
        3 AHN
        4 KIM
        5 KIM
        6 LEE
6 rows selected.
>>>  start print plan
< Execution Plan >
============================================================================
¦ IDX ¦  NODE DESCRIPTION                                          ¦ ROWS ¦
----------------------------------------------------------------------------
¦  0  ¦  SELECT STATEMENT                                         ¦    6 ¦
¦  1  ¦    QUERY BLOCK ("$QB_IDX_2")                            ¦    6 ¦
¦  2  ¦      TABLE ACCESS ("CUSTOMER")                          ¦    6 ¦
============================================================================
     1  -  TARGET : CUSTOMER.C_CUSTKEY, CUSTOMER.C_NAME
     2  -  CLONED
           READ COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_NAME
<<<  end print plan
```

The data is divided in a group by a shard key, then stored in the sharded table. Therefore, it performs the local access when a filter for a shard key exists and the value can be performed only in the current server.

Figure 32 Local access (sharded table)



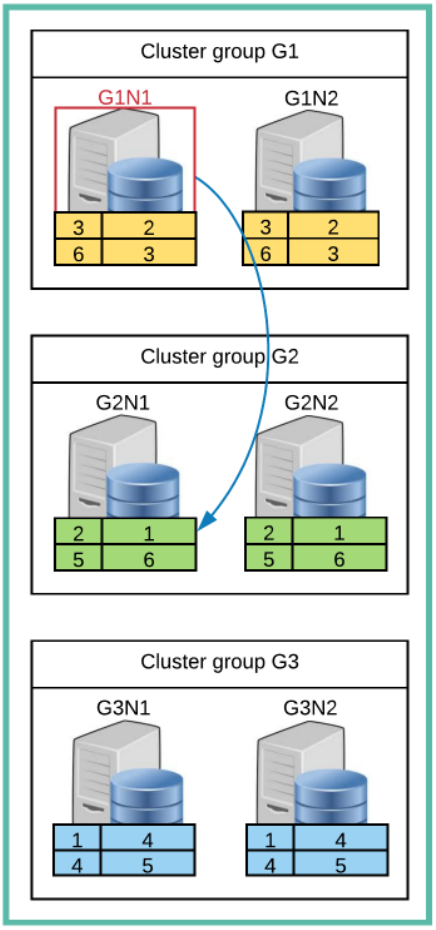

```
\EXPLAIN PLAN SELECT * FROM orders WHERE o_orderkey = 3;
O_ORDERKEY O_CUSTKEY
---------- ---------
         3         2
1 row selected.
>>>  start print plan
```

```
< Execution Plan >
============================================================================
| IDX |  NODE DESCRIPTION                                            | ROWS |
----------------------------------------------------------------------------
|  0  |  SELECT STATEMENT                                            |    1 |
|  1  |    QUERY BLOCK ("$QB_IDX_2")                                 |    1 |
|  2  |      TABLE ACCESS ("ORDERS")                                 |    1 |
============================================================================
     1  -  TARGET : ORDERS.O_ORDERKEY, ORDERS.O_CUSTKEY
     2  -  HASH SHARD ( # 3 )
           READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_CUSTKEY
             PHYSICAL FILTER : ORDERS.O_ORDERKEY = 3
<<<  end print plan
```

Remote Access

The data is divided in a group by a shard key, then stored in the sharded table. Therefore, it can fetch the result by the remote access only to a specific server when a filter for a shard key exists.

Figure 33 Remote access

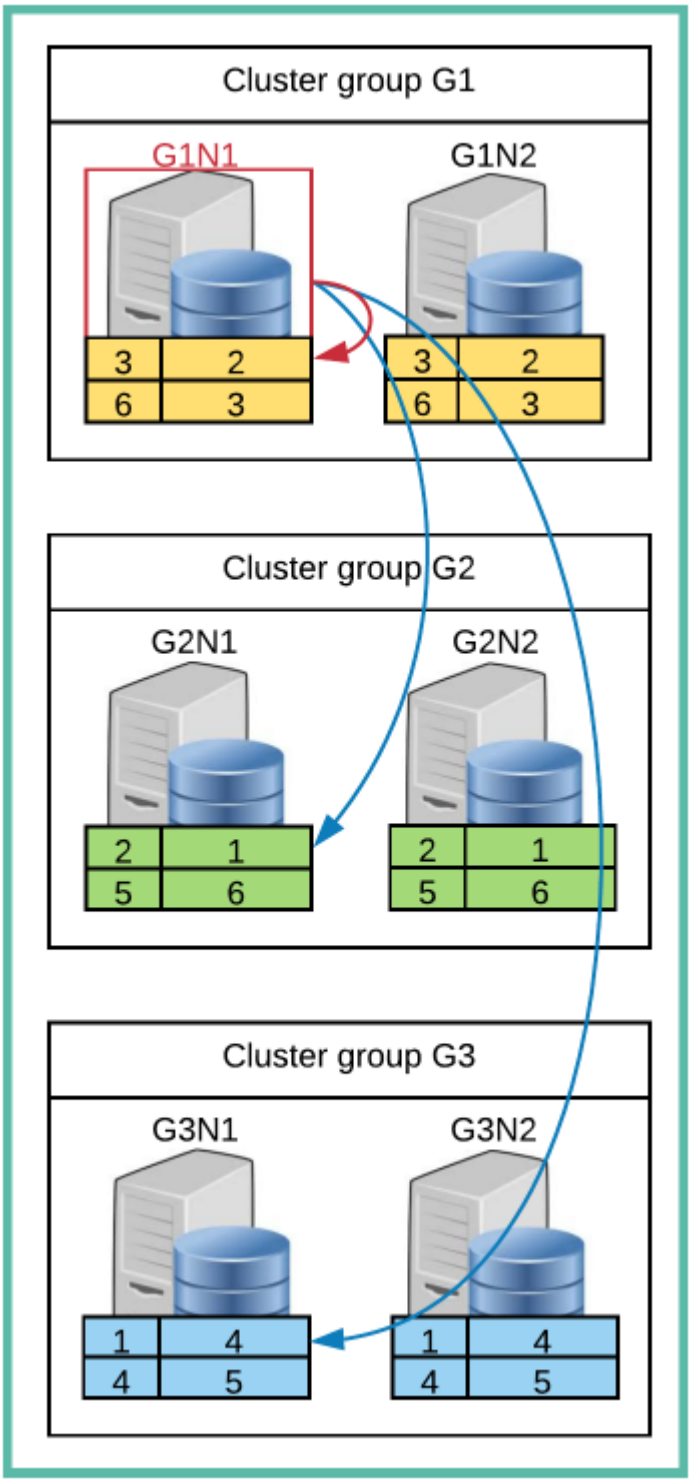


```
\EXPLAIN PLAN SELECT * FROM orders WHERE o_orderkey = 2;
O_ORDERKEY O_CUSTKEY
---------- ---------
         2         1
1 row selected.
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                                |          ROWS |
--------------------------------------------------------------------------------
|  0  |  SELECT STATEMENT                                  |             1 |
|  1  |    QUERY BLOCK ("$QB_IDX_2")                       |             1 |
|  2  |      PLAN BASED CLUSTER                            | REMOTE ONLY 1 |
|  3  |        TABLE ACCESS ("ORDERS")                     |             0 |
================================================================================
     1  -  TARGET : ORDERS.O_ORDERKEY, ORDERS.O_CUSTKEY
     2  -  SQL : SELECT /*+ FULL( _A1 ) */
                         "_A1"."O_ORDERKEY", "_A1"."O_CUSTKEY"
                    FROM "PUBLIC"."ORDERS"@LOCAL AS "_A1"
                   WHERE "_A1"."O_ORDERKEY" = :_V0
           TARGET DOMAIN : G2(G2N1,G2N2) 1 rows
     3  -  HASH SHARD ( # 3 )
           READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_CUSTKEY
             PHYSICAL FILTER : ORDERS.O_ORDERKEY = 2
<<<  end print plan
```

In the execution plan above, it fetched the result by remotely transferring SQL to G2.

If the sharded table does not have a filter for the shard key, then it should receive the result by transferrin g the query to each server.

Figure 34 Local & remote access




```
\EXPLAIN PLAN SELECT * FROM orders WHERE o_custkey = 1;
O_ORDERKEY O_CUSTKEY
---------- ---------
         2         1
1 row selected.
>>>  start print plan
```

```
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                                |          ROWS |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                |             1 |
|  1    |    QUERY BLOCK ("$QB_IDX_2")                     |             1 |
|  2    |      PLAN BASED CLUSTER                          |LOCAL/REMOTE 1 |
|  3    |        TABLE ACCESS ("ORDERS")                   |             0 |
================================================================================
     1  -  TARGET : ORDERS.O_ORDERKEY, ORDERS.O_CUSTKEY
     2  -  SQL : SELECT /*+ FULL( _A1 ) */
                         "_A1"."O_ORDERKEY", "_A1"."O_CUSTKEY"
                    FROM "PUBLIC"."ORDERS"@LOCAL AS "_A1"
                   WHERE "_A1"."O_CUSTKEY" = :_V0
           TARGET DOMAIN : G1(G1N1,G1N2) 0 rows,
                           G2(G2N1,G2N2) 1 rows,
                           G3(G3N1,G3N2) 0 rows
     3  -  HASH SHARD ( # 3 )
           READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_CUSTKEY
             PHYSICAL FILTER : ORDERS.O_CUSTKEY = 1
<<<  end print plan
```

In the execution plan above, it fetched the result by remotely transferring SQL to G1, G2, G3.

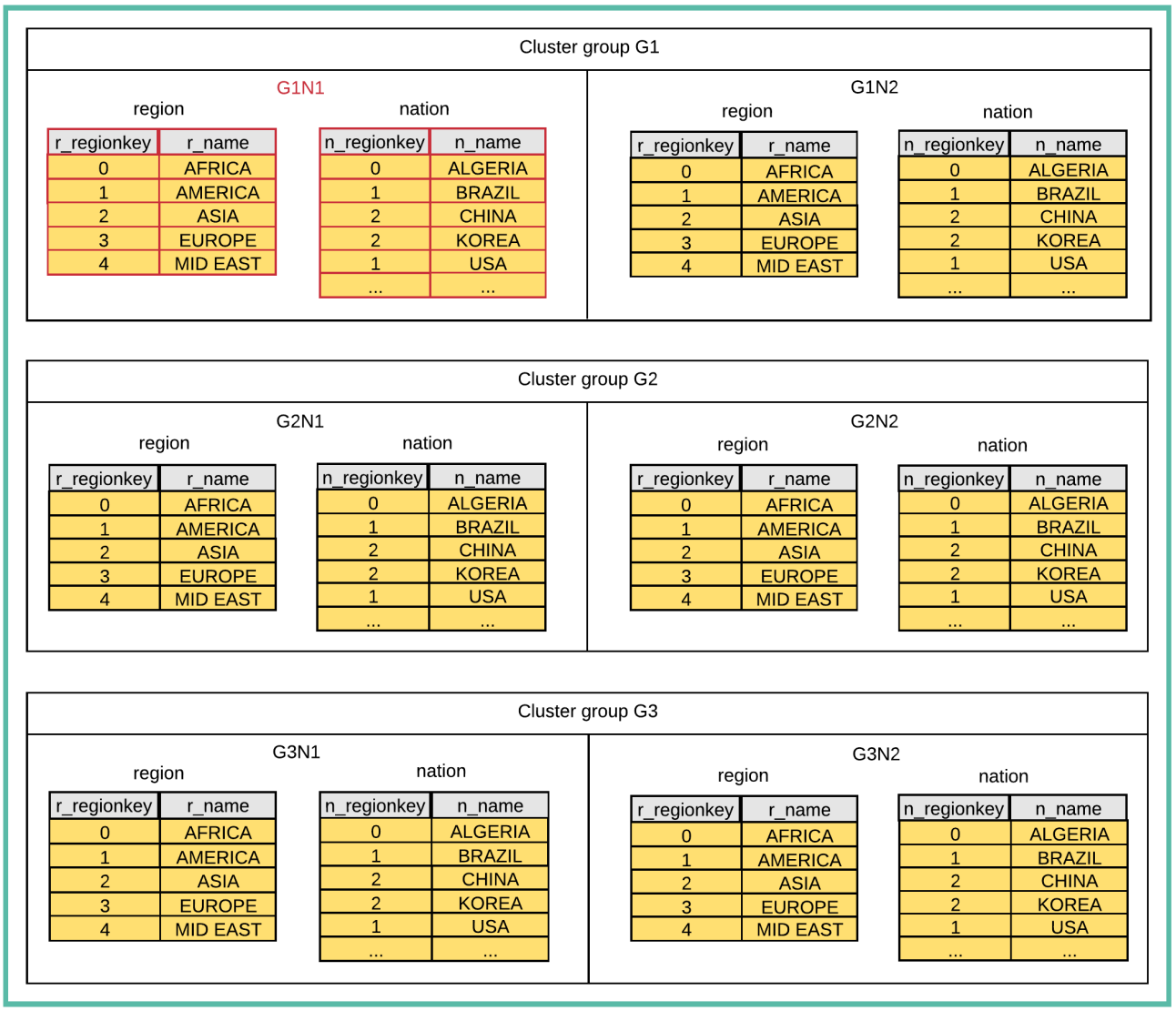
Join

Local Join

It performs join in the current server. Local join types are various as follows.

The following is an example of joining *region* and *nation*. Both tables are cloned tables. All nodes in every group of the cloned tables has the same data. Therefore, it can perform the join only with data in G1N1 i n the following example whose drive is G1N1.

```
\EXPLAIN PLAN
SELECT r_name, n_name
  FROM region, nation
 WHERE r_regionkey = n_regionkey;
```

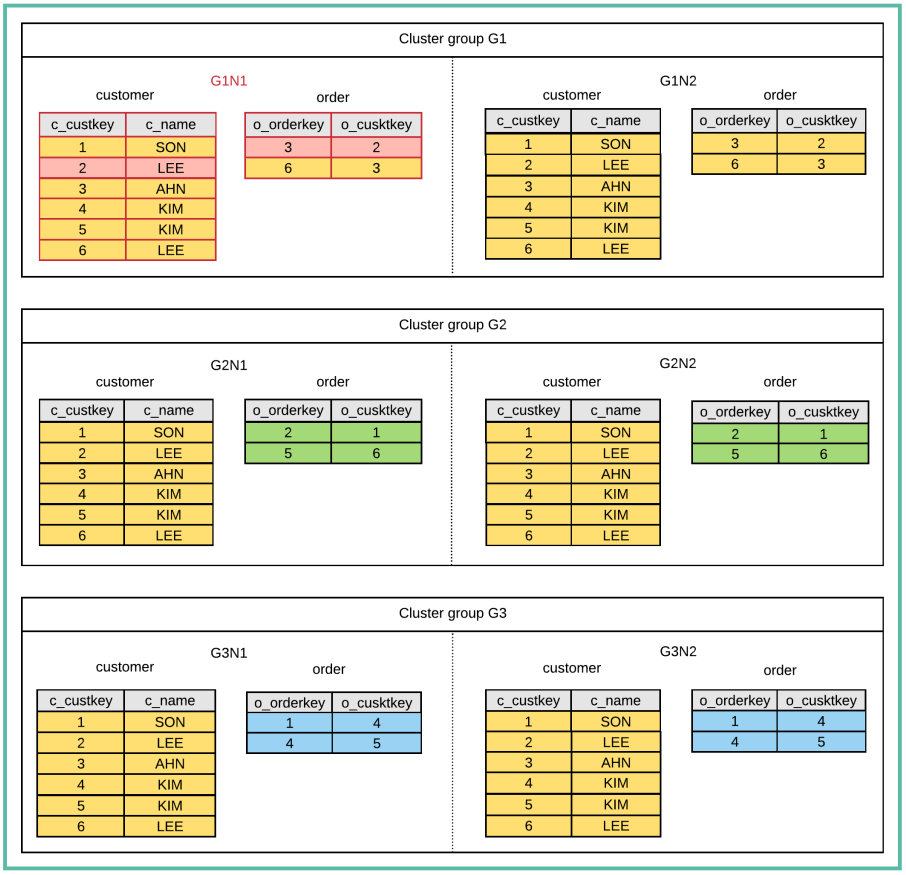



```
>>>  start print plan
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
|    2  |      HASH JOIN (INNER JOIN)                                       |
|    3  |        TABLE ACCESS ("NATION")                                    |
|    4  |        HASH JOIN INSTANT                                          |
|    5  |          TABLE ACCESS ("REGION")                                  |
=============================================================================
     1  -  TARGET : REGION.R_NAME, NATION.N_NAME
     2  -  JOINED COLUMN : REGION.R_NAME, NATION.N_NAME
     3  -  CLONED
```

```
        READ COLUMN : NATION.N_NAME, NATION.N_REGIONKEY
    4  -  HASH KEY : REGION.R_REGIONKEY
        RECORD COLUMN : REGION.R_NAME
        READ KEY COLUMN : REGION.R_REGIONKEY, REGION.R_NAME
          HASH FILTER : REGION.R_REGIONKEY = NATION.N_REGIONKEY
        FETCH ONE ROW
    5  -  CLONED
        READ COLUMN : REGION.R_REGIONKEY, REGION.R_NAME
<<<  end print plan
```

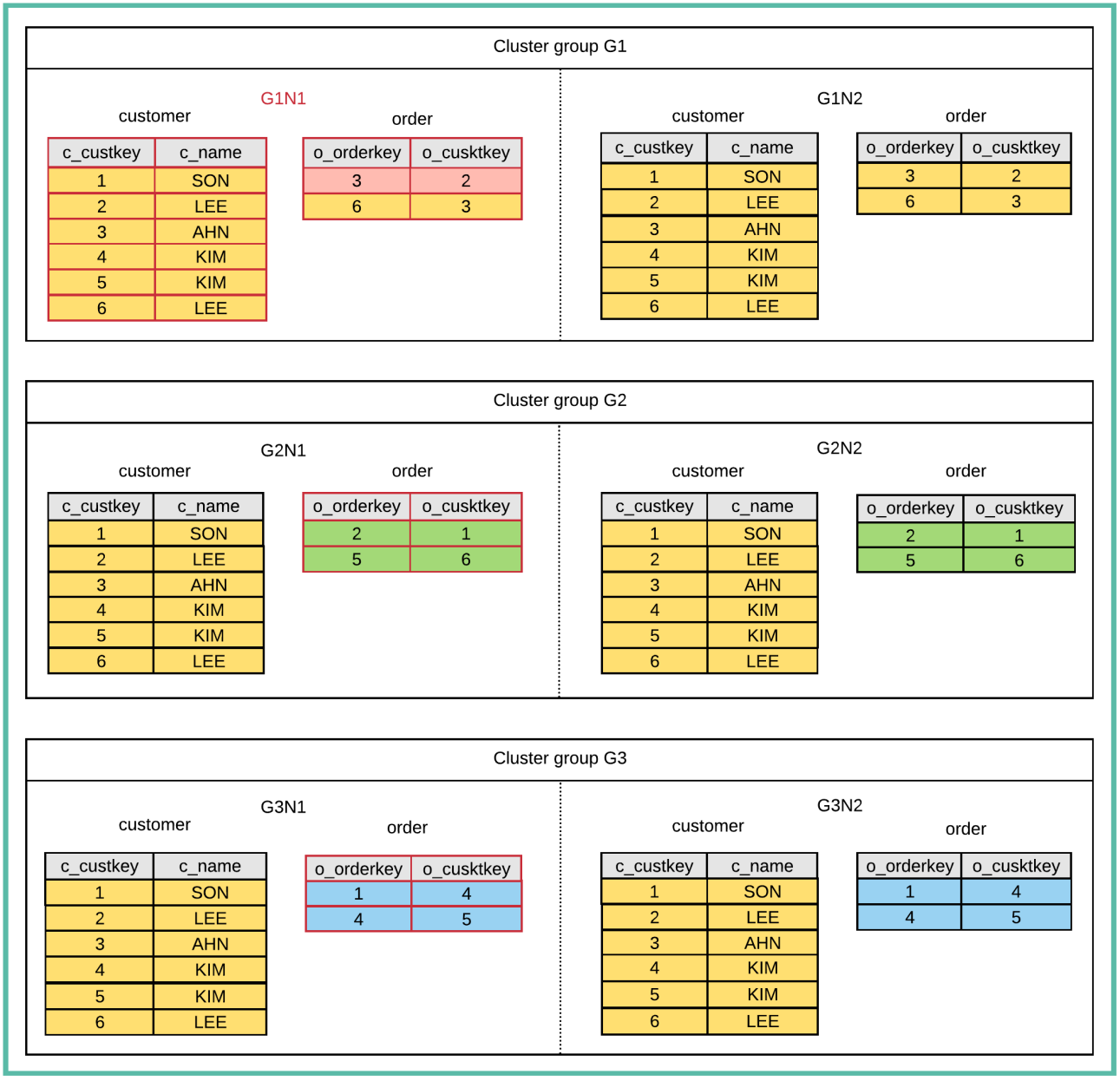
The following is an example of joining customer and orders. Customer is a cloned table and order is a sha rded table. The data in the sharded table divided in a group by a shard key. Therefore, it can perform the local join when a filter for a shard key column exists and the data exists in the current server as the follow ing example.

```
\EXPLAIN PLAN
SELECT c_custkey, COUNT(o_orderkey)
  FROM customer, orders
 WHERE c_custkey = o_custkey
   AND o_orderkey = 3
GROUP BY c_custkey;
```



```
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                                                     |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                     |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                          |
|    2  |      GROUP HASH INSTANT                                               |
|    3  |        NESTED JOIN (INNER JOIN)                                       |
|    4  |          INDEX ACCESS ("ORDERS", "ORDERS_PK_INDEX")                   |
|    5  |          INDEX ACCESS ("CUSTOMER", "CUSTOMER_PK_INDEX")               |
================================================================================
     1  -  TARGET : CUSTOMER.C_CUSTKEY, COUNT( ORDERS.O_ORDERKEY )
     2  -  GROUP KEY : CUSTOMER.C_CUSTKEY
           RECORD COLUMN : COUNT( ORDERS.O_ORDERKEY )
           READ KEY COLUMN : CUSTOMER.C_CUSTKEY
           READ RECORD COLUMN : COUNT( ORDERS.O_ORDERKEY )
     3  -  JOINED COLUMN : CUSTOMER.C_CUSTKEY, ORDERS.O_ORDERKEY
     4  -  HASH SHARD ( # 3 )
           READ INDEX COLUMN : ORDERS.O_ORDERKEY
           READ TABLE COLUMN : ORDERS.O_CUSTKEY
             MIN RANGE : ORDERS.O_ORDERKEY = 3
             MAX RANGE : ORDERS.O_ORDERKEY = 3
           FETCH ONE ROW
     5  -  CLONED
           READ INDEX COLUMN : CUSTOMER.C_CUSTKEY
             MIN RANGE : CUSTOMER.C_CUSTKEY = {ORDERS.O_CUSTKEY}
             MAX RANGE : CUSTOMER.C_CUSTKEY = {ORDERS.O_CUSTKEY}
           FETCH ONE ROW
<<<  end print plan
```

The following is an example of joining customer and orders without a filter for a shard key column. In this case, the local join is available only after fetching all data in the sharded table.




```
\EXPLAIN PLAN
SELECT /*+ LOCAL_JOIN(orders) */
       c_custkey, COUNT(o_orderkey)
  FROM customer, orders
 WHERE c_custkey = o_custkey
GROUP BY c_custkey;
```

```
< Execution Plan >
================================================================================
¦IDX¦  NODE DESCRIPTION                                ¦           ROWS ¦
--------------------------------------------------------------------------------
¦ 0 ¦  SELECT STATEMENT                                ¦              6 ¦
```

```
┆ 1 ┆    QUERY BLOCK ("$QB_IDX_2")                        ┆             6 ┆
┆ 2 ┆      HASH JOIN (INNER JOIN)                         ┆             6 ┆
┆ 3 ┆        PLAN BASED CLUSTER                           ┆ LOCAL/REMOTE 6 ┆
┆ 4 ┆          INDEX ACCESS ("ORDERS", "ORDERS_PK_INDEX") ┆ (6)          6 ┆
┆ 5 ┆        HASH JOIN INSTANT                            ┆             6 ┆
┆ 6 ┆          TABLE ACCESS ("CUSTOMER")                  ┆             6 ┆
=========================================================================
     1  -  TARGET : CUSTOMER.C_CUSTKEY, CUSTOMER.C_NAME, ORDERS.O_ORDERKEY
     2  -  JOINED COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_NAME, ORDERS.O_ORDERKEY
     3  -  SQL : SELECT /*+ INDEX( _A1, "PUBLIC"."ORDERS_PK_INDEX" ) */
                         "_A1"."O_ORDERKEY"
                    FROM "PUBLIC"."ORDERS"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 2 rows,
                           G2(G2N1,G2N2) 2 rows,
                           G3(G3N1,G3N2) 2 rows
     4  -  HASH SHARD ( # 3 )
           READ INDEX COLUMN : ORDERS.O_ORDERKEY
     5  -  HASH KEY : CUSTOMER.C_CUSTKEY
           RECORD COLUMN : CUSTOMER.C_NAME
           READ KEY COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_NAME
             HASH FILTER : CUSTOMER.C_CUSTKEY = ORDERS.O_ORDERKEY
           FETCH ONE ROW
     6  -  CLONED
           READ COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_NAME
<<<  end print plan
```

In the execution plan above, it fetched all data in order table by transferring SQL to G1, G2, G3.

Remote Join

It performs join in each server.

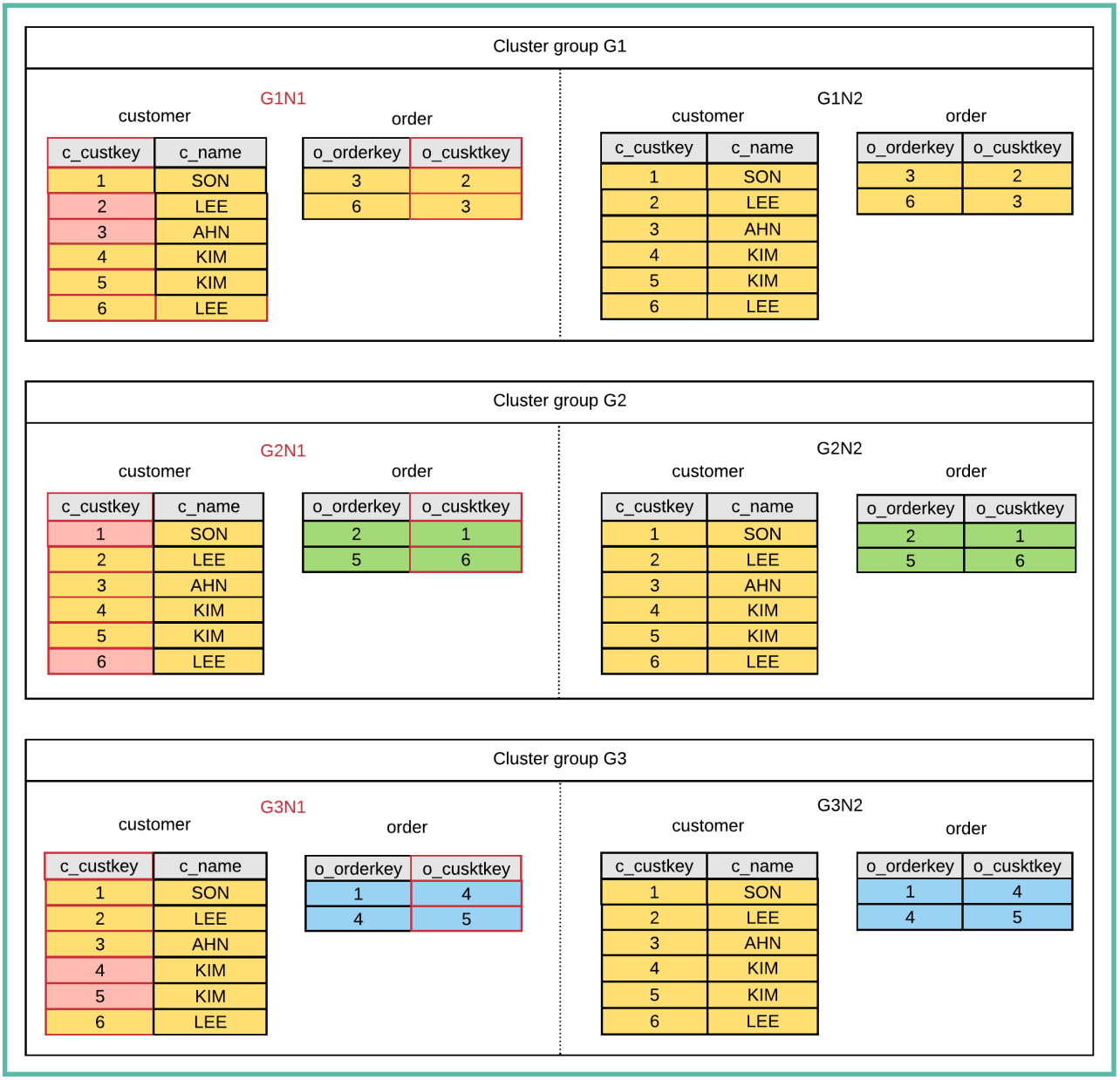
Join processing in each server has a parallel processing effect. Also, when results are decreased a lot by th e join, then bringing the result by performing join in the server reduces the network cost.

This chapter describes various forms of remote.

Joining Cloned Table and Sharded Table

This chapter describes about joining a cloned table and a sharded table.

The following is an example of joining customer and orders. Customer is a cloned table, order is a sharde d table, and the data is distributed as follows.




```
\EXPLAIN PLAN
SELECT /*+ REMOTE_JOIN(orders) */
       c_custkey, COUNT(o_orderkey)
  FROM customer, orders
 WHERE c_custkey = o_custkey
GROUP BY c_custkey;
>>>  start print plan
< Execution Plan >
================================================================================
¦IDX¦  NODE DESCRIPTION                                ¦           ROWS ¦
--------------------------------------------------------------------------------
¦ 0 ¦  SELECT STATEMENT                                ¦              6 ¦
```

```
| 1 |    QUERY BLOCK ("$QB_IDX_2")                          |              6 |
| 2 |      PLAN BASED CLUSTER                               | LOCAL/REMOTE 6 |
| 3 |        HASH JOIN (INNER JOIN)                         |              6 |
| 4 |          TABLE ACCESS ("ORDERS")                      |              6 |
| 5 |          HASH JOIN INSTANT                            |              6 |
| 6 |            TABLE ACCESS ("CUSTOMER")                  |              6 |
=============================================================================
     1  -  TARGET : CUSTOMER.C_CUSTKEY, CUSTOMER.C_NAME, ORDERS.O_ORDERKEY
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE USE_HASH_IN( _A1, 10 )
                            FULL( _A2 )
                            FULL( _A1 ) */
                       "_A1"."C_CUSTKEY", "_A1"."C_NAME", "_A2"."O_ORDERKEY"
                  FROM ("PUBLIC"."ORDERS"@LOCAL AS "_A2"
                        INNER JOIN
                        "PUBLIC"."CUSTOMER"@LOCAL AS "_A1"
                    ON "_A1"."C_CUSTKEY" = "_A2"."O_ORDERKEY") ALIAS "_A3"
           TARGET DOMAIN : G1(G1N1,G1N2) 2 rows,
                           G2(G2N1,G2N2) 2 rows,
                           G3(G3N1,G3N2) 2 rows
     3  -  JOINED COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_NAME, ORDERS.O_ORDERKEY
     4  -  HASH SHARD ( # 3 )
           READ COLUMN : ORDERS.O_ORDERKEY
     5  -  HASH KEY : CUSTOMER.C_CUSTKEY
           RECORD COLUMN : CUSTOMER.C_NAME
           READ KEY COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_NAME
             HASH FILTER : CUSTOMER.C_CUSTKEY = ORDERS.O_ORDERKEY
     6  -  CLONED
           READ COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_NAME
<<<  end print plan
```

In the execution plan above, it performed join in each server and fetched the result by transferring SQL to G1, G2, G3.

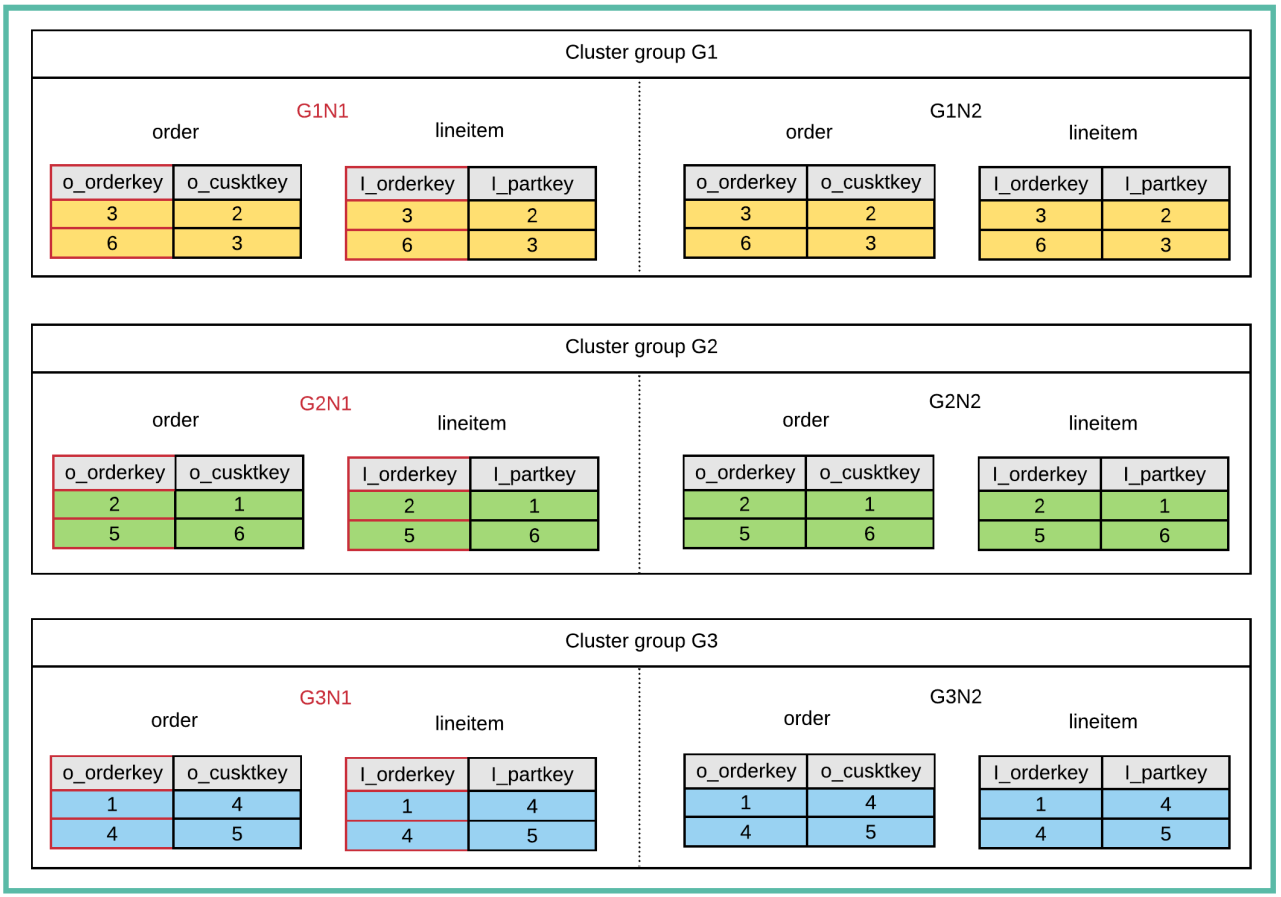
Joining Sharded Table and Sharded Table

The remote join is available when satisfying the following conditions.

- A shard key join condition exsists. (e.g. t1.shardKeyCol = t2.shardKeyCol)
- Shard strategies are same.
  - Joining hash sharded table and hash sharded table
  - Joining range shard and range shard
  - Joining list shard and list shard

- Shard counts are same.
- Shard key column types are same.
- The number of shard key columns are same.

The following is an example of joining orders and lineitem. Both tables are hash sharded tables, and a sh ard key join condition exists. They are sharded for orderkey of the same standard, so the join is performed in each server.




```
\EXPLAIN PLAN
SELECT o_orderkey, l_partkey
  FROM orders, lineitem
 WHERE o_orderkey = l_orderkey;
```

```
>>>  start print plan
< Execution Plan >
================================================================================
¦IDX¦  NODE DESCRIPTION                                  ¦           ROWS ¦
--------------------------------------------------------------------------------
¦ 0 ¦  SELECT STATEMENT                                  ¦              6 ¦
¦ 1 ¦    QUERY BLOCK ("$QB_IDX_2")                       ¦              6 ¦
```

```
┆ 2 ┆      PLAN BASED CLUSTER                              ┆ LOCAL/REMOTE 6 ┆
┆ 3 ┆        HASH JOIN (INNER JOIN)                        ┆              2 ┆
┆ 4 ┆          INDEX ACCESS ("ORDERS", "ORDERS_PK_INDEX") ┆ (2)          2 ┆
┆ 5 ┆          HASH JOIN INSTANT                           ┆              2 ┆
┆ 6 ┆            TABLE ACCESS ("LINEITEM")                 ┆              2 ┆
================================================================================
     1  -  TARGET : ORDERS.O_ORDERKEY, LINEITEM.L_PARTKEY
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE USE_HASH_IN( _A1, 10 )
                            INDEX( _A2, "PUBLIC"."ORDERS_PK_INDEX" )
                            FULL( _A1 ) */
                       "_A2"."O_ORDERKEY", "_A1"."L_PARTKEY"
                  FROM ( "PUBLIC"."ORDERS"@LOCAL AS "_A2"
                         INNER JOIN
                         "PUBLIC"."LINEITEM"@LOCAL AS "_A1"
                    ON "_A1"."L_ORDERKEY" = "_A2"."O_ORDERKEY") ALIAS "_A3"
           TARGET DOMAIN : G1(G1N1,G1N2) 2 rows,
                           G2(G2N1,G2N2) 2 rows,
                           G3(G3N1,G3N2) 2 rows
     3  -  JOINED COLUMN : ORDERS.O_ORDERKEY, LINEITEM.L_PARTKEY
     4  -  HASH SHARD ( # 3 )
           READ INDEX COLUMN : ORDERS.O_ORDERKEY
     5  -  HASH KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : LINEITEM.L_PARTKEY
           READ KEY COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_PARTKEY
             HASH FILTER : LINEITEM.L_ORDERKEY = ORDERS.O_ORDERKEY
     6  -  HASH SHARD ( # 3 )
           READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_PARTKEY
<<<  end print plan
```

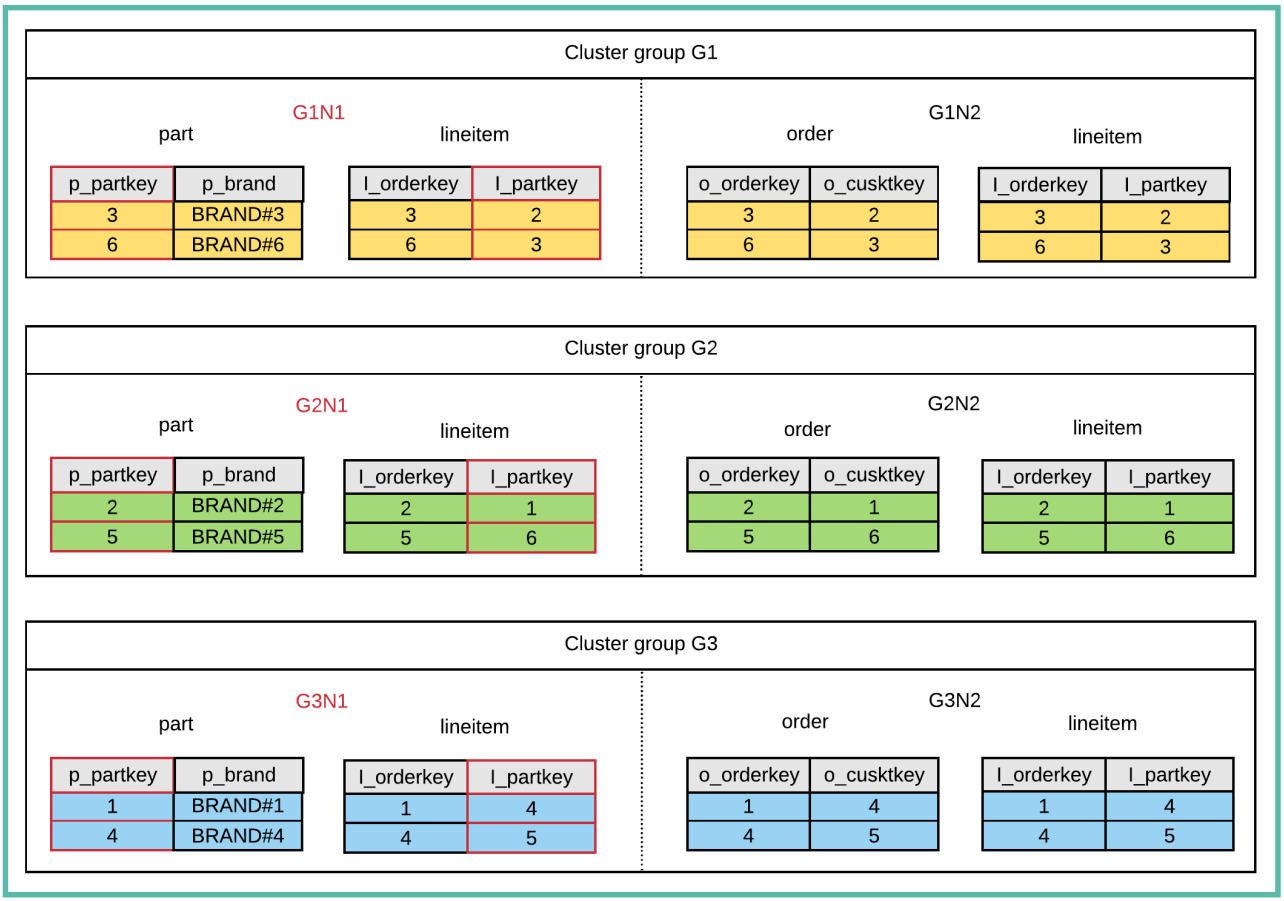
In the execution plan above, it performed join in each server and fetched the result by transferring join SQL to each server.

The following is an example of joining part and lineitem. Both tables are hash sharded tables. However, a shard key join condition does not exist.

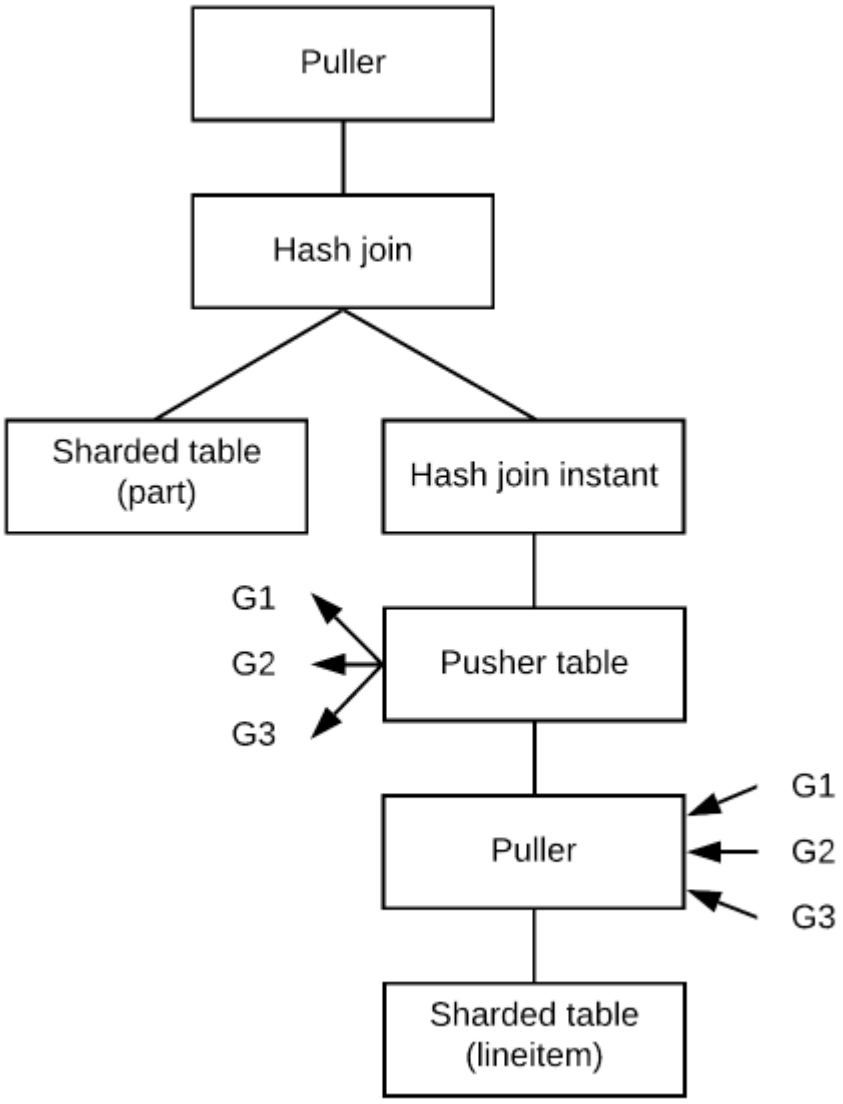
Both part and lineitem are hash sharded tables. However, data in part is sharded based on p_partkey because the shard key of part is p_partkey. On the other hand, data in lineitem is sharded based on l_orderkey because the shard key of lineitem is l_orderkey.

```
\EXPLAIN PLAN
SELECT /*+ REMOTE_JOIN(lineitem) */
       l_orderkey, p_partkey
```

```
 FROM part, lineitem
WHERE p_partkey = l_partkey;
```



For the query above to perform the remote join, it should fetch all data of lineitem, then shard them with l_partkey and transfer to G1, G2, G3 . In this case, a puller and a pusher take this role.



- **Cluster Puller**: It fetches data by transferring SQL each server.
- **Cluster Pusher**: It transfers data to each server.

In the figure above, a puller fetches all data from lineitem. Then, l_partkey makes a pusher table which is a shard key, and transfers data to G1, G2, G3. And it performs a remote join of part table and pusher tab le in G1, G2 and G3.

The execution plan is as follows.

```
>>>  start print plan
< Execution Plan >
================================================================================
¦IDX¦  NODE DESCRIPTION                                          ¦   ROWS ¦
--------------------------------------------------------------------------------
¦ 0 ¦  SELECT STATEMENT                                          ¦      6 ¦
```

```
| 1 |    QUERY BLOCK ("$QB_IDX_2")                                    |      6 |
| 2 |      SINGLE CLUSTER                                          |LOCAL/REMOTE 6 |
| 3 |        CLUSTER PUSHER ("_$NI_7")                                |      6 |
| 4 |          PLAN BASED CLUSTER                                  |LOCAL/REMOTE 6 |
| 5 |            TABLE ACCESS ("LINEITEM")                            |      2 |
| 6 |        SELECT STATEMENT                                         |      2 |
| 7 |          QUERY BLOCK ("$QB_IDX_2")                              |      2 |
| 8 |            HASH JOIN (INNER JOIN)                               |      2 |
| 9 |              INDEX ACCESS ("PART" AS _A2, "PART_PK_INDEX")   | (2)  2 |
| 10 |             HASH JOIN INSTANT                                   |      2 |
| 11 |               PUSHER TABLE ACCESS ("_$NI_7" AS _A1)          |     2 |
=========================================================================

     1  -  TARGET : _$NI_7.L_ORDERKEY, PART.P_PARTKEY
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE USE_HASH_IN( _A1, 10 )
                            INDEX( _A2, "PUBLIC"."PART_PK_INDEX" )
                            FULL( _A1 ) */
                        "_A1"."L_ORDERKEY", "_A2"."P_PARTKEY"
                   FROM ( "PUBLIC"."PART"@LOCAL AS "_A2"
                          INNER JOIN
                         "SESSION_SCHEMA"."_$NI_7"@LOCAL AS "_A1"
                     ON "_A1"."L_PARTKEY" = "_A2"."P_PARTKEY") ALIAS "_A3"
          TARGET DOMAIN : G1(G1N1,G1N2) 2 rows,
                          G2(G2N1,G2N2) 2 rows,
                          G3(G3N1,G3N2) 2 rows
     3  -  SQL : DECLARE INSTANT TABLE "SESSION_SCHEMA"."_$NI_7"
               ( "L_PARTKEY" NUMBER(10, 0), "L_ORDERKEY" NUMBER(10, 0) )
          COLUMN : LINEITEM.L_PARTKEY AS L_PARTKEY,
                   LINEITEM.L_ORDERKEY AS L_ORDERKEY
          SHARDED : LINEITEM.L_PARTKEY
          TARGET DOMAIN : G1(G1N1,G1N2) 2 rows,
                          G2(G2N1,G2N2) 2 rows,
                          G3(G3N1,G3N2) 2 rows
     4  -  SQL : SELECT /*+ FULL( _A1 ) */
                        "_A1"."L_ORDERKEY", "_A1"."L_PARTKEY"
                   FROM "PUBLIC"."LINEITEM"@LOCAL AS "_A1"
          TARGET DOMAIN : G1(G1N1,G1N2) 2 rows,
                          G2(G2N1,G2N2) 2 rows,
                          G3(G3N1,G3N2) 2 rows
     5  -  HASH SHARD ( # 3 )
          READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_PARTKEY
     7  -  TARGET : _A1.L_ORDERKEY, _A2.P_PARTKEY
```

```
     8  -  JOINED COLUMN : _A1.L_ORDERKEY, _A2.P_PARTKEY
     9  -  HASH SHARD ( # 3 )
           READ INDEX COLUMN : _A2.P_PARTKEY
    10  -  HASH KEY : _A1.L_PARTKEY
           RECORD COLUMN : _A1.L_ORDERKEY
           READ KEY COLUMN : _A1.L_PARTKEY, _A1.L_ORDERKEY
             HASH FILTER : _A1.L_PARTKEY = _A2.P_PARTKEY
    11  -  READ COLUMN : _A1.L_PARTKEY, _A1.L_ORDERKEY
<<<  end print plan
```

In the execution plan above, (l_orderkey, l_partkey) is fetched by transferring SQL to G1, G2, G3 in 4. And (l_partkey, l_orderkey) value fetched from 4 is stored in a pusher table whose shard key is (l_partkey, l_orderkey) in 3. This pusher table is sharded by l_partkey, then transferred and temporarily stored in G1, G2, G3.
A remote join is performed as joining part table and pusher table.

Group By

Local Group By

It performs *group by* in current server.

If it is group by for a cloned table, then every node in all groups have the same data, so just perform *group by* in the current server.

```
\EXPLAIN PLAN
  SELECT c_nationkey, COUNT(c_custkey)
    FROM customer
GROUP BY c_nationkey;
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                  ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                  ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                     ¦
¦    2  ¦      GROUP HASH INSTANT                                          ¦
¦    3  ¦        TABLE ACCESS ("CUSTOMER")                                 ¦
=============================================================================
     1  -  TARGET : CUSTOMER.C_NATIONKEY, COUNT( CUSTOMER.C_CUSTKEY )
     2  -  GROUP KEY : CUSTOMER.C_NATIONKEY
```

```
           RECORD COLUMN : COUNT( CUSTOMER.C_CUSTKEY )
           READ KEY COLUMN : CUSTOMER.C_NATIONKEY
           READ RECORD COLUMN : COUNT( CUSTOMER.C_CUSTKEY )
     3  -  CLONED
           READ COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_NATIONKEY
<<<  end print plan
```

Remote Group By

It performs *group by* in each server.

Group by processing in each server has a parallel processing effect. Also, generally *group by* reduces resul ts, so it reduces the network cost to bring the data.

```
\EXPLAIN PLAN
SELECT c_custkey, COUNT(o_orderkey)
  FROM customer, orders
 WHERE c_custkey = o_custkey
GROUP BY c_custkey;
>>>  start print plan
< Execution Plan >
================================================================================
¦IDX¦  NODE DESCRIPTION                                ¦           ROWS ¦
--------------------------------------------------------------------------------
¦ 0 ¦  SELECT STATEMENT                                ¦              6 ¦
¦ 1 ¦    QUERY BLOCK ("$QB_IDX_2")                     ¦              6 ¦
¦ 2 ¦     SINGLE CLUSTER                               ¦ LOCAL/REMOTE 6 ¦
¦ 3 ¦        SELECT STATEMENT                          ¦              2 ¦
¦ 4 ¦          QUERY BLOCK ("$QB_IDX_2")               ¦              2 ¦
¦ 5 ¦            GROUP HASH INSTANT                    ¦              2 ¦
¦ 6 ¦              HASH JOIN (INNER JOIN)              ¦              2 ¦
¦ 7 ¦                TABLE ACCESS ("CUSTOMER" AS _A2)  ¦              6 ¦
¦ 8 ¦                HASH JOIN INSTANT                 ¦              2 ¦
¦ 9 ¦                  TABLE ACCESS ("ORDERS" AS _A1)  ¦              2 ¦
================================================================================
     1  -  TARGET : CUSTOMER.C_CUSTKEY, COUNT( ORDERS.O_ORDERKEY )
     2  -  SQL : SELECT /*+ USE_GROUP_HASH(10)
                            KEEP_JOINED_TABLE USE_HASH_IN( _A1, 100 )
                            FULL( _A2 ) FULL( _A1 )
                        */
                        "_A2"."C_CUSTKEY", COUNT( "_A1"."O_ORDERKEY" )
                    FROM ( "PUBLIC"."CUSTOMER"@LOCAL AS "_A2"
```

```
                        INNER JOIN
                        "PUBLIC"."ORDERS"@LOCAL AS "_A1"
                    ON "_A1"."O_CUSTKEY" = "_A2"."C_CUSTKEY") ALIAS "_A3"
                GROUP BY "_A2"."C_CUSTKEY"
            TARGET DOMAIN : G1(G1N1,G1N2) 2 rows,
                            G2(G2N1,G2N2) 2 rows,
                            G3(G3N1,G3N2) 2 rows
            RE-GROUPING
              GROUP KEY : CUSTOMER.C_CUSTKEY
              AGGREGATION : SUM( COUNT( ORDERS.O_ORDERKEY ) )
     4  -  TARGET : _A2.C_CUSTKEY, COUNT( _A1.O_ORDERKEY )
     5  -  GROUP KEY : _A2.C_CUSTKEY
           RECORD COLUMN : COUNT( _A1.O_ORDERKEY )
           READ KEY COLUMN : _A2.C_CUSTKEY
           READ RECORD COLUMN : COUNT( _A1.O_ORDERKEY )
     6  -  JOINED COLUMN : _A2.C_CUSTKEY, _A1.O_ORDERKEY
     7  -  CLONED
           READ COLUMN : _A2.C_CUSTKEY
     8  -  HASH KEY : _A1.O_CUSTKEY
           RECORD COLUMN : _A1.O_ORDERKEY
           READ KEY COLUMN : _A1.O_CUSTKEY, _A1.O_ORDERKEY
             HASH FILTER : _A1.O_CUSTKEY = _A2.C_CUSTKEY
     9  -  HASH SHARD ( # 3 )
           READ COLUMN : _A1.O_ORDERKEY, _A1.O_CUSTKEY
<<<  end print plan
```

Distinct

Local Distinct

It performs distinct in current server.

The following is an example of using distinct in customer which is a cloned table. Every node in all groups in a cloned table has the same data, so just perform distinct in the current server.

```
\EXPLAIN PLAN
SELECT DISTINCT c_nationkey
  FROM customer;
>>>  start print plan
< Execution Plan >
```

```
================================================================================
|  IDX  |  NODE DESCRIPTION                                                    |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                    |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                         |
|    2  |      GROUP                                                           |
|    3  |        INDEX ACCESS ("CUSTOMER", "CUSTOMER_NATIONKEY_FK")            |
================================================================================
     1  -  TARGET : CUSTOMER.C_NATIONKEY
     2  -  GROUP KEY : CUSTOMER.C_NATIONKEY
     3  -  CLONED
           READ INDEX COLUMN : CUSTOMER.C_NATIONKEY
<<<  end print plan
```

The following is an example of using distinct in orders which is a sharded table. The data in a sharded tab le is sharded based in a shard key, so the data from all groups should be fetched when it is needed to loc ally perform distinct.

```
\EXPLAIN PLAN
SELECT /*+ LOCAL_DISTINCT */
       DISTINCT o_orderstatus
  FROM orders
 WHERE o_orderdate = date '1995-03-15';
>>>  start print plan
< Execution Plan >
================================================================================
|IDX|  NODE DESCRIPTION                              |             ROWS |
--------------------------------------------------------------------------------
| 0 |  SELECT STATEMENT                              |                3 |
| 1 |    QUERY BLOCK ("$QB_IDX_2")                   |                3 |
| 2 |      GROUP HASH INSTANT                        |                3 |
| 3 |        PLAN BASED CLUSTER                      | LOCAL/REMOTE 603 |
| 4 |          TABLE ACCESS ("ORDERS")               |              221 |
================================================================================
     1  -  TARGET : ORDERS.O_ORDERSTATUS
     2  -  GROUP KEY : ORDERS.O_ORDERSTATUS
           READ KEY COLUMN : ORDERS.O_ORDERSTATUS
     3  -  SQL : SELECT /*+ FULL( _A1 ) */
                         "_A1"."O_ORDERSTATUS"
                  FROM "PUBLIC"."ORDERS"@LOCAL AS "_A1"
                 WHERE "_A1"."O_ORDERDATE" = :_V0
```

```
          TARGET DOMAIN : G1(G1N1,G1N2) 221 rows,
                          G2(G2N1,G2N2) 184 rows,
                          G3(G3N1,G3N2) 198 rows
     4  -  HASH SHARD ( # 3 )
          READ COLUMN : ORDERS.O_ORDERSTATUS, ORDERS.O_ORDERDATE
            PHYSICAL FILTER : ORDERS.O_ORDERDATE = DATE'1995-03-15'
<<<  end print plan
```

In the execution plan above, GROUP HASH INSTANT for distinct was performed after fetching data satisfy ing the condition from G1, G2, G3.

Remote Distinct

It performs distinct in each server.

Distinct processing in each server has a parallel processing effect. Also, generally distinct reduces results, s o it reduces the network cost to bring the data.

```
\EXPLAIN PLAN
SELECT DISTINCT o_orderstatus
  FROM orders
 WHERE o_orderdate = date '1995-03-15';
>>>  start print plan
< Execution Plan >
================================================================================
¦IDX¦  NODE DESCRIPTION                                  ¦           ROWS ¦
--------------------------------------------------------------------------------
¦ 0 ¦  SELECT STATEMENT                                  ¦              3 ¦
¦ 1 ¦    QUERY BLOCK ("$QB_IDX_2")                       ¦              3 ¦
¦ 2 ¦      SINGLE CLUSTER                                ¦ LOCAL/REMOTE 3 ¦
¦ 3 ¦        SELECT STATEMENT                            ¦              3 ¦
¦ 4 ¦          QUERY BLOCK ("$QB_IDX_2")                 ¦              3 ¦
¦ 5 ¦            GROUP HASH INSTANT                      ¦              3 ¦
¦ 6 ¦              TABLE ACCESS ("ORDERS" AS _A1)        ¦            221 ¦
================================================================================
     1  -  TARGET : ORDERS.O_ORDERSTATUS
     2  -  SQL : SELECT /*+ USE_DISTINCT_HASH(3) FULL( _A1 ) */
                        DISTINCT "_A1"."O_ORDERSTATUS"
                   FROM "PUBLIC"."ORDERS"@LOCAL AS "_A1"
                  WHERE "_A1"."O_ORDERDATE" = :_V0
          TARGET DOMAIN : G1(G1N1,G1N2) 3 rows,
                          G2(G2N1,G2N2) 3 rows,
```

```
                        G3(G3N1,G3N2) 3 rows
          RE-GROUPING
            GROUP KEY : ORDERS.O_ORDERSTATUS
     4  -  TARGET : _A1.O_ORDERSTATUS
     5  -  GROUP KEY : _A1.O_ORDERSTATUS
           READ KEY COLUMN : _A1.O_ORDERSTATUS
     6  -  HASH SHARD ( # 3 )
           READ COLUMN : _A1.O_ORDERSTATUS, _A1.O_ORDERDATE
             PHYSICAL FILTER : _A1.O_ORDERDATE = :_V0
<<<  end print plan
```

In the execution plan above, It fetched results of performing distinct from G1, G2, G3, then performed di stinct again with RE-GROUPING. Though RE-GROUPING is performed, the results are decreased a lot so t he network cost is reduced and it is efficient.

Order By

Local Order By

It performs *order by* in the current server.

The following is an example of using *order by* in customer which is a cloned table. Every node in all group s in a cloned table has the same data, so just perform *order by* in the current server.

```
\EXPLAIN PLAN
  SELECT c_nationkey, COUNT(c_custkey)
    FROM customer
GROUP BY c_nationkey
ORDER BY c_nationkey;
>>>  start print plan
< Execution Plan >
=============================================================================
¦ IDX ¦  NODE DESCRIPTION                                    ¦   ROWS ¦
-----------------------------------------------------------------------------
¦  0  ¦  SELECT STATEMENT                                    ¦     25 ¦
¦  1  ¦    QUERY BLOCK ("$QB_IDX_2")                        ¦     25 ¦
¦  2  ¦      SORT INSTANT                                   ¦     25 ¦
¦  3  ¦        GROUP HASH INSTANT                           ¦     25 ¦
¦  4  ¦          TABLE ACCESS ("CUSTOMER")                  ¦ 150000 ¦
=============================================================================
     1  -  TARGET : CUSTOMER.C_NATIONKEY, COUNT( CUSTOMER.C_CUSTKEY )
```

```
     2  -  SORT KEY : "CUSTOMER.C_NATIONKEY ASC NULLS LAST"
           RECORD COLUMN : COUNT( CUSTOMER.C_CUSTKEY )
           READ KEY COLUMN : CUSTOMER.C_NATIONKEY
           READ RECORD COLUMN : COUNT( CUSTOMER.C_CUSTKEY )
     3  -  GROUP KEY : CUSTOMER.C_NATIONKEY
           RECORD COLUMN : COUNT( CUSTOMER.C_CUSTKEY )
           READ KEY COLUMN : CUSTOMER.C_NATIONKEY
           READ RECORD COLUMN : COUNT( CUSTOMER.C_CUSTKEY )
     4  -  CLONED
           READ COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_NATIONKEY
<<<  end print plan
```

The following is an example of using *order by* in orders which is a sharded table. The data in a sharded ta ble is sharded based in a shard key, so the data from all groups should be fetched when it is needed to lo cally perform *order by*.

```
\EXPLAIN PLAN
  SELECT /*+ LOCAL_ORDER */ o_orderdate, o_shippriority
    FROM orders
   WHERE o_orderdate >= date '1993-07-01'
     AND o_orderdate < date '1993-07-01' + interval '1' month
ORDER BY o_orderdate;
>>>  start print plan
< Execution Plan >
================================================================================
¦IDX¦  NODE DESCRIPTION                                ¦               ROWS ¦
--------------------------------------------------------------------------------
¦ 0 ¦  SELECT STATEMENT                                ¦              19319 ¦
¦ 1 ¦    QUERY BLOCK ("$QB_IDX_2")                     ¦              19319 ¦
¦ 2 ¦      SORT INSTANT                                ¦              19319 ¦
¦ 3 ¦        PLAN BASED CLUSTER                        ¦ LOCAL/REMOTE 19319 ¦
¦ 4 ¦          TABLE ACCESS ("ORDERS")                 ¦               6453 ¦
================================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_SHIPPRIORITY
     2  -  SORT KEY : "ORDERS.O_ORDERDATE ASC NULLS LAST"
           RECORD COLUMN : ORDERS.O_SHIPPRIORITY
           READ KEY COLUMN : ORDERS.O_ORDERDATE
           READ RECORD COLUMN : ORDERS.O_SHIPPRIORITY
     3  -  SQL : SELECT /*+ FULL( _A1 ) */
                        "_A1"."O_ORDERDATE", "_A1"."O_SHIPPRIORITY"
                   FROM "PUBLIC"."ORDERS"@LOCAL AS "_A1"
```

```
                WHERE "_A1"."O_ORDERDATE" < :_V0
                  AND "_A1"."O_ORDERDATE" >= :_V1
          TARGET DOMAIN : G1(G1N1,G1N2) 6453 rows,
                          G2(G2N1,G2N2) 6450 rows,
                          G3(G3N1,G3N2) 6416 rows
     4  -  HASH SHARD ( # 3 )
          READ COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_SHIPPRIORITY
            PHYSICAL FILTER : ORDERS.O_ORDERDATE < DATE'1993-07-01' + CAST( '1' AS
INTERVAL(MONTH) ) AND ORDERS.O_ORDERDATE >= DATE'1993-07-01'
<<<  end print plan
```

In the execution plan above, It fetched data satisfying the condition from G1, G2, G3, then performed S ORT INSTANT.

Remote Order By

It performs *order by* in each server.

Order by processing in each server has a parallel processing effect. Also, if results are sorted on the subor dinate node and ascends to the superordinate node, then each server does not need to separately proces s *order by* and the driver just merges and sorts the result received from each server.

The following is an example of remote order by. orders is a sharded table.

```
\EXPLAIN PLAN
  SELECT o_custkey, o_orderstatus
    FROM orders
   WHERE o_custkey > 0 AND o_custkey < 10
ORDER BY o_custkey;
>>>  start print plan
< Execution Plan >
================================================================================
¦IDX¦ NODE DESCRIPTION                                                  ¦ ROWS ¦
--------------------------------------------------------------------------------
¦ 0 ¦ SELECT STATEMENT                                                  ¦   66 ¦
¦ 1 ¦   QUERY BLOCK ("$QB_IDX_2")                                       ¦   66 ¦
¦ 2 ¦     MULTIPLE CLUSTER                                  ¦ LOCAL/REMOTE 66 ¦
¦ 3 ¦       SELECT STATEMENT                                            ¦   24 ¦
¦ 4 ¦         QUERY BLOCK ("$QB_IDX_2")                                 ¦   24 ¦
¦ 5 ¦           INDEX ACCESS ("ORDERS" AS _A1, "ORDERS_CUSTKEY_FK")     ¦   24 ¦
================================================================================
     1  -  TARGET : ORDERS.O_CUSTKEY, ORDERS.O_ORDERSTATUS
```

```
    2  -  SQL : SELECT /*+ INDEX( _A1, "PUBLIC"."ORDERS_CUSTKEY_FK" ) */
                       "_A1"."O_CUSTKEY", "_A1"."O_ORDERSTATUS"
                  FROM "PUBLIC"."ORDERS"@LOCAL AS "_A1"
                 WHERE "_A1"."O_CUSTKEY" > :_V0
                   AND "_A1"."O_CUSTKEY" < :_V1
              ORDER BY "_A1"."O_CUSTKEY" ASC NULLS LAST
          TARGET DOMAIN : G1(G1N1,G1N2) 24 rows,
                          G2(G2N1,G2N2) 23 rows,
                          G3(G3N1,G3N2) 19 rows
          MERGE SORTING
            SORT KEY : ORDERS.O_CUSTKEY
    4  -  TARGET : _A1.O_CUSTKEY, _A1.O_ORDERSTATUS
    5  -  HASH SHARD ( # 3 )
          READ INDEX COLUMN : _A1.O_CUSTKEY
          READ TABLE COLUMN : _A1.O_ORDERSTATUS
            MIN RANGE : _A1.O_CUSTKEY > :_V0
            MAX RANGE : _A1.O_CUSTKEY IS NOT NULL AND _A1.O_CUSTKEY < :_V1
<<<  end print plan
```

In the execution plan above, it fetched ordered data from G1, G2, G3, then merged data, and sorted for order by key col.

Aggregation

Local Aggregation

It performs aggregation in the current server.

The following is an example of a local aggregation. customer is a cloned table.

```
\EXPLAIN PLAN
SELECT COUNT(DISTINCT c_nationkey)
  FROM customer;
>>>  start print plan
< Execution Plan >
================================================================================
¦IDX¦  NODE DESCRIPTION                                          ¦   ROWS ¦
--------------------------------------------------------------------------------
¦ 0 ¦  SELECT STATEMENT                                          ¦      1 ¦
¦ 1 ¦    QUERY BLOCK ("$QB_IDX_2")                               ¦      1 ¦
¦ 2 ¦      AGGREGATION BY HASH                                   ¦      1 ¦
```

```
| 3 |        INDEX ACCESS ("CUSTOMER", "CUSTOMER_NATIONKEY_FK")   | 150000 |
================================================================================
     1  -  TARGET : COUNT( DISTINCT CUSTOMER.C_NATIONKEY )
     2  -  DISTINCT AGGREGATION : COUNT( DISTINCT CUSTOMER.C_NATIONKEY )
     3  -  CLONED
           READ INDEX COLUMN : CUSTOMER.C_NATIONKEY
<<<  end print plan
```

Remote Aggregation

It performs aggregation in each server.

Aggregation processing in each server has a parallel processing effect. Also, generally aggregation reduce s results, so it reduces the network cost to bring the data.

```
\EXPLAIN PLAN
SELECT sum(ps_supplycost * ps_availqty) * 0.0001
  FROM partsupp,
       supplier,
       nation
 WHERE ps_suppkey = s_suppkey
   AND s_nationkey = n_nationkey
   AND n_name = 'GERMANY';
>>>  start print plan
< Execution Plan >
================================================================================
|IDX|  NODE DESCRIPTION                                          |  ROWS |
--------------------------------------------------------------------------------
| 0 |  SELECT STATEMENT                                          |     1 |
| 1 |    QUERY BLOCK ("$QB_IDX_2")                               |     1 |
| 2 |      SINGLE CLUSTER                                 | LOCAL/REMOTE  1 |
| 3 |        SELECT STATEMENT                                    |     1 |
| 4 |          QUERY BLOCK ("$QB_IDX_2")                         |     1 |
| 5 |            AGGREGATION BY HASH                             |     1 |
| 6 |              NESTED JOIN (INNER JOIN)                      | 10508 |
| 7 |                NESTED JOIN (INNER JOIN)                    |   396 |
| 8 |                  TABLE ACCESS ("NATION" AS _A3)            |     1 |
| 9 |               INDEX ACCESS ("SUPPLIER" AS _A2)             |   396 |
| 10 |                INDEX ACCESS ("PARTSUPP" AS _A1)           | 10508 |
================================================================================
     1  -  TARGET : SUM( PARTSUPP.PS_SUPPLYCOST * PARTSUPP.PS_AVAILQTY ) * 0.0001
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE USE_NL_IN( _A1 )
```

```
                    USE_NL_IN( _A2 )
                    FULL( _A3 )
                    INDEX( _A2, "PUBLIC"."SUPPLIER_NATIONKEY_FK" )
                    INDEX( _A1, "PUBLIC"."PARTSUPP_SUPPKEY_FK" )
                 */
                 SUM( "_A1"."PS_SUPPLYCOST" * "_A1"."PS_AVAILQTY" )
           FROM ( ( "PUBLIC"."NATION"@LOCAL AS "_A3"
                    INNER JOIN
                    "PUBLIC"."SUPPLIER"@LOCAL AS "_A2"
                    ON true
                  ) ALIAS "_A4"
                 INNER JOIN
                 "PUBLIC"."PARTSUPP"@LOCAL AS "_A1" ON true
                ) ALIAS "_A5"
         WHERE "_A3"."N_NAME" = :_V0
           AND "_A2"."S_NATIONKEY" = "_A3"."N_NATIONKEY"
           AND "_A1"."PS_SUPPKEY" = "_A2"."S_SUPPKEY"
      TARGET DOMAIN : G1(G1N1,G1N2) 1 rows,
                      G2(G2N1,G2N2) 1 rows,
                      G3(G3N1,G3N2) 1 rows
      RE-AGGREGATION
        AGGREGATION : SUM( SUM( PARTSUPP.PS_SUPPLYCOST * PARTSUPP.PS_AVAILQTY ) )
 4  -  TARGET : SUM( _A1.PS_SUPPLYCOST * _A1.PS_AVAILQTY )
 5  -  AGGREGATION : SUM( _A1.PS_SUPPLYCOST * _A1.PS_AVAILQTY )
 6  -  JOINED COLUMN : _A1.PS_SUPPLYCOST, _A1.PS_AVAILQTY
        CONSTANT FILTER : TRUE
 7  -  JOINED COLUMN : _A2.S_SUPPKEY
        CONSTANT FILTER : TRUE
 8  -  CLONED
      READ COLUMN : _A3.N_NATIONKEY, _A3.N_NAME
        PHYSICAL FILTER : _A3.N_NAME = :_V0
 9  -  CLONED
      READ INDEX COLUMN : _A2.S_NATIONKEY
      READ TABLE COLUMN : _A2.S_SUPPKEY
        MIN RANGE : _A2.S_NATIONKEY = {_A3.N_NATIONKEY}
        MAX RANGE : _A2.S_NATIONKEY = {_A3.N_NATIONKEY}
10  -  HASH SHARD ( # 3 )
      READ INDEX COLUMN : _A1.PS_SUPPKEY
      READ TABLE COLUMN : _A1.PS_AVAILQTY, _A1.PS_SUPPLYCOST
        MIN RANGE : _A1.PS_SUPPKEY = {_A2.S_SUPPKEY}
        MAX RANGE : _A1.PS_SUPPKEY = {_A2.S_SUPPKEY}
```

```
<<<  end print plan
```

In the execution plan above, it fetched aggregation results from G1, G2, G3, then performed re-aggregat
ion and returned the result.

15.5 Statistics Information

A query optimizer calculates the cost by using the statistics information. The statistics information used by a query optimizer are table statistics information, column statistics information, index statistics informatio n.

- Table statistics information
  - The number of rows
  - The number of pages
- Column statistics information
  - The number of each different values
  - The number of NULL values
  - The average length of value
  - The minimum value
  - The maximum value
- Index statistics information
  - The number of each different keys
  - The number of pages
  - The number of leaf pages
  - The tree level
  - The clustering factor of the index

Perform **ANALYZE TABLE** statement to build the statistics information. The built statistics information is st ored in the database, and the same statistics information is used until it is rebuilt.

If the statistics information is not built for a table, then build a simple statistics information by using a cat alog information and the page information at the time of query execution and use it.

Adjusting Optimizer

Generally, a query optimizer select the most efficient plan by using the given statistics information. Howe ver, there could be a better plan than the plan selected by a query optimizer, so a user can specify to sele ct the plan when it was not selected by a query optimizer.

Currently, a query optimizer of GOLDILOCKS provides a hint, and the hint specified by a user is preferenti ally applied when it is applicable regardless of the calculated cost. Therefore, if the better plan exists, a us er can forcibly change the plan by using a hint.

For more information about hints, refer to **SQL Hint**.

15.6 SQL Hint

Description

A hint is a comment which is used by a user to directly instruct the GOLDILOCKS optimizer how to perform the SQL statement. The user uses the hint to select the execution plan when GOLDILOCKS optimizer can not select the appropriate execution plan due to an inaccurate statistics information.

GOLDILOCKS optimizer preferentially selects the user defined hints to when selecting the execution plan. However, if the hint specified by the user is not available, then GOLDILOCKS optimizer selects the plan by itself.

If a user specifies a hint, then GOLDILOCKS optimizer is operated as described in the hint as possible. Therefore, it is recommended to use the hint only when GOLDILOCKS optimizer determines that SQL execution plan is wrong.

Especially, if a hint is used when repeatedly using the same SQL statements, then GOLDILOCKS optimizer performs the SQL statement according to the described hint. In this case, the data in other tables or views including that SQL statement is updated, so the performance may be degraded.

The hint in GOLDILOCKS can be used in SELECT, INSERT SELECT, UPDATE, DELETE statements. The hint should be described next to the keyword of each statement between /*+ keyword and */ keywords.

Syntax

The following is a hint syntax.

```
<hint clause> ::=
    /*+ <hint element> [ comment ] [ [ , ] <hint element> [ comment ] ] */
<hint element> ::=
      <statement hints>
    ¦ <query block hints>
    ¦ <operation hints>
<statement hints> ::=
      <dml hints >
    ¦ <fetch fail over hints>
<dml hints> ::=
      DML_GLOBAL_ROWID
<fetch fail over hints> ::=
```

```
      FETCH_FAIL_OVER
<query block hints> ::=
      <push subquery hints>
    ¦ <cte query hints>
    ¦ <query transformation hints>
<push subquery hints> ::=
      PUSH_SUBQ
    ¦ NO_PUSH_SUBQ
<cte query hints> ::=
      INLINE
    ¦ MATERIALIZE
<query transformation hints> ::=
      NO_QUERY_TRANSFORMATION
    ¦ <unnest subquery hints>
    ¦ <transitive closure hints>
    ¦ <view hints>
<unnest subquery hints> ::=
      <unnest hints>
    ¦ <unnest join operation hints>
    ¦ <unnest join driver hints>
    ¦ <unnest join pusher hints>
    ¦ <unnest merge hints>
<unnest hints> ::=
      UNNEST
    ¦ NO_UNNEST
<unnest join operation hints> ::=
      UNNEST_NL
    ¦ UNNEST_NL_IN
    ¦ UNNEST_NL_OUT
    ¦ UNNEST_INL
    ¦ UNNEST_INL_IN
    ¦ UNNEST_INL_OUT
    ¦ UNNEST_HASH
    ¦ UNNEST_HASH( hash_bucket_count )
    ¦ UNNEST_HASH_IN
    ¦ UNNEST_HASH_IN( hash_bucket_count )
    ¦ UNNEST_HASH_OUT
    ¦ UNNEST_HASH_OUT( hash_bucket_count )
    ¦ UNNEST_MERGE
    ¦ UNNEST_MERGE_IN
    ¦ UNNEST_MERGE_OUT
```

```
    ¦ NL_SJ
    ¦ NL_ISJ
    ¦ NL_AJ
    ¦ INL_SJ
    ¦ INL_AJ
    ¦ MERGE_SJ
    ¦ MERGE_AJ
    ¦ HASH_SJ
    ¦ HASH_ISJ
    ¦ HASH_AJ
<unnest join driver hints> ::=
    ¦ LOCAL_UNNEST
    ¦ REMOTE_UNNEST
<unnest join pusher hints> ::=
      PUSHER_SUBQ
    ¦ NO_PUSHER_SUBQ
    ¦ PUSHER_OUTQ
    ¦ NO_PUSHER_OUTQ
<unnest merge hints> ::=
      MERGE_SUBQ
    ¦ NO_MERGE_SUBQ
<transitive closure hints> ::=
    ¦ TRANSITIVE_CLOSURE
    ¦ NO_TRANSITIVE_CLOSURE
< view hints > ::=
      <view merge hints>
    ¦ <push view predicate hints>
<view merge hints> ::=
    ¦ MERGE( view_name )
    ¦ NO_MERGE( view_name )
<push view predicate hints> ::=
      PUSH_PRED
    ¦ NO_PUSH_PRED
    ¦ PUSH_PRED( view_name[ [ , ] table_name ] )
    ¦ NO_PUSH_PRED( view_name[ [ , ] table_name ] )
<operation hints> ::=
      <access path hints>
    ¦ <join hints>
    ¦ <group hints>
    ¦ <distinct hints>
    ¦ <order hints>
```

```
    ¦ <aggr hints>
    ¦ <rownum hints>
<access path hints> ::=
      FULL( table_name )
    ¦ INDEX( table_name [ , ] [ index_name [ [ , ] index_name ] ] )
    ¦ NO_INDEX( table_name [ , ] [ index_name [ [ , ] index_name ] ] )
    ¦ INDEX_FORWARD( table_name [ , ] [ index_name [ [ , ] index_name ] ] )
    ¦ INDEX_BACKWARD( table_name [ , ] [ index_name [ [ , ] index_name ] ] )
    ¦ INDEX_ASC( table_name [ , ] [ index_name [ [ , ] index_name ] ] )
    ¦ INDEX_DESC( table_name [ , ] [ index_name [ [ , ] index_name ] ] )
    ¦ INDEX_COMBINE( table_name [ , ] [ index_name [ [ , ] index_name ] ] )
    ¦ IN_KEY_RANGE( table_name [ , ] [ index_name [ [ , ] index_name ] ] )
    ¦ ROWID( table_name )
<join hints> ::=
    < join order hints >
    ¦ < join operation hints >
    ¦ < join driver hints >
    ¦ < join pusher hints >
<join order hints> ::=
      ORDERED
    ¦ ORDERING( table_name [LEFT ¦ RIGHT] [ , table_name [LEFT ¦ RIGHT] ]
    ¦ LEADING( table_name [ [ , ] table_name ] )
    ¦ KEEP_JOINED_TABLE
<join operation hints> ::=
      USE_HASH( table_name [ [ , ] table_name ] )
    ¦ USE_HASH( table_name [ [ , ] table_name ], hash_bucket_count )
    ¦ NO_USE_HASH( table_name [ [ , ] table_name ] )
    ¦ USE_MERGE( table_name [ [ , ] table_name ] )
    ¦ NO_USE_MERGE( table_name [ [ , ] table_name ] )
    ¦ USE_NL( table_name [ [ , ] table_name ] )
    ¦ NO_USE_NL( table_name [ [ , ] table_name ] )
    ¦ USE_INL( table_name [ [ , ] table_name ] )
    ¦ NO_USE_INL( table_name [ [ , ] table_name ] )
    ¦ USE_JOIN_COMBINE( alias )
    ¦ NO_USE_JOIN_COMBINE( alias )
    ¦ USE_NL_IN( alias )
    ¦ USE_NL_OUT( alias )
    ¦ USE_INL_IN( alias )
    ¦ USE_INL_OUT( alias )
    ¦ USE_HASH_IN( alias )
    ¦ USE_HASH_IN( alias, hash_bucket_count )
```

```
    ¦ USE_HASH_OUT( alias )
    ¦ USE_HASH_OUT( alias, hash_bucket_count )
    ¦ USE_MERGE_IN( alias )
    ¦ USE_MERGE_OUT( alias )
<join driver hints> ::=
    ¦ LOCAL_JOIN( alias )
    ¦ REMOTE_JOIN( alias )
<join pusher hints> ::=
      PUSHER( alias )
    ¦ NO_PUSHER( alias )
<group hints> ::=
      <group operation hints>
    ¦ <group driver hints>
<group operation hints> ::=
      USE_GROUP_HASH
    ¦ USE_GROUP_HASH( hash_bucket_count )
<group driver hints> ::=
      LOCAL_GROUP
    ¦ REMOTE_GROUP
<distinct hints> ::=
      <distinct operation hints>
    ¦ <distinct driver hints>
<distinct operation hints> ::=
      DISTINCT_HASH
    ¦ USE_DISTINCT_HASH( hash_bucket_count )
<distinct driver hints> ::=
      LOCAL_DISTINCT
    ¦ REMOTE_DISTINCT
<order hints> ::=
      <order operation hints>
    ¦ <order driver hints>
<order operation hints> ::=
      USE_ORDER_SORT
    ¦ NO_USE_ORDER_SORT
    ¦ USE_ORDER_LIMIT_SORT
    ¦ NO_USE_ORDER_LIMIT_SORT
<order driver hints> ::=
      LOCAL_ORDER
    ¦ REMOTE_ORDER
<aggr hints> ::=
      <aggr driver hint>
```

```
<aggr driver hints> ::=
      LOCAL_AGGR
    ¦ REMOTE_AGGR
```

Invocation and Access Rules

The user should have the privilege to perform the query to perform <hint clause> statement.

Syntax Rules and Parameters

The followings are syntax rules to use <hint clause>.

- Multiple <hint element> can be described by using white space or comma (,) in <hint clause>.
- If there are two or more <hint element> and they can not be applied simultaneously, then only the pr eviously described <hint element> is applied.
- If a syntactic error occurs in <hint element>, then the <hint element> is ignored, and if hint_error pro perty is turned on, then it is processed as a validation error for <hint clause>.
- table_name and view_name described in <hint clause> should be matched with one of table_name, v iew_name described in <from clause> or aliases indicating them.
- table_name can not be described together with a schema name.
- Even when <hint element> is correctly described, but if it is not applicable, then the <hint element>is i gnored.

Examples

The following is an example of using <hint clause> in SELECT, INSERT SELECT, UPDATE, DELETE statemen t.

- Using <hint clause> in SELECT statement

```
SELECT /*+ INDEX(orders, o_orderdate_idx) */ *
  FROM orders
 WHERE o_orderdate < date '2019-04-12';
```

- Using <hint clause> in INSERT SELECT statement

```
INSERT INTO orders_bk SELECT /*+ INDEX(orders, o_orderdate_idx) */ *
                        FROM orders
                       WHERE o_orderdate < date '2019-04-12';
```

- Using <hint clause> in UPDATE statement

```
UPDATE /*+ INDEX(lineitem, l_shipdate_idx) */ * lineitem
   SET l_receiptdate = CURRENT_DATE
 WHERE l_shipdate = date '2020-04-12';
```

- Using <hint clause> in DELETE statement

```
DELETE /*+ INDEX(lineitem, l_shipdate_idx) */ * lineitem
 WHERE l_receiptdate < date '2020-04-12';
```

The following is an example of incorrectly using a hint after setting HINT_ERROR property to *ON* by using ALTER statement.

```
ALTER SESSION SET HINT_ERROR = ON;
SELECT /*+ INDEX(orders, o_orderdate_idxx) */ *
  FROM orders
 WHERE o_orderdate < date '2019-04-12';
ERR-42000(16058): not applicable hint ( cannot find index ) :
SELECT /*+ INDEX(orders, o_orderdate_idxx) */ *
                          *
ERROR at line 1:
```

For more information about descriptions and examples of all hints, refer to **Query Block Hint** and **Operation Hint**.

Statement Hint

Those hints are applied per each statement.

<dml hints>

DML_GLOBAL_ROWID

When processing DML statement in cluster environment, a node received a user query transfers the updated information of the record to another node, then applies it.

DML statement in cluster environment is processed with the following two methods.

- Query based DML: It creates a new query to transfer the updated statement.
- Global rowid based DML: It uses the record identifier information to transfer the updated information of a specific record.

If DML_GLOBAL_ROWID hint is specified, then the global rowid based DML method is preferentially appli ed.
This hint is applied to SELECT FOR UPDATE, INSERT, UPDATE, DELETE statements.

The following is an example of applying the query based DML.

```
\EXPLAIN PLAN
DELETE FROM orders;
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                  ¦
-----------------------------------------------------------------------------
¦    0  ¦  DELETE STATEMENT ("ORDERS")                                      ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                    ¦
¦    2  ¦      DML CLUSTER                                          ¦
¦    3  ¦        TABLE ACCESS ("ORDERS")                                  ¦
=============================================================================
     1  -  TARGET : NOTHING
     2  -  WITHOUT FETCH
           Non-Fetch SQL : DELETE /*+ FULL( _A1 ) */  "_A1" FROM "PUBLIC"."ORDERS"@LOCAL AS
"_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 0 rows, G2(G2N1,G2N2) 0 rows, G3(G3N1,G3N2) 0 rows
     3  -  HASH SHARD ( # 3 )
           READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_CUSTKEY
<<<  end print plan
```

The following is an example of using DML_GLOBAL_ROWID hint.

```
\EXPLAIN PLAN
DELETE /*+ DML_GLOBAL_ROWID */ FROM orders;
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                  ¦
-----------------------------------------------------------------------------
¦    0  ¦  DELETE STATEMENT ("ORDERS")                                      ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                    ¦
¦    2  ¦      PLAN BASED CLUSTER                                         ¦
¦    3  ¦        TABLE ACCESS ("ORDERS")                                  ¦
=============================================================================
     1  -  TARGET : NOTHING
```

```
     2  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."$PHYSICAL_ROWID", "_A1"."O_ORDERKEY",
"_A1"."O_CUSTKEY" FROM "PUBLIC"."ORDERS"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 0 rows, G2(G2N1,G2N2) 0 rows, G3(G3N1,G3N2) 0 rows
     3  -  HASH SHARD ( # 3 )
           READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_CUSTKEY
<<<  end print plan
```

〈fetch fail over hints〉

FETCH_FAIL_OVER

The fetch fail over feature provides an intact fetch result by considering communication error situation w hile performing fetch for SELECT statement in cluster environment.
Using FETCH_FAIL_OVER hint enables the fetch fail over feature.
This hint is applied to SELECT statement.

The following is an example of using FETCH_FAIL_OVER hint.

```
\EXPLAIN PLAN
SELECT /*+ FETCH_FAIL_OVER */ o_orderkey FROM orders;
>>>  start print plan
< Execution Plan >
=========================================================================
¦  IDX  ¦  NODE DESCRIPTION                                              ¦
-------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                              ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                  ¦
¦    2  ¦      PLAN BASED CLUSTER                                       ¦
¦    3  ¦        INDEX ACCESS ("ORDERS", "ORDERS_PK_INDEX")             ¦
=========================================================================
     0  -  FETCH FAIL OVER
     1  -  TARGET : ORDERS.O_ORDERKEY
     2  -  SQL : SELECT /*+ INDEX( _A1, "PUBLIC"."ORDERS_PK_INDEX" ) */ "_A1"."O_ORDERKEY"
FROM "PUBLIC"."ORDERS"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 0 rows, G2(G2N1,G2N2) 0 rows, G3(G3N1,G3N2) 0 rows
     3  -  HASH SHARD ( # 3 )
           READ INDEX COLUMN : ORDERS.O_ORDERKEY
<<<  end print plan
```

Query Block Hint

Those hints are applied per each query block.

〈push subquery hints〉

PUSH_SUBQ

If PUSH_SUBQ hint is specified, then an optimizer pushes the subquery to the lowest node of available ex ecution plan nodes. It is a hint for a subquery which is not unnested, so PUSH_SUBQ hint is ignored when 〈unnest subquery hints〉 is specified beforehand.

If PUSH_SUBQ hint is not specified, then an optimizer calculates a cost, then pushes the subquery to the node whose cost is the best.

If PUSH_SUBQ hint is used, the subquery is applied unnested as soon as possible. Therefore, the performa nce is upgraded, when subquery filtering has a large effect and the intermediate result can be stored by p erforming the subquery only once.
The following is an example of using PUSH_SUBQ hint in this case.

```
\EXPLAIN PLAN
SELECT
       c_name,
       o_totalprice
  FROM
       customer,
       orders,
       lineitem
 WHERE c_custkey = o_custkey
   AND o_orderkey = l_orderkey
   AND o_orderkey IN (
                       SELECT   /*+ PUSH_SUBQ */
                                l_orderkey
                         FROM   lineitem
                       GROUP BY l_orderkey
                       HAVING   sum(l_quantity) > 300
                     );
>>>  start print plan
< Execution Plan >
=========================================================================
¦  IDX  ¦  NODE DESCRIPTION                                              ¦
```

```
-------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                              ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                  ¦
¦    2  ¦      NESTED JOIN (INNER JOIN)                                 ¦
¦    3  ¦        MERGE JOIN (INNER JOIN)                                ¦
¦    4  ¦          INDEX ACCESS ("ORDERS", "ORDERS_PK_INDEX")           ¦
¦    5  ¦          INDEX ACCESS ("LINEITEM", "LINEITEM_PK_INDEX")       ¦
¦    6  ¦        INDEX ACCESS ("CUSTOMER", "CUSTOMER_PK_INDEX")         ¦
¦    7  ¦  SUB QUERY LIST                                               ¦
¦    8  ¦    INLINE_VIEW ("$V8") (MATERIALIZED)                         ¦
¦    9  ¦      QUERY BLOCK ("$QB_IDX_10")                               ¦
¦   10  ¦        GROUP HASH INSTANT                                     ¦
¦   11  ¦          TABLE ACCESS ("LINEITEM")                            ¦
=========================================================================
     1  -  TARGET : CUSTOMER.C_NAME, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : CUSTOMER.C_NAME, ORDERS.O_TOTALPRICE
     3  -  JOINED COLUMN : ORDERS.O_CUSTKEY, ORDERS.O_TOTALPRICE
           ON FILTER (Equi) : ORDERS.O_ORDERKEY = LINEITEM.L_ORDERKEY
     4  -  READ INDEX COLUMN : ORDERS.O_ORDERKEY
           READ TABLE COLUMN : ORDERS.O_CUSTKEY, ORDERS.O_TOTALPRICE
           POST FILTER : ( ORDERS.O_ORDERKEY ) IN ( $V8.L_ORDERKEY )
     5  -  READ INDEX COLUMN : LINEITEM.L_ORDERKEY
           MIN RANGE : LINEITEM.L_ORDERKEY >= {ORDERS.O_ORDERKEY}
           MAX RANGE : LINEITEM.L_ORDERKEY IS NOT NULL
     6  -  READ INDEX COLUMN : CUSTOMER.C_CUSTKEY
           READ TABLE COLUMN : CUSTOMER.C_NAME
           MIN RANGE : CUSTOMER.C_CUSTKEY = {ORDERS.O_CUSTKEY}
           MAX RANGE : CUSTOMER.C_CUSTKEY = {ORDERS.O_CUSTKEY}
           FETCH ONE ROW
     8  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     9  -  TARGET : LINEITEM.L_ORDERKEY
    10  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
    11  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
<<<  end print plan
```

The subquery in the example above can be performed either in join or on orders, and it is performed on *INDEX ACCESS ("ORDERS")* which is the lowest node of available execution nodes, and it is not unnested

either.

NO_PUSH_SUBQ

If NO_PUSH_SUBQ hint is specified, then an optimizer does not push the subquery. Therefore, the subquery is performed on the top node of available execution plan nodes. It is a hint for a subquery which is not unnested, so NO_PUSH_SUBQ hint is ignored when <unnest subquery hints> is specified beforehand.

If NO_PUSH_SUBQ hint is not specified, then an optimizer calculates a cost, then pushes the subquery to the node whose cost is the best.

If NO_PUSH_SUBQ hint is specified, the subquery is applied unnested as late as possible. Therefore, if the subquery is applied before the intermediate result is reduced by the join when the subquery filtering has a small effect and the subquery is repeatedly performed, then the performance gets slow because the subquery is repeatedly performed. In this case, it is recommended to delay the time of applying the subquery as long as possible by using NO_PUSH_SUBQ hint.

The following is an example of using NO_PUSH_SUBQ hint in this case.

```
\EXPLAIN PLAN
SELECT
       c_name,
       o_totalprice
  FROM
       customer,
       orders,
       lineitem
 WHERE c_custkey = o_custkey
   AND o_orderkey = l_orderkey
   AND o_orderkey IN (
                       SELECT   /*+ NO_PUSH_SUBQ */
                              l_orderkey
                         FROM  lineitem
                        WHERE  o_orderdate = l_shipdate
                      );
>>>  start print plan
< Execution Plan >
============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                 ¦
----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                    ¦
¦    2  ¦      NESTED JOIN (INNER JOIN)                                   ¦
```

```
¦    3  ¦        MERGE JOIN (INNER JOIN)                              ¦
¦    4  ¦          INDEX ACCESS ("ORDERS", "ORDERS_PK_INDEX")         ¦
¦    5  ¦          INDEX ACCESS ("LINEITEM", "LINEITEM_PK_INDEX")     ¦
¦    6  ¦        INDEX ACCESS ("CUSTOMER", "CUSTOMER_PK_INDEX")       ¦
¦    7  ¦      SUB QUERY LIST                                         ¦
¦    8  ¦        INLINE_VIEW ("$V8")                                  ¦
¦    9  ¦          QUERY BLOCK ("$QB_IDX_10")                         ¦
¦   10  ¦            TABLE ACCESS ("LINEITEM")                        ¦
========================================================================
     1  -  TARGET : CUSTOMER.C_NAME, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_ORDERKEY, CUSTOMER.C_NAME,
ORDERS.O_TOTALPRICE
             POST WHERE FILTER : ( ORDERS.O_ORDERKEY ) IN ( $V8.L_ORDERKEY )
     3  -  JOINED COLUMN : ORDERS.O_CUSTKEY, ORDERS.O_ORDERDATE, ORDERS.O_ORDERKEY,
ORDERS.O_TOTALPRICE
             ON FILTER (Equi) : ORDERS.O_ORDERKEY = LINEITEM.L_ORDERKEY
     4  -  READ INDEX COLUMN : ORDERS.O_ORDERKEY
           READ TABLE COLUMN : ORDERS.O_CUSTKEY, ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
     5  -  READ INDEX COLUMN : LINEITEM.L_ORDERKEY
             MIN RANGE : LINEITEM.L_ORDERKEY >= {ORDERS.O_ORDERKEY}
             MAX RANGE : LINEITEM.L_ORDERKEY IS NOT NULL
     6  -  READ INDEX COLUMN : CUSTOMER.C_CUSTKEY
           READ TABLE COLUMN : CUSTOMER.C_NAME
             MIN RANGE : CUSTOMER.C_CUSTKEY = {ORDERS.O_CUSTKEY}
             MAX RANGE : CUSTOMER.C_CUSTKEY = {ORDERS.O_CUSTKEY}
           FETCH ONE ROW
     8  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     9  -  TARGET : LINEITEM.L_ORDERKEY
    10  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_SHIPDATE
             PHYSICAL FILTER : LINEITEM.L_SHIPDATE = {ORDERS.O_ORDERDATE}
```

The subquery in the example above can be performed either in nested join, merge join or orders index ac cess, and it is performed on NESTED JOIN (INNER JOIN) which is the top node of available execution node s.

〈 cte query hints 〉

This hint is for the query block of the common table expression.

The common table expression is performed either in inline method or in materialize method.

- Inline method: It is performed in the form of inline view.

- Materialize method: It stores the result of the common table expression in an instant, then perform t he query by reading rows from the instant table.

Generally, in case of when using the common table expression only once, then it is performed in inline m ethod. In case of when using the common table expression two or more times, then it is performed in ma terialize method.

INLINE

INLINE hint is valid only in a query block of the common table expression, and if this hint is specified, then the common table expression is performed like as inline view.

The following is an example of using INLINE hint.

```
gSQL> \explain plan
WITH revenue ( supplier_no, total_revenue ) AS
     (
        SELECT /*+ INLINE */
               l_suppkey,
               SUM(l_extendedprice * (1 - l_discount))
          FROM lineitem
         WHERE l_shipdate >= DATE '1996-01-01'
           AND l_shipdate < DATE '1996-01-01' + INTERVAL '3' MONTH
         GROUP BY
               l_suppkey
     )
SELECT
    s_suppkey,
    s_name,
    ROUND( total_revenue, 2 ) as total_revenue
FROM
    supplier,
    revenue
WHERE
      s_suppkey = supplier_no
  AND total_revenue = (
                        select
                            max(total_revenue)
                        from
                            revenue
                      )
ORDER BY
   s_suppkey;
```

```
S_SUPPKEY S_NAME                    TOTAL_REVENUE
--------- ------------------------- -------------
     8449 Supplier#000008449           1772627.21
1 row selected.
>>>  start print plan
< Execution Plan >
============================================================================
|  IDX  |  NODE DESCRIPTION                                                |
----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                     |
|    2  |      SORT INSTANT                                                |
|    3  |        NESTED JOIN (INNER JOIN)                                  |
|    4  |          VIEW ("REVENUE")                                        |
|    5  |            QUERY BLOCK ("$QB_IDX_7")                             |
|    6  |              GROUP HASH INSTANT                                  |
|    7  |                TABLE ACCESS ("LINEITEM")                         |
|    8  |          INDEX ACCESS ("SUPPLIER", "SUPPLIER_PK_INDEX")          |
|    9  |  SUB QUERY LIST                                                  |
|   10  |    INLINE_VIEW ("$V10")                                          |
|   11  |      QUERY BLOCK ("$QB_IDX_14")                                  |
|   12  |        AGGREGATION BY HASH                                       |
|   13  |          VIEW ("REVENUE")                                        |
|   14  |            QUERY BLOCK ("$QB_IDX_17")                            |
|   15  |              GROUP HASH INSTANT                                  |
|   16  |                TABLE ACCESS ("LINEITEM")                         |
============================================================================
     1  -  TARGET : SUPPLIER.S_SUPPKEY, SUPPLIER.S_NAME, ROUND(REVENUE.TOTAL_REVENUE,2) AS
TOTAL_REVENUE
     2  -  SORT KEY : "SUPPLIER.S_SUPPKEY ASC NULLS LAST"
           RECORD COLUMN : SUPPLIER.S_NAME, ROUND(REVENUE.TOTAL_REVENUE,2)
           READ KEY COLUMN : SUPPLIER.S_SUPPKEY
           READ RECORD COLUMN : SUPPLIER.S_NAME, ROUND(REVENUE.TOTAL_REVENUE,2)
     3  -  JOINED COLUMN : SUPPLIER.S_SUPPKEY, SUPPLIER.S_NAME, REVENUE.TOTAL_REVENUE
     4  -  COLUMN : LINEITEM.L_SUPPKEY AS SUPPLIER_NO, SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 -
LINEITEM.L_DISCOUNT ) ) AS TOTAL_REVENUE
     5  -  TARGET : LINEITEM.L_SUPPKEY, SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 -
LINEITEM.L_DISCOUNT ) )
     6  -  GROUP KEY : LINEITEM.L_SUPPKEY
           RECORD COLUMN : SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 - LINEITEM.L_DISCOUNT ) )
           READ KEY COLUMN : LINEITEM.L_SUPPKEY
```

```
           READ RECORD COLUMN : SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 - LINEITEM.L_DISCOUNT ) )
             PHYSICAL FILTER : SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 - LINEITEM.L_DISCOUNT ) ) =
$V10.$C0
     7  -  READ COLUMN : LINEITEM.L_SUPPKEY, LINEITEM.L_EXTENDEDPRICE, LINEITEM.L_DISCOUNT,
LINEITEM.L_SHIPDATE
             PHYSICAL FILTER : LINEITEM.L_SHIPDATE < DATE'1996-01-01' + CAST( '3' AS
INTERVAL(MONTH) ) AND LINEITEM.L_SHIPDATE >= DATE'1996-01-01'
     8  -  READ INDEX COLUMN : SUPPLIER.S_SUPPKEY
           READ TABLE COLUMN : SUPPLIER.S_NAME
             MIN RANGE : SUPPLIER.S_SUPPKEY = {REVENUE.SUPPLIER_NO}
             MAX RANGE : SUPPLIER.S_SUPPKEY = {REVENUE.SUPPLIER_NO}
           FETCH ONE ROW
    10  -  COLUMN : MAX( REVENUE.TOTAL_REVENUE ) AS $C0
    11  -  TARGET : MAX( REVENUE.TOTAL_REVENUE )
    12  -  AGGREGATION : MAX( REVENUE.TOTAL_REVENUE )
    13  -  COLUMN : SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 - LINEITEM.L_DISCOUNT ) ) AS
TOTAL_REVENUE
    14  -  TARGET : SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 - LINEITEM.L_DISCOUNT ) )
    15  -  GROUP KEY : LINEITEM.L_SUPPKEY
           RECORD COLUMN : SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 - LINEITEM.L_DISCOUNT ) )
           READ RECORD COLUMN : SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 - LINEITEM.L_DISCOUNT ) )
    16  -  READ COLUMN : LINEITEM.L_SUPPKEY, LINEITEM.L_EXTENDEDPRICE, LINEITEM.L_DISCOUNT,
LINEITEM.L_SHIPDATE
             PHYSICAL FILTER : LINEITEM.L_SHIPDATE < DATE'1996-01-01' + CAST( '3' AS
INTERVAL(MONTH) ) AND LINEITEM.L_SHIPDATE >= DATE'1996-01-01'
<<<  end print plan
```

In the query above, revenue defined in WITH clause is used twice in SELECT query.
In this case, it is recommended to process it in materialize method not to perform revenue multiple times. However, if the cost of materialize is bigger than that of performing it twice, then use INLINE hint to prev ent it performing in materialize method.

MATERIALIZE

MATERIALIZE hint is valid only in a query block of the common table expression, and if this hint is specifie d, then the common table expression is performed once and the result is stored in an instant table.

The following is an example of using MATERIALIZE hint.

```
gSQL> \explain plan
WITH revenue ( supplier_no, total_revenue ) AS
     (
```

```
        SELECT /*+ MATERIALIZE */
               l_suppkey,
               SUM(l_extendedprice * (1 - l_discount))
          FROM lineitem
         WHERE l_shipdate >= DATE '1996-01-01'
           AND l_shipdate < DATE '1996-01-01' + INTERVAL '3' MONTH
         GROUP BY
               l_suppkey
     )
SELECT
    s_suppkey,
    s_name,
    ROUND( total_revenue, 2 ) as total_revenue
FROM
    supplier,
    revenue
WHERE
      s_suppkey = supplier_no
  AND total_revenue = (
                        select
                            max(total_revenue)
                        from
                            revenue
                       )
ORDER BY
   s_suppkey;
S_SUPPKEY S_NAME                    TOTAL_REVENUE
--------- ------------------------- -------------
     8449 Supplier#000008449           1772627.21
1 row selected.
>>>  start print plan
< Execution Plan >
============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                 ¦
----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                    ¦
¦    2  ¦      SORT INSTANT                                               ¦
¦    3  ¦        NESTED JOIN (INNER JOIN)                                 ¦
¦    4  ¦          MTR ACCESS ("REVENUE")                                 ¦
¦    5  ¦          INDEX ACCESS ("SUPPLIER", "SUPPLIER_PK_INDEX")         ¦
```

```
┆    6  ┆    MTR LOADER LIST                                          ┆
┆    7  ┆      MTR LOADER ("REVENUE")                                 ┆
┆    8  ┆        VIEW ("REVENUE")                                     ┆
┆    9  ┆          QUERY BLOCK ("$QB_IDX_7")                          ┆
┆   10  ┆            GROUP HASH INSTANT                               ┆
┆   11  ┆              TABLE ACCESS ("LINEITEM")                      ┆
┆   12  ┆  SUB QUERY LIST                                             ┆
┆   13  ┆    INLINE_VIEW ("$V12")                                     ┆
┆   14  ┆      QUERY BLOCK ("$QB_IDX_14")                             ┆
┆   15  ┆        MTR ACCESS ("REVENUE")                               ┆
=================================================================================
     1  -  TARGET : SUPPLIER.S_SUPPKEY,
                    SUPPLIER.S_NAME,
                    ROUND(REVENUE.TOTAL_REVENUE,2) AS TOTAL_REVENUE
     2  -  SORT KEY : "SUPPLIER.S_SUPPKEY ASC NULLS LAST"
           RECORD COLUMN : SUPPLIER.S_NAME, ROUND(REVENUE.TOTAL_REVENUE,2)
           READ KEY COLUMN : SUPPLIER.S_SUPPKEY
           READ RECORD COLUMN : SUPPLIER.S_NAME,
                                ROUND(REVENUE.TOTAL_REVENUE,2)
     3  -  JOINED COLUMN : SUPPLIER.S_SUPPKEY,
                           SUPPLIER.S_NAME,
                           REVENUE.TOTAL_REVENUE
     4  -  READ COLUMN : REVENUE.SUPPLIER_NO, REVENUE.TOTAL_REVENUE
             PHYSICAL FILTER : REVENUE.TOTAL_REVENUE = $V12.$C0
     5  -  READ INDEX COLUMN : SUPPLIER.S_SUPPKEY
           READ TABLE COLUMN : SUPPLIER.S_NAME
             MIN RANGE : SUPPLIER.S_SUPPKEY = {REVENUE.SUPPLIER_NO}
             MAX RANGE : SUPPLIER.S_SUPPKEY = {REVENUE.SUPPLIER_NO}
           FETCH ONE ROW
     7  -  RECORD COLUMN : REVENUE.SUPPLIER_NO, REVENUE.TOTAL_REVENUE
     8  -  COLUMN : LINEITEM.L_SUPPKEY AS SUPPLIER_NO,
                    SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 - LINEITEM.L_DISCOUNT ) ) AS
TOTAL_REVENUE
     9  -  TARGET : LINEITEM.L_SUPPKEY,
                   SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 - LINEITEM.L_DISCOUNT ) )
    10  -  GROUP KEY : LINEITEM.L_SUPPKEY
           RECORD COLUMN : SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 - LINEITEM.L_DISCOUNT ) )
           READ KEY COLUMN : LINEITEM.L_SUPPKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 - LINEITEM.L_DISCOUNT ) )
    11  -  READ COLUMN : LINEITEM.L_SUPPKEY,
                         LINEITEM.L_EXTENDEDPRICE,
```

```
                        LINEITEM.L_DISCOUNT, LINEITEM.L_SHIPDATE
        PHYSICAL FILTER : LINEITEM.L_SHIPDATE < DATE'1996-01-01' + CAST( '3' AS
INTERVAL(MONTH) ) AND LINEITEM.L_SHIPDATE >= DATE'1996-01-01'
    13  -  COLUMN : MAX( REVENUE.TOTAL_REVENUE ) AS $C0
    14  -  TARGET : MAX( REVENUE.TOTAL_REVENUE )
    15  -  READ COLUMN : REVENUE.TOTAL_REVENUE
           AGGREGATION : MAX( REVENUE.TOTAL_REVENUE )
<<<  end print plan
```

In the query above, revenue defined in WITH clause is used twice in SELECT query.
When performing it in materialize method, then it performs the common table expression only once and stores the result in an instant table, then uses it, so the operating time is reduced.

NO_QUERY_TRANSFORMATION

If NO_QUERY_TRANSFORMATION hint is specified, then the query block in which the hint is specified an d all query blocks below it do not perform the query transformation.

The following is an example of using NO_QUERY_TRANSFORMATION hint.

```
\EXPLAIN PLAN
SELECT /*+ NO_QUERY_TRANSFORMATION */
       s_name,
       l_quantity
  FROM supplier,
       (
        SELECT
               l_suppkey,
               l_quantity
          FROM lineitem
         WHERE l_shipdate >= date '1996-01-01'
           AND l_shipdate < date '1996-01-01' + interval '3' month
       ) lineitem_view
 WHERE s_suppkey = l_suppkey
   AND s_nationkey IN (
                        SELECT n_nationkey
                          FROM nation
                         WHERE n_name = 'FRANCE' OR n_name = 'GERMANY'
                       )
;
>>>  start print plan
< Execution Plan >
```

```
=========================================================================
¦  IDX  ¦  NODE DESCRIPTION                                              ¦
-------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                              ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                 ¦
¦    2  ¦      NESTED JOIN (INNER JOIN)                                ¦
¦    3  ¦        INLINE_VIEW ("LINEITEM_VIEW")                         ¦
¦    4  ¦          QUERY BLOCK ("$QB_IDX_7")                           ¦
¦    5  ¦            TABLE ACCESS ("LINEITEM")                         ¦
¦    6  ¦        INDEX ACCESS ("SUPPLIER", "SUPPLIER_PK_INDEX")        ¦
¦    7  ¦  SUB QUERY LIST                                              ¦
¦    8  ¦    INLINE_VIEW ("$V8") (MATERIALIZED)                        ¦
¦    9  ¦      QUERY BLOCK ("$QB_IDX_12")                              ¦
¦   10  ¦        TABLE ACCESS ("NATION")                               ¦
=========================================================================
     1  -  TARGET : SUPPLIER.S_NAME, LINEITEM_VIEW.L_QUANTITY
     2  -  JOINED COLUMN : SUPPLIER.S_NATIONKEY, SUPPLIER.S_NAME, LINEITEM_VIEW.L_QUANTITY
             POST WHERE FILTER : ( SUPPLIER.S_NATIONKEY ) IN ( $V8.N_NATIONKEY )
     3  -  COLUMN : LINEITEM.L_SUPPKEY AS L_SUPPKEY, LINEITEM.L_QUANTITY AS L_QUANTITY
     4  -  TARGET : LINEITEM.L_SUPPKEY, LINEITEM.L_QUANTITY
     5  -  READ COLUMN : LINEITEM.L_SUPPKEY, LINEITEM.L_QUANTITY, LINEITEM.L_SHIPDATE
             PHYSICAL FILTER : LINEITEM.L_SHIPDATE >= DATE'1996-01-01' AND LINEITEM.L_SHIPDATE
< DATE'1996-01-01' + CAST( '3' AS INTERVAL(MONTH) )
     6  -  READ INDEX COLUMN : SUPPLIER.S_SUPPKEY
           READ TABLE COLUMN : SUPPLIER.S_NAME, SUPPLIER.S_NATIONKEY
             MIN RANGE : SUPPLIER.S_SUPPKEY = {LINEITEM_VIEW.L_SUPPKEY}
             MAX RANGE : SUPPLIER.S_SUPPKEY = {LINEITEM_VIEW.L_SUPPKEY}
           FETCH ONE ROW
     8  -  COLUMN : NATION.N_NATIONKEY AS N_NATIONKEY
     9  -  TARGET : NATION.N_NATIONKEY
    10  -  READ COLUMN : NATION.N_NATIONKEY, NATION.N_NAME
             LOGICAL FILTER : NATION.N_NAME = 'FRANCE' OR NATION.N_NAME = 'GERMANY'
<<<  end print plan
```

A simple view merging and a subquery unnesting can be applied in the example above, but neither of th ose query transformation method are applied due to NO_QUERY_TRANSFORMATION hint.

〈unnest subquery hints〉

<unnest subquery hints> are <unnest hints>, <unnest join operation hints>, <unnest join driver hints>, <unnest join pusher hints>, <unnest merge hints>.

Subquery unnesting is a part of the query transformation process. Therefore, if NO_QUERY_TRANSFORMATION hint is specified, then subquery unnesting is not performed.

If NO_QUERY_TRANSFORMATION hint and <unnest subquery hints> are simultaneously specified in the same subquery, then the first specified hint is applied only and other hints are ignored.

If <push subquery hinst> and <unnest subquery hints> are simultaneously specified in the same subquery, then then the first specified hint is applied only and other hints are ignored as well. It is because that it can not be applied together with <unnest subquery hints> because <push subquery hints> is a hint for an unnested subquery.

Descriptions and examples for the hints are as follows.

<unnest hints>

UNNEST

If UNNEST hint is specified, an optimizer transforms the subquery into the join statement which guarantees the same result. However, the hint can be applied when it satisfies the subquery unnesting constraints. If the hint is not applicable to the subquery, then the hint is ignored.
For more information about constraints, refer to **SubQuery Unnesting**.

The following is an example of using UNNEST hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey IN (
                       SELECT   /*+ UNNEST */
                                l_orderkey
                         FROM   lineitem
                       GROUP BY l_orderkey
                       HAVING   sum(l_quantity) > 300
                     )
;
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                  ¦
```

```
-------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                             ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                ¦
¦    2  ¦      NESTED JOIN (INNER JOIN)                               ¦
¦    3  ¦        INLINE_VIEW ("$V4")                                  ¦
¦    4  ¦          QUERY BLOCK ("$QB_IDX_6")                          ¦
¦    5  ¦            GROUP HASH INSTANT                               ¦
¦    6  ¦              TABLE ACCESS ("LINEITEM")                      ¦
¦    7  ¦        INDEX ACCESS ("ORDERS", "ORDERS_PK_INDEX")           ¦
=========================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     4  -  TARGET : LINEITEM.L_ORDERKEY
     5  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     6  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
     7  -  READ INDEX COLUMN : ORDERS.O_ORDERKEY
           READ TABLE COLUMN : ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
             MIN RANGE : ORDERS.O_ORDERKEY = {$V4.L_ORDERKEY}
             MAX RANGE : ORDERS.O_ORDERKEY = {$V4.L_ORDERKEY}
           FETCH ONE ROW
<<<  end print plan
```

NO_UNNEST

If NO_UNNEST hint is specified, an optimizer does not unnest a subquery. In other words, the subquery is filtered as it is.

The following is an example of using NO_UNNEST hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey IN (
                      SELECT   /*+ NO_UNNEST */
                               l_orderkey
```

```
                      FROM   lineitem
                    GROUP BY l_orderkey
                    HAVING   sum(l_quantity) > 300
                  )
;
no rows selected.
>>>  start print plan
< Execution Plan >
============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
|    2  |      TABLE ACCESS ("ORDERS")                                      |
|    3  |  SUB QUERY LIST                                                   |
|    4  |    INLINE_VIEW ("$V4") (MATERIALIZED)                             |
|    5  |      QUERY BLOCK ("$QB_IDX_6")                                    |
|    6  |        GROUP HASH INSTANT                                         |
|    7  |          TABLE ACCESS ("LINEITEM")                                |
============================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
           POST FILTER : ( ORDERS.O_ORDERKEY ) IN ( $V4.L_ORDERKEY )
     4  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     5  -  TARGET : LINEITEM.L_ORDERKEY
     6  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     7  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
<<<  end print plan
```

〈unnest join operation hints〉

If 〈unnest join operation hints〉 is specified, an optimizer transforms the subquery into the join statement which guarantees the same result, and performs the join as specified by a hint.

UNNEST_NL

If UNNEST_NL hint is specified, an optimizer transforms the subquery into the join statement which guara ntees the same result, and performs the join by nested loop join method.

The following is an example of using UNNEST_NL hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey IN (
                      SELECT   /*+ UNNEST_NL */
                               l_orderkey
                        FROM   lineitem
                      GROUP BY l_orderkey
                      HAVING   sum(l_quantity) > 300
                     )
;
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                                                      |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                      |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                           |
|    2  |      NESTED JOIN (INNER JOIN)                                          |
|    3  |        INLINE_VIEW ("$V4")                                             |
|    4  |          QUERY BLOCK ("$QB_IDX_6")                                     |
|    5  |            GROUP HASH INSTANT                                          |
|    6  |              TABLE ACCESS ("LINEITEM")                                 |
|    7  |        INDEX ACCESS ("ORDERS", "ORDERS_PK_INDEX")                      |
================================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     4  -  TARGET : LINEITEM.L_ORDERKEY
     5  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     6  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
     7  -  READ INDEX COLUMN : ORDERS.O_ORDERKEY
           READ TABLE COLUMN : ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
```

```
            MIN RANGE : ORDERS.O_ORDERKEY = {$V4.L_ORDERKEY}
            MAX RANGE : ORDERS.O_ORDERKEY = {$V4.L_ORDERKEY}
          FETCH ONE ROW
<<<  end print plan
```

In the example above, a subquery is unnested to INLINE_VIEW ("$V4") and participates in NESTED JOIN.

UNNEST_NL_IN

If UNNEST_NL_IN hint is specified, an optimizer transforms the subquery into the join statement which gu arantees the same result, and performs the join by nested loop join method. Then, it places the subquery which is unnested to a table or a view, on the right of the join.

The following is an example of using UNNEST_NL_IN hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey IN (
                       SELECT   /*+ UNNEST_NL_IN */
                                l_orderkey
                         FROM   lineitem
                       GROUP BY l_orderkey
                       HAVING   sum(l_quantity) > 300
                     )
;
>>>  start print plan
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
|    2  |      NESTED JOIN (INNER JOIN)                                     |
|    3  |        TABLE ACCESS ("ORDERS")                                    |
|    4  |        INLINE_VIEW ("$V5")                                        |
|    5  |          QUERY BLOCK ("$QB_IDX_6")                                |
|    6  |            GROUP                                                  |
|    7  |              INDEX ACCESS ("LINEITEM", "LINEITEM_PK_INDEX")       |
=============================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
```

```
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
     4  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     5  -  TARGET : LINEITEM.L_ORDERKEY
     6  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             LOGICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     7  -  READ INDEX COLUMN : LINEITEM.L_ORDERKEY
           READ TABLE COLUMN : LINEITEM.L_QUANTITY
             MIN RANGE : LINEITEM.L_ORDERKEY = {ORDERS.O_ORDERKEY}
             MAX RANGE : LINEITEM.L_ORDERKEY = {ORDERS.O_ORDERKEY}
<<<  end print plan
```

In the example above, a subquery is unnested to INLINE_VIEW ("$V5"), becomes a right child of NESTED J OIN and participates in the join.

UNNEST_NL_OUT

If UNNEST_NL_OUT hint is specified, an optimizer transforms the subquery into the join statement which guarantees the same result, and performs the join by nested loop join method. Then, it places the subque ry which is unnested to a table or a view, on the left of the join.

The following is an example of using UNNEST_NL_OUT hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey IN (
                       SELECT   /*+ UNNEST_NL_OUT */
                                l_orderkey
                         FROM   lineitem
                       GROUP BY l_orderkey
                       HAVING   sum(l_quantity) > 300
                     )
;
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                  ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                  ¦
```

```
┆    1  ┆    QUERY BLOCK ("$QB_IDX_2")                                  ┆
┆    2  ┆      NESTED JOIN (INNER JOIN)                                 ┆
┆    3  ┆        INLINE_VIEW ("$V4")                                    ┆
┆    4  ┆          QUERY BLOCK ("$QB_IDX_6")                            ┆
┆    5  ┆            GROUP HASH INSTANT                                 ┆
┆    6  ┆              TABLE ACCESS ("LINEITEM")                        ┆
┆    7  ┆        INDEX ACCESS ("ORDERS", "ORDERS_PK_INDEX")             ┆
=======================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     4  -  TARGET : LINEITEM.L_ORDERKEY
     5  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     6  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
     7  -  READ INDEX COLUMN : ORDERS.O_ORDERKEY
           READ TABLE COLUMN : ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
             MIN RANGE : ORDERS.O_ORDERKEY = {$V4.L_ORDERKEY}
             MAX RANGE : ORDERS.O_ORDERKEY = {$V4.L_ORDERKEY}
           FETCH ONE ROW
<<<  end print plan
```

In the example above, a subquery is unnested to INLINE_VIEW ("$V4"), becomes a left child of NESTED J OIN and participates in the join.

UNNEST_INL

If UNNEST_INL hint is specified, an optimizer transforms the subquery into the join statement which guar antees the same result, and performs the join by instant nested loop join method.

The following is an example of using UNNEST_INL hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey IN (
                      SELECT   /*+ UNNEST_INL */
                               l_orderkey
```

```
                        FROM   lineitem
                      GROUP BY l_orderkey
                      HAVING   sum(l_quantity) > 300
                    )
;
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                  ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                    ¦
¦    2  ¦      NESTED JOIN (INNER JOIN)                                   ¦
¦    3  ¦        TABLE ACCESS ("ORDERS")                                  ¦
¦    4  ¦        SORT JOIN INSTANT                                        ¦
¦    5  ¦          INLINE_VIEW ("$V6")                                    ¦
¦    6  ¦            QUERY BLOCK ("$QB_IDX_6")                            ¦
¦    7  ¦              GROUP HASH INSTANT                                 ¦
¦    8  ¦                TABLE ACCESS ("LINEITEM")                        ¦
=============================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
     4  -  SORT KEY : "$V6.L_ORDERKEY ASC NULLS LAST"
           READ KEY COLUMN : $V6.L_ORDERKEY
             MIN RANGE : $V6.L_ORDERKEY = {ORDERS.O_ORDERKEY}
             MAX RANGE : $V6.L_ORDERKEY = {ORDERS.O_ORDERKEY}
     5  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     6  -  TARGET : LINEITEM.L_ORDERKEY
     7  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     8  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
<<<  end print plan
```

In the example above, a subquery is unnested to INLINE_VIEW ("$V6") and participates in NESTED JOIN.

UNNEST_INL_IN

If UNNEST_INL_IN hint is specified, an optimizer transforms the subquery into the join statement which g uarantees the same result, and performs the join by instant nested loop join method. Then, it places the s ubquery which is unnested to a table or a view, on the right of the join.

The following is an example of using UNNEST_INL_IN hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey IN (
                      SELECT   /*+ UNNEST_INL_IN */
                               l_orderkey
                        FROM   lineitem
                      GROUP BY l_orderkey
                      HAVING   sum(l_quantity) > 300
                     )
;
no rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                     ¦
--------------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                    ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                       ¦
¦    2  ¦      NESTED JOIN (INNER JOIN)                                      ¦
¦    3  ¦        TABLE ACCESS ("ORDERS")                                     ¦
¦    4  ¦        SORT JOIN INSTANT                                           ¦
¦    5  ¦          INLINE_VIEW ("$V6")                                       ¦
¦    6  ¦            QUERY BLOCK ("$QB_IDX_6")                               ¦
¦    7  ¦              GROUP HASH INSTANT                                    ¦
¦    8  ¦                TABLE ACCESS ("LINEITEM")                           ¦
================================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
     4  -  SORT KEY : "$V6.L_ORDERKEY ASC NULLS LAST"
           READ KEY COLUMN : $V6.L_ORDERKEY
```

```
            MIN RANGE : $V6.L_ORDERKEY = {ORDERS.O_ORDERKEY}
            MAX RANGE : $V6.L_ORDERKEY = {ORDERS.O_ORDERKEY}
     5  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     6  -  TARGET : LINEITEM.L_ORDERKEY
     7  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     8  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
<<<  end print plan
```

In the example above, a subquery is unnested to INLINE_VIEW ("$V6"), becomes a right child of NESTED J OIN and participates in the join.

UNNEST_INL_OUT

If UNNEST_INL_OUT hint is specified, an optimizer transforms the subquery into the join statement which guarantees the same result, and performs the join by instant nested loop join method. Then, it places the subquery which is unnested to a table or a view, on the left of the join.

The following is an example of using UNNEST_INL_OUT hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey IN (
                       SELECT   /*+ UNNEST_INL_OUT */
                                l_orderkey
                         FROM   lineitem
                       GROUP BY l_orderkey
                       HAVING   sum(l_quantity) > 300
                     )
;
no rows selected.
>>>  start print plan
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
```

```
┆    1  ┆    QUERY BLOCK ("$QB_IDX_2")                                  ┆
┆    2  ┆      NESTED JOIN (INNER JOIN)                                 ┆
┆    3  ┆        INLINE_VIEW ("$V4")                                    ┆
┆    4  ┆          QUERY BLOCK ("$QB_IDX_6")                            ┆
┆    5  ┆            GROUP HASH INSTANT                                 ┆
┆    6  ┆              TABLE ACCESS ("LINEITEM")                        ┆
┆    7  ┆        SORT JOIN INSTANT                                      ┆
┆    8  ┆          TABLE ACCESS ("ORDERS")                              ┆
=========================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     4  -  TARGET : LINEITEM.L_ORDERKEY
     5  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     6  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
     7  -  SORT KEY : "ORDERS.O_ORDERKEY ASC NULLS LAST"
           RECORD COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
           READ KEY COLUMN : ORDERS.O_ORDERKEY
           READ RECORD COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
             MIN RANGE : ORDERS.O_ORDERKEY = {$V4.L_ORDERKEY}
             MAX RANGE : ORDERS.O_ORDERKEY = {$V4.L_ORDERKEY}
     8  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
<<<  end print plan
```

In the example above, a subquery is unnested to INLINE_VIEW ("$V4"), becomes a left child of NESTED J OIN and participates in the join.

UNNEST_HASH

If UNNEST_HASH hint is specified, an optimizer transforms the subquery into the join statement which gu arantees the same result, and performs the join by hash join method.

The following is an example of using UNNEST_HASH hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
```

```
 WHERE o_orderkey IN (
                      SELECT   /*+ UNNEST_HASH */
                               l_orderkey
                        FROM   lineitem
                      GROUP BY l_orderkey
                      HAVING   sum(l_quantity) > 300
                    )
;
>>>  start print plan
< Execution Plan >
============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
|    2  |      HASH JOIN (INNER JOIN)                                       |
|    3  |        TABLE ACCESS ("ORDERS")                                    |
|    4  |        HASH JOIN INSTANT                                          |
|    5  |          INLINE_VIEW ("$V6")                                      |
|    6  |            QUERY BLOCK ("$QB_IDX_6")                              |
|    7  |              GROUP HASH INSTANT                                   |
|    8  |                TABLE ACCESS ("LINEITEM")                          |
============================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
     4  -  HASH KEY : $V6.L_ORDERKEY
           READ KEY COLUMN : $V6.L_ORDERKEY
             HASH FILTER : $V6.L_ORDERKEY = ORDERS.O_ORDERKEY
           FETCH ONE ROW
     5  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     6  -  TARGET : LINEITEM.L_ORDERKEY
     7  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     8  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
<<<  end print plan
```

In the example above, a subquery is unnested to INLINE_VIEW ("$V6") and participates in NESTED JOIN.

UNNEST_HASH(hash_bucket_count)

UNNEST_HASH(hash_bucket_count) hint is as same as UNNEST_HASH hint, but in addition to that it can specify the hash bucket count.
Therefore, if UNNEST_HASH(hash_bucket_count) hint is used, then it creates hash buckets as many as ha sh_bucket_count.

The following is an example of using UNNEST_HASH(hash_bucket_count) hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey IN (
                      SELECT   /*+ UNNEST_HASH(3) */
                               l_orderkey
                        FROM   lineitem
                      GROUP BY l_orderkey
                      HAVING   sum(l_quantity) > 300
                    )
;
>>>  start print plan
< Execution Plan >
============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
|    2  |      HASH JOIN (INNER JOIN)                                       |
|    3  |        TABLE ACCESS ("ORDERS")                                    |
|    4  |        HASH JOIN INSTANT                                          |
|    5  |          INLINE_VIEW ("$V6")                                      |
|    6  |            QUERY BLOCK ("$QB_IDX_6")                              |
|    7  |              GROUP HASH INSTANT                                   |
|    8  |                TABLE ACCESS ("LINEITEM")                          |
============================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
     4  -  HASH KEY : $V6.L_ORDERKEY
           READ KEY COLUMN : $V6.L_ORDERKEY
```

```
            HASH FILTER : $V6.L_ORDERKEY = ORDERS.O_ORDERKEY
          FETCH ONE ROW
     5  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     6  -  TARGET : LINEITEM.L_ORDERKEY
     7  -  GROUP KEY : LINEITEM.L_ORDERKEY
          RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
          READ KEY COLUMN : LINEITEM.L_ORDERKEY
          READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
            PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     8  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
<<<  end print plan
```

The execution plan is as same as the example of using UNNEST_HASH hint, but hash buckets are created in HASH JOIN INSTANT as many as hash bucket count when using the hint above. Therefore, use this hint to adjust hash bucket count if the performance gets slower because hash bucket count is estimated too b ig or too small while performing cost estimation.

UNNEST_HASH_IN

If UNNEST_HASH_IN hint is specified, an optimizer transforms the subquery into the join statement which guarantees the same result, and performs the join by hash join method. Then, it places the subquery whi ch is unnested to a table or a view, on the right of the join.

The following is an example of using UNNEST_HASH_IN hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey IN (
                       SELECT   /*+ UNNEST_HASH_IN */
                                l_orderkey
                         FROM   lineitem
                       GROUP BY l_orderkey
                       HAVING   sum(l_quantity) > 300
                     )
;
>>>  start print plan
< Execution Plan >
================================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                     ¦
--------------------------------------------------------------------------------
```

```
|    0  |  SELECT STATEMENT                                          |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                               |
|    2  |      HASH JOIN (INNER JOIN)                                |
|    3  |        TABLE ACCESS ("ORDERS")                             |
|    4  |        HASH JOIN INSTANT                                   |
|    5  |          INLINE_VIEW ("$V6")                               |
|    6  |            QUERY BLOCK ("$QB_IDX_6")                       |
|    7  |              GROUP HASH INSTANT                            |
|    8  |                TABLE ACCESS ("LINEITEM")                   |
=====================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
     4  -  HASH KEY : $V6.L_ORDERKEY
           READ KEY COLUMN : $V6.L_ORDERKEY
             HASH FILTER : $V6.L_ORDERKEY = ORDERS.O_ORDERKEY
           FETCH ONE ROW
     5  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     6  -  TARGET : LINEITEM.L_ORDERKEY
     7  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     8  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
<<<  end print plan
```

In the example above, a subquery is unnested to INLINE_VIEW ("$V4"), becomes a right child of NESTED J OIN and participates in the join.

UNNEST_HASH_IN(hash_bucket_count)

UNNEST_HASH_IN(hash_bucket_count) hint is as same as UNNEST_HASH_IN hint, but in addition to that it can specify the hash bucket count.
Therefore, if UNNEST_HASH_IN(hash_bucket_count) hint is used, then it creates hash buckets as many as hash_bucket_count.

The following is an example of using UNNEST_HASH_IN(hash_bucket_count) hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
```

```
       o_totalprice
  FROM orders
 WHERE o_orderkey IN (
                       SELECT   /*+ UNNEST_HASH_IN(3) */
                                l_orderkey
                         FROM   lineitem
                       GROUP BY l_orderkey
                       HAVING   sum(l_quantity) > 300
                     )
;
>>>  start print plan
< Execution Plan >
============================================================================
|  IDX  |  NODE DESCRIPTION                                                |
----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                     |
|    2  |      HASH JOIN (INNER JOIN)                                      |
|    3  |        TABLE ACCESS ("ORDERS")                                   |
|    4  |        HASH JOIN INSTANT                                         |
|    5  |          INLINE_VIEW ("$V6")                                     |
|    6  |            QUERY BLOCK ("$QB_IDX_6")                             |
|    7  |              GROUP HASH INSTANT                                  |
|    8  |                TABLE ACCESS ("LINEITEM")                         |
============================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
     4  -  HASH KEY : $V6.L_ORDERKEY
           READ KEY COLUMN : $V6.L_ORDERKEY
             HASH FILTER : $V6.L_ORDERKEY = ORDERS.O_ORDERKEY
           FETCH ONE ROW
     5  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     6  -  TARGET : LINEITEM.L_ORDERKEY
     7  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     8  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
<<<  end print plan
```

The execution plan is as same as the example of using UNNEST_HASH_IN hint, but hash buckets are created in HASH JOIN INSTANT as many as hash bucket count when using the hint above. Therefore, use this hint to adjust hash bucket count if the performance gets slower because hash bucket count is estimated too big or too small while performing cost estimation.

UNNEST_HASH_OUT

If UNNEST_HASH_OUT hint is specified, an optimizer transforms the subquery into the join statement which guarantees the same result, and performs the join by hash join method. Then, it places the subquery which is unnested to a table or a view, on the left of the join.

The following is an example of using UNNEST_HASH_OUT hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey IN (
                       SELECT   /*+ UNNEST_HASH_OUT */
                                l_orderkey
                         FROM   lineitem
                       GROUP BY l_orderkey
                       HAVING   sum(l_quantity) > 300
                     )
;
>>>  start print plan
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
|    2  |      HASH JOIN (INNER JOIN)                                       |
|    3  |        INLINE_VIEW ("$V4")                                        |
|    4  |          QUERY BLOCK ("$QB_IDX_6")                                |
|    5  |            GROUP HASH INSTANT                                     |
|    6  |              TABLE ACCESS ("LINEITEM")                            |
|    7  |        HASH JOIN INSTANT                                          |
|    8  |          TABLE ACCESS ("ORDERS")                                  |
=============================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
```

```
     3  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     4  -  TARGET : LINEITEM.L_ORDERKEY
     5  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     6  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
     7  -  HASH KEY : ORDERS.O_ORDERKEY
           RECORD COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
           READ KEY COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
             HASH FILTER : ORDERS.O_ORDERKEY = $V4.L_ORDERKEY
           FETCH ONE ROW
     8  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
<<<  end print plan
```

In the example above, a subquery is unnested to INLINE_VIEW ("$V4"), becomes a left child of HASH JOI N and participates in the join.

UNNEST_HASH_OUT(hash_bucket_count)

UNNEST_HASH_OUT(hash_bucket_count) hint is as same as UNNEST_HASH_OUT hint, but in addition to that it can specify the hash bucket count.
Therefore, if UNNEST_HASH_OUT(hash_bucket_count) hint is used, then it creates hash buckets as many as hash_bucket_count.

The following is an example of using UNNEST_HASH_OUT(hash_bucket_count) hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey IN (
                       SELECT   /*+ UNNEST_HASH_OUT(3) */
                                l_orderkey
                         FROM   lineitem
                       GROUP BY l_orderkey
                       HAVING   sum(l_quantity) > 300
                     )
;
>>>  start print plan
```

```
< Execution Plan >
============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
|    2  |      HASH JOIN (INNER JOIN)                                       |
|    3  |        INLINE_VIEW ("$V4")                                        |
|    4  |          QUERY BLOCK ("$QB_IDX_6")                                |
|    5  |            GROUP HASH INSTANT                                     |
|    6  |              TABLE ACCESS ("LINEITEM")                            |
|    7  |        HASH JOIN INSTANT                                          |
|    8  |          TABLE ACCESS ("ORDERS")                                  |
============================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     4  -  TARGET : LINEITEM.L_ORDERKEY
     5  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     6  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
     7  -  HASH KEY : ORDERS.O_ORDERKEY
           RECORD COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
           READ KEY COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
             HASH FILTER : ORDERS.O_ORDERKEY = $V4.L_ORDERKEY
           FETCH ONE ROW
     8  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
<<<  end print plan
```

The execution plan is as same as the example of using UNNEST_HASH_OUT hint, but hash buckets are cr eated in HASH JOIN INSTANT as many as hash bucket count when using the hint above. Therefore, use th is hint to adjust hash bucket count if the performance gets slower because hash bucket count is estimate d too big or too small while performing cost estimation.

UNNEST_MERGE

If UNNEST_MERGE hint is specified, an optimizer transforms the subquery into the join statement which guarantees the same result, and performs the join by merge join method.

The following is an example of using UNNEST_MERGE hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey IN (
                       SELECT   /*+ UNNEST_MERGE */
                                l_orderkey
                         FROM   lineitem
                       GROUP BY l_orderkey
                       HAVING   sum(l_quantity) > 300
                     )
;
>>>  start print plan
< Execution Plan >
============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
|    2  |      MERGE JOIN (INNER JOIN)                                      |
|    3  |        INDEX ACCESS ("ORDERS", "ORDERS_PK_INDEX")                 |
|    4  |        SORT JOIN INSTANT                                          |
|    5  |          INLINE_VIEW ("$V6")                                      |
|    6  |            QUERY BLOCK ("$QB_IDX_6")                              |
|    7  |              GROUP HASH INSTANT                                   |
|    8  |                TABLE ACCESS ("LINEITEM")                          |
============================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
           ON FILTER (Equi) : ORDERS.O_ORDERKEY = $V6.L_ORDERKEY
     3  -  READ INDEX COLUMN : ORDERS.O_ORDERKEY
           READ TABLE COLUMN : ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
     4  -  SORT KEY : "$V6.L_ORDERKEY ASC NULLS LAST"
           READ KEY COLUMN : $V6.L_ORDERKEY
           MIN RANGE : $V6.L_ORDERKEY >= {ORDERS.O_ORDERKEY}
           MAX RANGE : $V6.L_ORDERKEY IS NOT NULL
     5  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     6  -  TARGET : LINEITEM.L_ORDERKEY
     7  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
```

```
            READ KEY COLUMN : LINEITEM.L_ORDERKEY
            READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
              PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
      8  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
<<<  end print plan
```

In the example above, a subquery is unnested to INLINE_VIEW ("$V6"), and participates in MERGE JOIN.

UNNEST_MERGE_IN

If UNNEST_MERGE_IN hint is specified, an optimizer transforms the subquery into the join statement whi ch guarantees the same result, and performs the join by merge join method. Then, it places the subquery which is unnested to a table or a view, on the right of the join.

The following is an example of using UNNEST_MERGE_IN hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey IN (
                      SELECT   /*+ UNNEST_MERGE_IN */
                               l_orderkey
                        FROM   lineitem
                      GROUP BY l_orderkey
                      HAVING   sum(l_quantity) > 300
                    )
;
>>>  start print plan
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
|    2  |      MERGE JOIN (INNER JOIN)                                      |
|    3  |        INDEX ACCESS ("ORDERS", "ORDERS_PK_INDEX")                 |
|    4  |        SORT JOIN INSTANT                                          |
|    5  |          INLINE_VIEW ("$V6")                                      |
|    6  |            QUERY BLOCK ("$QB_IDX_6")                              |
|    7  |              GROUP HASH INSTANT                                   |
|    8  |                TABLE ACCESS ("LINEITEM")                          |
```

```
=======================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
             ON FILTER (Equi) : ORDERS.O_ORDERKEY = $V6.L_ORDERKEY
     3  -  READ INDEX COLUMN : ORDERS.O_ORDERKEY
           READ TABLE COLUMN : ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
     4  -  SORT KEY : "$V6.L_ORDERKEY ASC NULLS LAST"
           READ KEY COLUMN : $V6.L_ORDERKEY
             MIN RANGE : $V6.L_ORDERKEY >= {ORDERS.O_ORDERKEY}
             MAX RANGE : $V6.L_ORDERKEY IS NOT NULL
     5  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     6  -  TARGET : LINEITEM.L_ORDERKEY
     7  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     8  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
<<<  end print plan
```

In the example above, a subquery is unnested to INLINE_VIEW ("$V6"), becomes a right child of MERGE JOIN and participates in the join.

UNNEST_MERGE_OUT

If UNNEST_MERGE_OUT hint is specified, an optimizer transforms the subquery into the join statement which guarantees the same result, and performs the join by merge join method. Then, it places the subquery which is unnested to a table or a view, on the left of the join.

The following is an example of using UNNEST_MERGE_OUT hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey IN (
                       SELECT   /*+ UNNEST_MERGE_OUT */
                                l_orderkey
                         FROM   lineitem
                       GROUP BY l_orderkey
                       HAVING   sum(l_quantity) > 300
                     )
```

```
;
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                  ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                    ¦
¦    2  ¦      MERGE JOIN (INNER JOIN)                                    ¦
¦    3  ¦        SORT JOIN INSTANT                                        ¦
¦    4  ¦          INLINE_VIEW ("$V5")                                    ¦
¦    5  ¦            QUERY BLOCK ("$QB_IDX_6")                            ¦
¦    6  ¦              GROUP HASH INSTANT                                 ¦
¦    7  ¦                TABLE ACCESS ("LINEITEM")                        ¦
¦    8  ¦        INDEX ACCESS ("ORDERS", "ORDERS_PK_INDEX")               ¦
=============================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
             ON FILTER (Equi) : $V5.L_ORDERKEY = ORDERS.O_ORDERKEY
     3  -  SORT KEY : "$V5.L_ORDERKEY ASC NULLS LAST"
           READ KEY COLUMN : $V5.L_ORDERKEY
     4  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     5  -  TARGET : LINEITEM.L_ORDERKEY
     6  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     7  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
     8  -  READ INDEX COLUMN : ORDERS.O_ORDERKEY
           READ TABLE COLUMN : ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
             MIN RANGE : ORDERS.O_ORDERKEY >= {$V5.L_ORDERKEY}
             MAX RANGE : ORDERS.O_ORDERKEY IS NOT NULL
<<<  end print plan
```

In the example above, a subquery is unnested to INLINE_VIEW ("$V5"), becomes a left child of MERGE JO IN and participates in the join.

NL_SJ

If NL_SJ hint is specified, an optimizer unnests the subquery in semi join form, and performs the semi join in nested loop method.

It is operated in the same way as that of UNNEST_NL hint. However, UNNEST_NL hint is applied to both s emi join and anti-semi join, on the other hand, NL_SJ hint is applied only to semi join.
Therefore, if the hint above is specified on a subquery which is unnested with anti-semi join, then an opti mizer ignores it.

The following is an example of using NL_SJ hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey IN (
                       SELECT   /*+ NL_SJ */
                                l_orderkey
                         FROM   lineitem
                       GROUP BY l_orderkey
                       HAVING   sum(l_quantity) > 300
                     )
;
>>>  start print plan
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
|    2  |      NESTED JOIN (INNER JOIN)                                     |
|    3  |        TABLE ACCESS ("ORDERS")                                    |
|    4  |        INLINE_VIEW ("$V5")                                        |
|    5  |          QUERY BLOCK ("$QB_IDX_6")                                |
|    6  |            GROUP                                                  |
|    7  |              INDEX ACCESS ("LINEITEM", "LINEITEM_PK_INDEX")       |
=============================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
     4  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     5  -  TARGET : LINEITEM.L_ORDERKEY
```

```
     6  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             LOGICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     7  -  READ INDEX COLUMN : LINEITEM.L_ORDERKEY
           READ TABLE COLUMN : LINEITEM.L_QUANTITY
             MIN RANGE : LINEITEM.L_ORDERKEY = {ORDERS.O_ORDERKEY}
             MAX RANGE : LINEITEM.L_ORDERKEY = {ORDERS.O_ORDERKEY}
<<<  end print plan
```

In the example above, NL_SJ hint is applied so a subquery is unnested with semi join and nested loop join is performed. Also, the semi join is converted into an inner join because both o_orderkey and l_orderkey are primary keys.

The following is an example of using NL_SJ hint on a subquery which can be unnested only in anti-semi j oin form.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey NOT IN (
                           SELECT   /*+ NL_SJ */
                                    l_orderkey
                             FROM   lineitem
                            GROUP BY l_orderkey
                            HAVING   sum(l_quantity) > 300
                           )
;
>>>  start print plan
< Execution Plan >
============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                 ¦
----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                   ¦
¦    2  ¦      HASH JOIN (ANTI SEMI)                                     ¦
¦    3  ¦        TABLE ACCESS ("ORDERS")                                 ¦
¦    4  ¦        HASH JOIN INSTANT (UNIQUE)                              ¦
¦    5  ¦          INLINE_VIEW ("$V6")                                   ¦
¦    6  ¦            QUERY BLOCK ("$QB_IDX_6")                           ¦
¦    7  ¦              GROUP HASH INSTANT                                ¦
```

```
|    8  |               TABLE ACCESS ("LINEITEM")                    |
=============================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
     4  -  HASH KEY : $V6.L_ORDERKEY
           READ KEY COLUMN : $V6.L_ORDERKEY
             HASH FILTER : $V6.L_ORDERKEY = ORDERS.O_ORDERKEY
           FETCH ONE ROW
     5  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     6  -  TARGET : LINEITEM.L_ORDERKEY
     7  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     8  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
<<<  end print plan
```

The hint is ignored so it is unnested by anti-semi join method, then performed in hash join method.

NL_ISJ

If NL_ISJ hint is specified, an optimizer unnests the subquery in inverted semi join form, and performs the semi join in nested loop method. Then, it places the subquery which is unnested to a table or a view, on t he left of the join because it is an inversted semi join.

It is operated in the same way as that of UNNEST_NL_OUT hint. However, UNNEST_NL_OUT hint is applie d to both semi join and anti-semi join, on the other hand, NL_ISJ hint is applied only to semi join. Therefore, if the hint above is specified on a subquery which is unnested with anti-semi join, then an opti mizer ignores it.

The following is an example of using NL_ISJ hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey IN (
                      SELECT   /*+ NL_ISJ */
                               l_orderkey
                        FROM   lineitem
```

```
                    GROUP BY l_orderkey
                    HAVING   sum(l_quantity) > 300
                  )
;
>>>  start print plan
< Execution Plan >
============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                 ¦
----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                   ¦
¦    2  ¦      NESTED JOIN (INNER JOIN)                                  ¦
¦    3  ¦        INLINE_VIEW ("$V4")                                     ¦
¦    4  ¦          QUERY BLOCK ("$QB_IDX_6")                             ¦
¦    5  ¦            GROUP HASH INSTANT                                  ¦
¦    6  ¦              TABLE ACCESS ("LINEITEM")                         ¦
¦    7  ¦        INDEX ACCESS ("ORDERS", "ORDERS_PK_INDEX")              ¦
============================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     4  -  TARGET : LINEITEM.L_ORDERKEY
     5  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     6  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
     7  -  READ INDEX COLUMN : ORDERS.O_ORDERKEY
           READ TABLE COLUMN : ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
             MIN RANGE : ORDERS.O_ORDERKEY = {$V4.L_ORDERKEY}
             MAX RANGE : ORDERS.O_ORDERKEY = {$V4.L_ORDERKEY}
           FETCH ONE ROW
<<<  end print plan
```

In the example above, NL_ISJ hint is applied so a subquery is unnested with inverted semi join and nested loop join is performed. Also, the semi join is converted into an inner join because both o_orderkey and l_orderkey are primary keys.

The following is an example of using NL_ISJ hint in a subquery which can be unnested only in anti-semi jo in form.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey NOT IN (
                          SELECT   /*+ NL_ISJ */
                                   l_orderkey
                            FROM   lineitem
                           GROUP BY l_orderkey
                           HAVING   sum(l_quantity) > 300
                         )
;
>>>  start print plan
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
|    2  |      HASH JOIN (ANTI SEMI)                                        |
|    3  |        TABLE ACCESS ("ORDERS")                                    |
|    4  |        HASH JOIN INSTANT (UNIQUE)                                 |
|    5  |          INLINE_VIEW ("$V6")                                      |
|    6  |            QUERY BLOCK ("$QB_IDX_6")                              |
|    7  |              GROUP HASH INSTANT                                   |
|    8  |                TABLE ACCESS ("LINEITEM")                          |
=============================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
     4  -  HASH KEY : $V6.L_ORDERKEY
           READ KEY COLUMN : $V6.L_ORDERKEY
             HASH FILTER : $V6.L_ORDERKEY = ORDERS.O_ORDERKEY
           FETCH ONE ROW
     5  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     6  -  TARGET : LINEITEM.L_ORDERKEY
     7  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
```

```
            PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     8  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
<<<  end print plan
```

The hint is ignored so it is unnested by anti-semi join method, performed in hash join method, and the un nested subquery is located on the right.

INL_SJ

If INL_SJ hint is specified, an optimizer unnests the subquery in semi join form, and performs the semi join in instant nested loop method.

It is operated in the same way as that of UNNEST_INL hint. However, UNNEST_INL hint is applied to both semi join and anti-semi join, on the other hand, INL_SJ hint is applied only to semi join. Therefore, if the h int above is specified on a subquery which is unnested with anti-semi join, then an optimizer ignores it.

The following is an example of using INL_SJ hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey IN (
                       SELECT   /*+ INL_SJ */
                                l_orderkey
                         FROM   lineitem
                       GROUP BY l_orderkey
                       HAVING   sum(l_quantity) > 300
                      )
;
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                  ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                    ¦
¦    2  ¦      NESTED JOIN (INNER JOIN)                                   ¦
¦    3  ¦        TABLE ACCESS ("ORDERS")                                  ¦
¦    4  ¦        SORT JOIN INSTANT                                        ¦
¦    5  ¦          INLINE_VIEW ("$V6")                                    ¦
¦    6  ¦            QUERY BLOCK ("$QB_IDX_6")                            ¦
```

```
|    7  |             GROUP HASH INSTANT                              |
|    8  |               TABLE ACCESS ("LINEITEM")                     |
=======================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
     4  -  SORT KEY : "$V6.L_ORDERKEY ASC NULLS LAST"
           READ KEY COLUMN : $V6.L_ORDERKEY
             MIN RANGE : $V6.L_ORDERKEY = {ORDERS.O_ORDERKEY}
             MAX RANGE : $V6.L_ORDERKEY = {ORDERS.O_ORDERKEY}
     5  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     6  -  TARGET : LINEITEM.L_ORDERKEY
     7  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     8  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
<<<  end print plan
```

In the example above, INL_SJ hint is applied so a subquery is unnested with semi join and instant nested l oop join is performed. Also, the semi join is converted into an inner join because both o_orderkey and l_o rderkey are primary keys.

The following is an example of using INL_SJ hint in a subquery which can be unnested only in anti-semi jo in form.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey NOT IN (
                           SELECT   /*+ INL_SJ */
                                    l_orderkey
                             FROM   lineitem
                            GROUP BY l_orderkey
                            HAVING   sum(l_quantity) > 300
                          )
;
>>>  start print plan
< Execution Plan >
```

```
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                 ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                    ¦
¦    2  ¦      HASH JOIN (ANTI SEMI)                                      ¦
¦    3  ¦        TABLE ACCESS ("ORDERS")                                  ¦
¦    4  ¦        HASH JOIN INSTANT (UNIQUE)                               ¦
¦    5  ¦          INLINE_VIEW ("$V6")                                    ¦
¦    6  ¦            QUERY BLOCK ("$QB_IDX_6")                            ¦
¦    7  ¦              GROUP HASH INSTANT                                 ¦
¦    8  ¦                TABLE ACCESS ("LINEITEM")                        ¦
=============================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
     4  -  HASH KEY : $V6.L_ORDERKEY
           READ KEY COLUMN : $V6.L_ORDERKEY
             HASH FILTER : $V6.L_ORDERKEY = ORDERS.O_ORDERKEY
           FETCH ONE ROW
     5  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     6  -  TARGET : LINEITEM.L_ORDERKEY
     7  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     8  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
<<<  end print plan
```

The hint is ignored so it is unnested by anti-semi join method, then performed in hash join method.

MERGE_SJ

If MERGE_SJ hint is specified, an optimizer unnests the subquery in semi join form, and performs the sem i join in merge join method.

It is operated in the same way as that of UNNEST_MERGE hint. However, UNNEST_MERGE hint is applied to both semi join and anti-semi join, on the other hand, MERGE_SJ hint is applied only to semi join. Therefore, if the hint above is specified on a subquery which is unnested with anti-semi join, then an opti mizer ignores it.

The following is an example of using MERGE_SJ hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey IN (
                      SELECT   /*+ MERGE_SJ */
                               l_orderkey
                        FROM   lineitem
                      GROUP BY l_orderkey
                      HAVING   sum(l_quantity) > 300
                    )
;
>>>  start print plan
< Execution Plan >
============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                 ¦
----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                    ¦
¦    2  ¦      MERGE JOIN (INNER JOIN)                                    ¦
¦    3  ¦        INDEX ACCESS ("ORDERS", "ORDERS_PK_INDEX")               ¦
¦    4  ¦        SORT JOIN INSTANT                                        ¦
¦    5  ¦          INLINE_VIEW ("$V6")                                    ¦
¦    6  ¦            QUERY BLOCK ("$QB_IDX_6")                            ¦
¦    7  ¦              GROUP HASH INSTANT                                 ¦
¦    8  ¦                TABLE ACCESS ("LINEITEM")                        ¦
============================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
             ON FILTER (Equi) : ORDERS.O_ORDERKEY = $V6.L_ORDERKEY
     3  -  READ INDEX COLUMN : ORDERS.O_ORDERKEY
           READ TABLE COLUMN : ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
     4  -  SORT KEY : "$V6.L_ORDERKEY ASC NULLS LAST"
           READ KEY COLUMN : $V6.L_ORDERKEY
             MIN RANGE : $V6.L_ORDERKEY >= {ORDERS.O_ORDERKEY}
             MAX RANGE : $V6.L_ORDERKEY IS NOT NULL
     5  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     6  -  TARGET : LINEITEM.L_ORDERKEY
```

```
     7  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     8  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
<<<  end print plan
```

In the example above, MERGE_SJ hint is applied so a subquery is unnested with semi join and merge join is performed. Also, the semi join is converted into an inner join because both o_orderkey and l_orderkey are primary keys.

The following is an example of using MERGE_SJ hint in a subquery which can be unnested only in anti-se mi join form.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey NOT IN (
                          SELECT   /*+ MERGE_SJ */
                                   l_orderkey
                            FROM   lineitem
                           GROUP BY l_orderkey
                           HAVING   sum(l_quantity) > 300
                         )
;
>>>  start print plan
< Execution Plan >
=========================================================================
¦  IDX  ¦  NODE DESCRIPTION                                              ¦
-------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                              ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                 ¦
¦    2  ¦      HASH JOIN (ANTI SEMI)                                   ¦
¦    3  ¦        TABLE ACCESS ("ORDERS")                               ¦
¦    4  ¦        HASH JOIN INSTANT (UNIQUE)                            ¦
¦    5  ¦          INLINE_VIEW ("$V6")                                 ¦
¦    6  ¦            QUERY BLOCK ("$QB_IDX_6")                         ¦
¦    7  ¦              GROUP HASH INSTANT                              ¦
¦    8  ¦                TABLE ACCESS ("LINEITEM")                     ¦
```

```
========================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
     4  -  HASH KEY : $V6.L_ORDERKEY
           READ KEY COLUMN : $V6.L_ORDERKEY
             HASH FILTER : $V6.L_ORDERKEY = ORDERS.O_ORDERKEY
           FETCH ONE ROW
     5  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     6  -  TARGET : LINEITEM.L_ORDERKEY
     7  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     8  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
<<<  end print plan
```

The hint is ignored so it is unnested by anti-semi join method, then performed in hash join method.

HASH_SJ

If HASH_SJ hint is specified, an optimizer unnests the subquery in semi join form, and performs the semi j oin in hash join method.

It is operated in the same way as that of UNNEST_HASH hint. However, UNNEST_HASH hint is applied to both semi join and anti-semi join, on the other hand, HASH_SJ hint is applied only to semi join. Therefore, if the hint above is specified on a subquery which is unnested with anti-semi join, then an optimizer ignor es it.

The following is an example of using HASH_SJ hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey IN (
                      SELECT   /*+ HASH_SJ */
                               l_orderkey
                        FROM   lineitem
                      GROUP BY l_orderkey
                      HAVING   sum(l_quantity) > 300
```

```
                    )
;
no rows selected.
>>>  start print plan
< Execution Plan >
============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                 ¦
----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                   ¦
¦    2  ¦      HASH JOIN (INNER JOIN)                                    ¦
¦    3  ¦        TABLE ACCESS ("ORDERS")                                 ¦
¦    4  ¦        HASH JOIN INSTANT                                       ¦
¦    5  ¦          INLINE_VIEW ("$V6")                                   ¦
¦    6  ¦            QUERY BLOCK ("$QB_IDX_6")                           ¦
¦    7  ¦              GROUP HASH INSTANT                                ¦
¦    8  ¦                TABLE ACCESS ("LINEITEM")                       ¦
============================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
     4  -  HASH KEY : $V6.L_ORDERKEY
           READ KEY COLUMN : $V6.L_ORDERKEY
             HASH FILTER : $V6.L_ORDERKEY = ORDERS.O_ORDERKEY
           FETCH ONE ROW
     5  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     6  -  TARGET : LINEITEM.L_ORDERKEY
     7  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     8  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
<<<  end print plan
```

In the example above, HASH_SJ hint is applied so a subquery is unnested with semi join and hash join is p erformed. Also, the semi join is converted into an inner join because both o_orderkey and l_orderkey are primary keys.

The following is an example of using HASH_SJ hint in a subquery which can be unnested only in anti-sem i join form.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey NOT IN (
                          SELECT   /*+ HASH_SJ */
                                   l_orderkey
                            FROM   lineitem
                           WHERE   l_shipdate >= date '1996-01-01'
                             AND   l_shipdate < date '1996-02-01'
                             AND   l_commitdate < l_receiptdate
                           GROUP BY l_orderkey
                           HAVING   sum(l_quantity) > 300
                         )
   AND  o_orderdate >= date '1996-01-01'
   AND  o_orderdate < date '1996-02-01'
   AND  o_comment like '%Customer%Complaints%'
   AND  o_orderstatus = 'F'
;
>>>  start print plan
< Execution Plan >
================================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                     ¦
--------------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                    ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                       ¦
¦    2  ¦      NESTED JOIN (ANTI SEMI)                                       ¦
¦    3  ¦        TABLE ACCESS ("ORDERS")                                     ¦
¦    4  ¦        INLINE_VIEW ("$V5")                                         ¦
¦    5  ¦          QUERY BLOCK ("$QB_IDX_6")                                 ¦
¦    6  ¦            GROUP                                                   ¦
¦    7  ¦              INDEX ACCESS ("LINEITEM", "LINEITEM_PK_INDEX")        ¦
================================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_ORDERSTATUS, ORDERS.O_TOTALPRICE,
ORDERS.O_ORDERDATE, ORDERS.O_COMMENT
             PHYSICAL FILTER : ORDERS.O_ORDERSTATUS = 'F' AND ORDERS.O_ORDERDATE >=
DATE'1996-01-01' AND ORDERS.O_ORDERDATE < DATE'1996-02-01'
             LOGICAL FILTER : ORDERS.O_COMMENT LIKE '%Customer%Complaints%'
```

```
     4  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     5  -  TARGET : LINEITEM.L_ORDERKEY
     6  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             LOGICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     7  -  READ INDEX COLUMN : LINEITEM.L_ORDERKEY
           READ TABLE COLUMN : LINEITEM.L_QUANTITY, LINEITEM.L_SHIPDATE,
LINEITEM.L_COMMITDATE, LINEITEM.L_RECEIPTDATE
             MIN RANGE : LINEITEM.L_ORDERKEY = {ORDERS.O_ORDERKEY}
             MAX RANGE : LINEITEM.L_ORDERKEY = {ORDERS.O_ORDERKEY}
             PHYSICAL TABLE FILTER : LINEITEM.L_SHIPDATE >= DATE'1996-01-01' AND
LINEITEM.L_SHIPDATE < DATE'1996-02-01' AND LINEITEM.L_RECEIPTDATE > LINEITEM.L_COMMITDATE
<<<  end print plan
```

The hint is ignored so it is unnested by anti-semi join method, then performed in nested loop join metho d.

HASH_ISJ

If HASH_ISJ hint is specified, an optimizer unnests the subquery in inverted semi join form, and performs t he join in hash join method.

It is operated in the same way as that of UNNEST_HASH_OUT hint. However, UNNEST_HASH_OUT hint is applied to both semi join and anti-semi join, on the other hand, HASH_ISJ hint is applied only to semi join. Therefore, if the hint above is specified on a subquery which is unnested with anti-semi join, then an opti mizer ignores it.

The following is an example of using HASH_ISJ hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey IN (
                      SELECT   /*+ HASH_ISJ */
                               l_orderkey
                        FROM   lineitem
                      GROUP BY l_orderkey
                      HAVING   sum(l_quantity) > 300
                    )
;
>>>  start print plan
```

```
< Execution Plan >
============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
|    2  |      HASH JOIN (INNER JOIN)                                       |
|    3  |        INLINE_VIEW ("$V4")                                        |
|    4  |          QUERY BLOCK ("$QB_IDX_6")                                |
|    5  |            GROUP HASH INSTANT                                     |
|    6  |              TABLE ACCESS ("LINEITEM")                            |
|    7  |        HASH JOIN INSTANT                                          |
|    8  |          TABLE ACCESS ("ORDERS")                                  |
============================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     4  -  TARGET : LINEITEM.L_ORDERKEY
     5  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     6  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
     7  -  HASH KEY : ORDERS.O_ORDERKEY
           RECORD COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
           READ KEY COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
             HASH FILTER : ORDERS.O_ORDERKEY = $V4.L_ORDERKEY
           FETCH ONE ROW
     8  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
<<<  end print plan
```

In the example above, HASH_ISJ hint is applied so a subquery is unnested with inverted semi join and has h join is performed. Also, the semi join is converted into an inner join because both o_orderkey and l_ord erkey are primary keys.

The following is an example of using HASH_ISJ hint in a subquery which can be unnested only in anti-se mi join form.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
```

```
       o_totalprice
  FROM orders
 WHERE o_orderkey NOT IN (
                          SELECT   /*+ HASH_ISJ */
                                   l_orderkey
                            FROM   lineitem
                           WHERE   l_shipdate >= date '1996-01-01'
                             AND   l_shipdate < date '1996-02-01'
                             AND   l_commitdate < l_receiptdate
                           GROUP BY l_orderkey
                           HAVING   sum(l_quantity) > 300
                         )
   AND  o_orderdate >= date '1996-01-01'
   AND  o_orderdate < date '1996-02-01'
   AND  o_comment like '%Customer%Complaints%'
   AND  o_orderstatus = 'F'
;
>>>  start print plan
< Execution Plan >
============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
|    2  |      NESTED JOIN (ANTI SEMI)                                      |
|    3  |        TABLE ACCESS ("ORDERS")                                    |
|    4  |        INLINE_VIEW ("$V5")                                        |
|    5  |          QUERY BLOCK ("$QB_IDX_6")                                |
|    6  |            GROUP                                                  |
|    7  |              INDEX ACCESS ("LINEITEM", "LINEITEM_PK_INDEX")       |
============================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_ORDERSTATUS, ORDERS.O_TOTALPRICE,
ORDERS.O_ORDERDATE, ORDERS.O_COMMENT
             PHYSICAL FILTER : ORDERS.O_ORDERSTATUS = 'F' AND ORDERS.O_ORDERDATE >=
DATE'1996-01-01' AND ORDERS.O_ORDERDATE < DATE'1996-02-01'
             LOGICAL FILTER : ORDERS.O_COMMENT LIKE '%Customer%Complaints%'
     4  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     5  -  TARGET : LINEITEM.L_ORDERKEY
     6  -  GROUP KEY : LINEITEM.L_ORDERKEY
```

```
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             LOGICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     7  -  READ INDEX COLUMN : LINEITEM.L_ORDERKEY
           READ TABLE COLUMN : LINEITEM.L_QUANTITY, LINEITEM.L_SHIPDATE,
LINEITEM.L_COMMITDATE, LINEITEM.L_RECEIPTDATE
             MIN RANGE : LINEITEM.L_ORDERKEY = {ORDERS.O_ORDERKEY}
             MAX RANGE : LINEITEM.L_ORDERKEY = {ORDERS.O_ORDERKEY}
             PHYSICAL TABLE FILTER : LINEITEM.L_SHIPDATE >= DATE'1996-01-01' AND
LINEITEM.L_SHIPDATE < DATE'1996-02-01' AND LINEITEM.L_RECEIPTDATE > LINEITEM.L_COMMITDATE
<<<  end print plan
```

The hint is ignored so it is unnested by anti-semi join method, then performed in nested loop join metho d.

NL_AJ

If NL_AJ hint is specified, an optimizer unnests the subquery in anti-semi join form, and performs the join in nested loop method.

It is operated in the same way as that of UNNEST_NL hint. However, UNNEST_NL hint is applied to both s emi join and anti-semi join, on the other hand, NL_AJ hint is applied only to anti-semi join. Therefore, if t he hint above is specified on a subquery which is unnested with semi join, then an optimizer ignores it.

The following is an example of using NL_AJ hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey NOT IN (
                           SELECT   /*+ NL_AJ */
                                    l_orderkey
                             FROM   lineitem
                            GROUP BY l_orderkey
                            HAVING   sum(l_quantity) > 300
                          )
;
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                                                      |
--------------------------------------------------------------------------------
```

```
|    0  |  SELECT STATEMENT                                          |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                               |
|    2  |      NESTED JOIN (ANTI SEMI)                               |
|    3  |        TABLE ACCESS ("ORDERS")                             |
|    4  |        INLINE_VIEW ("$V5")                                 |
|    5  |          QUERY BLOCK ("$QB_IDX_6")                         |
|    6  |            GROUP                                           |
|    7  |              INDEX ACCESS ("LINEITEM", "LINEITEM_PK_INDEX")   |
=============================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
     4  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     5  -  TARGET : LINEITEM.L_ORDERKEY
     6  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             LOGICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     7  -  READ INDEX COLUMN : LINEITEM.L_ORDERKEY
           READ TABLE COLUMN : LINEITEM.L_QUANTITY
             MIN RANGE : LINEITEM.L_ORDERKEY = {ORDERS.O_ORDERKEY}
             MAX RANGE : LINEITEM.L_ORDERKEY = {ORDERS.O_ORDERKEY}
<<<  end print plan
```

The following is an example of using NL_AJ hint in a subquery which can be unnested only in semi join fo rm.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_comment IN (
                      SELECT   /*+ NL_AJ */
                               l_comment
                        FROM   lineitem
                       WHERE   l_shipdate >= date '1996-01-01'
                         AND   l_shipdate < date '1996-02-01'
                    )
;
>>>  start print plan
< Execution Plan >
```

```
=============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
|    2  |      HASH JOIN (SEMI)                                             |
|    3  |        TABLE ACCESS ("ORDERS")                                    |
|    4  |        HASH JOIN INSTANT (UNIQUE)                                 |
|    5  |          TABLE ACCESS ("LINEITEM")                                |
=============================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  READ COLUMN : ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE, ORDERS.O_COMMENT
     4  -  HASH KEY : LINEITEM.L_COMMENT
           READ KEY COLUMN : LINEITEM.L_COMMENT
             HASH FILTER : LINEITEM.L_COMMENT = ORDERS.O_COMMENT
           FETCH ONE ROW
     5  -  READ COLUMN : LINEITEM.L_SHIPDATE, LINEITEM.L_COMMENT
             PHYSICAL FILTER : LINEITEM.L_SHIPDATE >= DATE'1996-01-01' AND LINEITEM.L_SHIPDATE
< DATE'1996-02-01'
<<<  end print plan
```

The hint is ignored so it is unnested by semi join method, then performed in hash join method. Also, the semi join is converted into an inner join because both o_orderkey and l_orderkey are primary keys.

INL_AJ

If INL_AJ hint is specified, an optimizer unnests the subquery in anti-semi join form, and performs the join in instant nested loop join method.

It is operated in the same way as that of UNNEST_INL hint. However, UNNEST_INL hint is applied to both semi join and anti-semi join, on the other hand, INL_AJ hint is applied only to anti-semi join. Therefore, if the hint above is specified on a subquery which is unnested with semi join, then an optimizer ignores it.

The following is an example of using INL_AJ hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey NOT IN (
                          SELECT   /*+ INL_AJ */
```

```
                              l_orderkey
                       FROM   lineitem
                      GROUP BY l_orderkey
                      HAVING   sum(l_quantity) > 300
                     )
;
>>>  start print plan
< Execution Plan >
============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
|    2  |      NESTED JOIN (ANTI SEMI)                                      |
|    3  |        TABLE ACCESS ("ORDERS")                                    |
|    4  |        SORT JOIN INSTANT                                          |
|    5  |          INLINE_VIEW ("$V6")                                      |
|    6  |            QUERY BLOCK ("$QB_IDX_6")                              |
|    7  |              GROUP HASH INSTANT                                   |
|    8  |                TABLE ACCESS ("LINEITEM")                          |
============================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
     4  -  SORT KEY : "$V6.L_ORDERKEY ASC NULLS LAST"
           READ KEY COLUMN : $V6.L_ORDERKEY
             MIN RANGE : $V6.L_ORDERKEY = {ORDERS.O_ORDERKEY}
             MAX RANGE : $V6.L_ORDERKEY = {ORDERS.O_ORDERKEY}
     5  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     6  -  TARGET : LINEITEM.L_ORDERKEY
     7  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     8  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
<<<  end print plan
```

The following is an example of using INL_AJ hint in a subquery which can be unnested only in semi join form.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey IN (
                       SELECT   /*+ INL_AJ */
                                l_orderkey
                         FROM   lineitem
                       GROUP BY l_orderkey
                       HAVING   sum(l_quantity) > 300
                     )
;
>>>  start print plan
< Execution Plan >
============================================================================
|  IDX  |  NODE DESCRIPTION                                                |
----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                     |
|    2  |      NESTED JOIN (INNER JOIN)                                    |
|    3  |        INLINE_VIEW ("$V4")                                       |
|    4  |          QUERY BLOCK ("$QB_IDX_6")                               |
|    5  |            GROUP HASH INSTANT                                    |
|    6  |              TABLE ACCESS ("LINEITEM")                           |
|    7  |        INDEX ACCESS ("ORDERS", "ORDERS_PK_INDEX")                |
============================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     4  -  TARGET : LINEITEM.L_ORDERKEY
     5  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     6  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
     7  -  READ INDEX COLUMN : ORDERS.O_ORDERKEY
           READ TABLE COLUMN : ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
             MIN RANGE : ORDERS.O_ORDERKEY = {$V4.L_ORDERKEY}
             MAX RANGE : ORDERS.O_ORDERKEY = {$V4.L_ORDERKEY}
```

```
           FETCH ONE ROW
<<<  end print plan
```

The hint is ignored so it is unnested by semi join method, then performed in nested loop join method. Als o, the semi join is converted into an inner join because both o_orderkey and l_orderkey are primary keys.

MERGE_AJ

If MERGE_AJ hint is specified, an optimizer unnests the subquery in anti-semi join form, and performs the join in merge join method.

It is operated in the same way as that of UNNEST_MERGE hint. However, UNNEST_MERGE hint is applied to both semi join and anti-semi join, on the other hand, MERGE_AJ hint is applied only to anti-semi join. Therefore, if the hint above is specified on a subquery which is unnested with semi join, then an optimize r ignores it.

The following is an example of using MERGE_AJ hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey NOT IN (
                           SELECT   /*+ MERGE_AJ */
                                    l_orderkey
                             FROM   lineitem
                            GROUP BY l_orderkey
                            HAVING   sum(l_quantity) > 300
                          )
;
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                  ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                    ¦
¦    2  ¦      MERGE JOIN (ANTI SEMI)                                     ¦
¦    3  ¦        INDEX ACCESS ("ORDERS", "ORDERS_PK_INDEX")               ¦
¦    4  ¦        SORT JOIN INSTANT (UNIQUE)                               ¦
¦    5  ¦          INLINE_VIEW ("$V6")                                    ¦
¦    6  ¦            QUERY BLOCK ("$QB_IDX_6")                            ¦
```

```
|    7  |              GROUP HASH INSTANT                              |
|    8  |                TABLE ACCESS ("LINEITEM")                     |
=========================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
             ON FILTER (Equi) : ORDERS.O_ORDERKEY = $V6.L_ORDERKEY
     3  -  READ INDEX COLUMN : ORDERS.O_ORDERKEY
           READ TABLE COLUMN : ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
     4  -  SORT KEY : "$V6.L_ORDERKEY ASC NULLS LAST"
           READ KEY COLUMN : $V6.L_ORDERKEY
             MIN RANGE : $V6.L_ORDERKEY >= {ORDERS.O_ORDERKEY}
             MAX RANGE : $V6.L_ORDERKEY IS NOT NULL
     5  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     6  -  TARGET : LINEITEM.L_ORDERKEY
     7  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     8  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
<<<  end print plan
```

The following is an example of using MERGE_AJ hint in a subquery which can be unnested only in semi j oin form.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey IN (
                      SELECT   /*+ MERGE_AJ */
                               l_orderkey
                        FROM   lineitem
                      GROUP BY l_orderkey
                      HAVING   sum(l_quantity) > 300
                    )
;
>>>  start print plan
< Execution Plan >
=========================================================================
```

```
¦  IDX  ¦  NODE DESCRIPTION                                          ¦
-------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                          ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                             ¦
¦    2  ¦      NESTED JOIN (INNER JOIN)                            ¦
¦    3  ¦        INLINE_VIEW ("$V4")                               ¦
¦    4  ¦          QUERY BLOCK ("$QB_IDX_6")                       ¦
¦    5  ¦            GROUP HASH INSTANT                            ¦
¦    6  ¦              TABLE ACCESS ("LINEITEM")                   ¦
¦    7  ¦        INDEX ACCESS ("ORDERS", "ORDERS_PK_INDEX")        ¦
=========================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     4  -  TARGET : LINEITEM.L_ORDERKEY
     5  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     6  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
     7  -  READ INDEX COLUMN : ORDERS.O_ORDERKEY
           READ TABLE COLUMN : ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
             MIN RANGE : ORDERS.O_ORDERKEY = {$V4.L_ORDERKEY}
             MAX RANGE : ORDERS.O_ORDERKEY = {$V4.L_ORDERKEY}
           FETCH ONE ROW
<<<  end print plan
```

The hint is ignored so it is unnested by semi join method, then performed in nested loop join method. Als o, the semi join is converted into an inner join because both o_orderkey and l_orderkey are primary keys.

HASH_AJ

If HASH_AJ hint is specified, an optimizer unnests the subquery in anti-semi join form, and performs the j oin in hash join method.

It is operated in the same way as that of UNNEST_HASH hint. However, UNNEST_HASH hint is applied to both semi join and anti-semi join, on the other hand, HASH_AJ hint is applied only to anti-semi join. Ther efore, if the hint above is specified on a subquery which is unnested with semi join, then an optimizer ign ores it.

The following is an example of using HASH_AJ hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey NOT IN (
                           SELECT   /*+ HASH_AJ */
                                    l_orderkey
                             FROM   lineitem
                            GROUP BY l_orderkey
                            HAVING   sum(l_quantity) > 300
                          )
;
>>>  start print plan
< Execution Plan >
============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
|    2  |      HASH JOIN (ANTI SEMI)                                        |
|    3  |        TABLE ACCESS ("ORDERS")                                    |
|    4  |        HASH JOIN INSTANT (UNIQUE)                                 |
|    5  |          INLINE_VIEW ("$V6")                                      |
|    6  |            QUERY BLOCK ("$QB_IDX_6")                              |
|    7  |              GROUP HASH INSTANT                                   |
|    8  |                TABLE ACCESS ("LINEITEM")                          |
============================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
     4  -  HASH KEY : $V6.L_ORDERKEY
           READ KEY COLUMN : $V6.L_ORDERKEY
             HASH FILTER : $V6.L_ORDERKEY = ORDERS.O_ORDERKEY
           FETCH ONE ROW
     5  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     6  -  TARGET : LINEITEM.L_ORDERKEY
     7  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
```

```
             PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     8  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
<<<  end print plan
```

The following is an example of using HASH_AJ hint in a subquery which can be unnested only in semi join form.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey IN (
                      SELECT   /*+ HASH_AJ */
                               l_orderkey
                        FROM   lineitem
                      GROUP BY l_orderkey
                      HAVING   sum(l_quantity) > 300
                    )
;
>>>  start print plan
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
|    2  |      NESTED JOIN (INNER JOIN)                                     |
|    3  |        INLINE_VIEW ("$V4")                                        |
|    4  |          QUERY BLOCK ("$QB_IDX_6")                                |
|    5  |            GROUP HASH INSTANT                                     |
|    6  |              TABLE ACCESS ("LINEITEM")                            |
|    7  |        INDEX ACCESS ("ORDERS", "ORDERS_PK_INDEX")                 |
=============================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     4  -  TARGET : LINEITEM.L_ORDERKEY
     5  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
```

```
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     6  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
     7  -  READ INDEX COLUMN : ORDERS.O_ORDERKEY
           READ TABLE COLUMN : ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
             MIN RANGE : ORDERS.O_ORDERKEY = {$V4.L_ORDERKEY}
             MAX RANGE : ORDERS.O_ORDERKEY = {$V4.L_ORDERKEY}
           FETCH ONE ROW
<<<  end print plan
```

The hint is ignored so it is unnested by semi join method, then performed in nested loop join method. Als o, the semi join is converted into an inner join because both o_orderkey and l_orderkey are primary keys.

〈unnest join driver hints〉

Those hints are available when performing subquery unnesting in the cluster system. They are ignored in the standalone system.

LOCAL_UNNEST

If LOCAL_UNNEST hint is specified, an optimizer transforms the subquery into the join statement which g uarantees the same result, then performs the join in a local. In other words, it performs the join by bring r esults of both a left child and a right child to the local.

The following is an example of using LOCAL_UNNEST hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey IN (
                       SELECT   /*+ LOCAL_UNNEST */
                                l_orderkey
                         FROM   lineitem
                       GROUP BY l_orderkey
                       HAVING   sum(l_quantity) > 300
                     )
;
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                  ¦
-----------------------------------------------------------------------------
```

```
┆    0  ┆  SELECT STATEMENT                                          ┆
┆    1  ┆    QUERY BLOCK ("$QB_IDX_2")                               ┆
┆    2  ┆      HASH JOIN (INNER JOIN)                                ┆
┆    3  ┆        PLAN BASED CLUSTER                                  ┆
┆    4  ┆          TABLE ACCESS ("ORDERS")                           ┆
┆    5  ┆        HASH JOIN INSTANT                                   ┆
┆    6  ┆          PLAN BASED CLUSTER                                ┆
┆    7  ┆            INLINE_VIEW ("$V8")                             ┆
┆    8  ┆              QUERY BLOCK ("$QB_IDX_6")                     ┆
┆    9  ┆                GROUP HASH INSTANT                          ┆
┆   10  ┆                  TABLE ACCESS ("LINEITEM")                 ┆
=====================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  SQL : SELECT /*+ FULL( _A1 ) */ "_A1"."O_ORDERKEY", "_A1"."O_TOTALPRICE",
"_A1"."O_ORDERDATE" FROM "PUBLIC"."ORDERS"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 0 rows, G2(G2N1,G2N2) 0 rows, G3(G3N1,G3N2) 0 rows
     4  -  HASH SHARD ( # 3 )
           READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
     5  -  HASH KEY : $V8.L_ORDERKEY
           READ KEY COLUMN : $V8.L_ORDERKEY
             HASH FILTER : $V8.L_ORDERKEY = ORDERS.O_ORDERKEY
     6  -  SQL : SELECT /*+ NO_MERGE( _A1 ) */ * FROM ( SELECT /*+ USE_GROUP_HASH(10) FULL(
_A2 ) */ "_A2"."L_ORDERKEY" FROM "PUBLIC"."LINEITEM"@LOCAL AS "_A2" GROUP BY
"_A2"."L_ORDERKEY" HAVING SUM( "_A2"."L_QUANTITY" ) > :_V0 ) AS "_A1"("L_ORDERKEY")
           TARGET DOMAIN : G1(G1N1,G1N2) 0 rows, G2(G2N1,G2N2) 0 rows, G3(G3N1,G3N2) 0 rows
     7  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     8  -  TARGET : LINEITEM.L_ORDERKEY
     9  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
    10  -  HASH SHARD ( # 3 )
           READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
<<<  end print plan
```

In the execution plan above, HASH JOIN (INNER JOIN) is performed in local by bringing TABLE ACCESS ("ORDERS") result from PLAN BASED CLUSTER of IDX 3 and bringing INLINE_VIEW ("$V8") result from PL AN BASED CLUSTER of IDX 6.

REMOTE_UNNEST

If REMOTE_UNNEST hint is specified, an optimizer transforms the subquery into the join statement which guarantees the same result. Then it performs the join in remote. In other words, it performs the join in ea ch group.

The following is an example of using REMOTE_UNNEST hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey IN (
                      SELECT   /*+ REMOTE_UNNEST */
                               l_orderkey
                        FROM   lineitem
                      GROUP BY l_orderkey
                      HAVING   sum(l_quantity) > 300
                     )
;
>>>  start print plan
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
|    2  |      PLAN BASED CLUSTER                                           |
|    3  |        NESTED JOIN (INNER JOIN)                                   |
|    4  |          INLINE_VIEW ("$V5")                                      |
|    5  |            QUERY BLOCK ("$QB_IDX_6")                              |
|    6  |              GROUP HASH INSTANT                                   |
|    7  |                TABLE ACCESS ("LINEITEM")                          |
|    8  |          INDEX ACCESS ("ORDERS", "ORDERS_PK_INDEX")               |
=============================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE USE_NL_IN( _A1 ) NO_MERGE( _A2 ) INDEX( _A1,
"PUBLIC"."ORDERS_PK_INDEX" ) */ "_A1"."O_ORDERDATE", "_A1"."O_TOTALPRICE" FROM ( ( SELECT /*+
USE_GROUP_HASH(10) FULL( _A3 ) */ "_A3"."L_ORDERKEY" FROM "PUBLIC"."LINEITEM"@LOCAL AS "_A3"
GROUP BY "_A3"."L_ORDERKEY" HAVING SUM( "_A3"."L_QUANTITY" ) > :_V0 ) AS "_A2"("L_ORDERKEY")
INNER JOIN "PUBLIC"."ORDERS"@LOCAL AS "_A1" ON true ) ALIAS "_A4" WHERE "_A1"."O_ORDERKEY" =
```

```
"_A2"."L_ORDERKEY"
           TARGET DOMAIN : G1(G1N1,G1N2) 0 rows, G2(G2N1,G2N2) 0 rows, G3(G3N1,G3N2) 0 rows
     3  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     4  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     5  -  TARGET : LINEITEM.L_ORDERKEY
     6  -  GROUP KEY : LINEITEM.L_ORDERKEY
           RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
           READ RECORD COLUMN : SUM( LINEITEM.L_QUANTITY )
             PHYSICAL FILTER : SUM( LINEITEM.L_QUANTITY ) > 300
     7  -  HASH SHARD ( # 3 )
           READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_QUANTITY
     8  -  HASH SHARD ( # 3 )
           READ INDEX COLUMN : ORDERS.O_ORDERKEY
           READ TABLE COLUMN : ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
             MIN RANGE : ORDERS.O_ORDERKEY = {$V5.L_ORDERKEY}
             MAX RANGE : ORDERS.O_ORDERKEY = {$V5.L_ORDERKEY}
           FETCH ONE ROW
<<<  end print plan
```

In the execution plan above, the join is performed in each group and join results are collected from PLAN BASED CLUSTER of IDX 2, then returned.

〈unnest join pusher hints〉

Those hints are available when performing subquery unnesting in the cluster system. They are ignored in the standalone system.

PUSHER_SUBQ

If PUSHER_SUBQ hint is specified, an optimizer transforms the subquery into the join statement which gu arantees the same result. Then, it builds the view or the table which was created by unnesting the subqu ery as a pusher table when performing the join in a remote.

The following is an example of using PUSHER_SUBQ hint.

```
\EXPLAIN PLAN
SELECT p_name
  FROM part
 WHERE p_partkey IN ( SELECT /*+ PUSHER_SUBQ */
                             l_partkey
                        FROM lineitem
                       WHERE l_shipdate = date'1998-12-01'
                    );
```

```
< Execution Plan >
============================================================================
¦IDX ¦  NODE DESCRIPTION                                          ¦ ROWS ¦
----------------------------------------------------------------------------
¦  0 ¦  SELECT STATEMENT                                          ¦  18 ¦
¦  1 ¦    QUERY BLOCK ("$QB_IDX_2")                               ¦  18 ¦
¦  2 ¦      SINGLE CLUSTER                                ¦ LOCAL/REMOTE 18 ¦
¦  3 ¦        CLUSTER PUSHER ("_$NI_6")                           ¦  18 ¦
¦  4 ¦          PLAN BASED CLUSTER                        ¦ LOCAL/REMOTE 18 ¦
¦  5 ¦            TABLE ACCESS ("LINEITEM")                       ¦   7 ¦
¦  6 ¦        SELECT STATEMENT                                    ¦   5 ¦
¦  7 ¦          QUERY BLOCK ("$QB_IDX_2")                         ¦   5 ¦
¦  8 ¦            NESTED JOIN (INVERTED SEMI)                     ¦   5 ¦
¦  9 ¦              SORT JOIN INSTANT (UNIQUE)                    ¦   5 ¦
¦ 10 ¦                PUSHER TABLE ACCESS ("_$NI_6" AS _A2)       ¦   5 ¦
¦ 11 ¦              INDEX ACCESS ("PART" AS _A1, "PART_PK_INDEX") ¦   5 ¦
============================================================================
     1  -  TARGET : PART.P_NAME
     2  -  SQL :
           SELECT /*+ KEEP_JOINED_TABLE USE_NL_IN( _A1 )
                      FULL( _A2 )
                      INDEX( _A1, "PUBLIC"."PART_PK_INDEX" )
                  */
                  "_A1"."P_NAME"
           FROM ( "PUBLIC"."PART"@"G1N1"¦"G1N2"¦"G2N1"¦"G2N2"¦"G3N1"¦"G3N2"
                   AS "_A1"
                  SEMI JOIN
                  "SESSION_SCHEMA"."_$NI_6"@LOCAL AS "_A2"
                  ON "_A1"."P_PARTKEY" = "_A2"."L_PARTKEY"
                ) ALIAS "_A3"
           TARGET DOMAIN : G1(G1N1,G1N2) 5 rows,
                           G2(G2N1,G2N2) 9 rows,
                           G3(G3N1,G3N2) 4 rows
     3  -  SQL : DECLARE INSTANT TABLE "SESSION_SCHEMA"."_$NI_6"
                 ( "L_PARTKEY" NUMBER(10, 0) )
           COLUMN : LINEITEM.L_PARTKEY AS L_PARTKEY
           SHARDED : LINEITEM.L_PARTKEY
           TARGET DOMAIN : G1(G1N1,G1N2) 5 rows,
                           G2(G2N1,G2N2) 9 rows,
                           G3(G3N1,G3N2) 4 rows
     4  -  SQL :
```

```
           SELECT /*+ FULL( _A1 ) */
                  "_A1"."L_PARTKEY"
             FROM "PUBLIC"."LINEITEM"@LOCAL AS "_A1"
            WHERE "_A1"."L_SHIPDATE" = :_V0
           TARGET DOMAIN : G1(G1N1,G1N2) 7 rows,
                           G2(G2N1,G2N2) 4 rows,
                           G3(G3N1,G3N2) 7 rows
     5  -  HASH SHARD ( # 3 )
           READ COLUMN : LINEITEM.L_PARTKEY, LINEITEM.L_SHIPDATE
             PHYSICAL FILTER : LINEITEM.L_SHIPDATE = DATE'1998-12-01'
     7  -  TARGET : _A1.P_NAME
     8  -  JOINED COLUMN : _A1.P_NAME
     9  -  SORT KEY : "_A2.L_PARTKEY ASC NULLS LAST"
           READ KEY COLUMN : _A2.L_PARTKEY
    10  -  READ COLUMN : _A2.L_PARTKEY
    11  -  HASH SHARD ( # 3 )
           READ INDEX COLUMN : _A1.P_PARTKEY
           READ TABLE COLUMN : _A1.P_NAME
             MIN RANGE : _A1.P_PARTKEY = {_A2.L_PARTKEY}
             MAX RANGE : _A1.P_PARTKEY = {_A2.L_PARTKEY}
           FETCH ONE ROW
<<<  end print plan
```

In [IDX 3] of the execution plan above, l_partkey creates a pusher table which is a shard key and the data read from [IDX 4] is stored. Then, remote semi joined is performed for part and the pusher table.

NO_PUSHER_SUBQ

If NO_PUSHER_SUBQ hint is specified, an optimizer transforms the subquery into the join statement whic h guarantees the same result. Then, it prevents to build the view or the table which was created by unnes ting the subquery as a pusher table when performing the join in a remote.

Therefore, the remote unnest may not be available, and a sibling table can be built as a pusher table.

The following is an example of using NO_PUSHER_SUBQ hint.

```
\EXPLAIN PLAN
SELECT p_name
  FROM part
 WHERE p_partkey IN ( SELECT /*+ NO_PUSHER_SUBQ */
                              l_partkey
                        FROM lineitem
                       WHERE l_shipdate = date'1998-12-01'
```

```
                    );
>>>  start print plan
< Execution Plan >
================================================================================
|IDX|  NODE DESCRIPTION                          |                ROWS |
--------------------------------------------------------------------------------
| 0 |  SELECT STATEMENT                          |                  18 |
| 1 |    QUERY BLOCK ("$QB_IDX_2")               |                  18 |
| 2 |      HASH JOIN (SEMI)                      |                  18 |
| 3 |        PLAN BASED CLUSTER                  | LOCAL/REMOTE 200000 |
| 4 |          TABLE ACCESS ("PART")             |               66675 |
| 5 |        HASH JOIN INSTANT (UNIQUE)          |                  18 |
| 6 |          PLAN BASED CLUSTER                |     LOCAL/REMOTE 18 |
| 7 |            TABLE ACCESS ("LINEITEM")       |                   7 |
================================================================================
     1  -  TARGET : PART.P_NAME
     2  -  JOINED COLUMN : PART.P_NAME
     3  -  SQL : SELECT /*+ FULL( _A1 ) */
                        "_A1"."P_PARTKEY", "_A1"."P_NAME"
                   FROM "PUBLIC"."PART"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 66675 rows,
                           G2(G2N1,G2N2) 66664 rows,
                           G3(G3N1,G3N2) 66661 rows
     4  -  HASH SHARD ( # 3 )
           READ COLUMN : PART.P_PARTKEY, PART.P_NAME
     5  -  HASH KEY : LINEITEM.L_PARTKEY
           READ KEY COLUMN : LINEITEM.L_PARTKEY
             HASH FILTER : LINEITEM.L_PARTKEY = PART.P_PARTKEY
           FETCH ONE ROW
     6  -  SQL : SELECT /*+ FULL( _A1 ) */
                        "_A1"."L_PARTKEY"
                   FROM "PUBLIC"."LINEITEM"@LOCAL AS "_A1"
                  WHERE "_A1"."L_SHIPDATE" = :_V0
           TARGET DOMAIN : G1(G1N1,G1N2) 7 rows,
                           G2(G2N1,G2N2) 4 rows,
                           G3(G3N1,G3N2) 7 rows
     7  -  HASH SHARD ( # 3 )
           READ COLUMN : LINEITEM.L_PARTKEY, LINEITEM.L_SHIPDATE
             PHYSICAL FILTER : LINEITEM.L_SHIPDATE = DATE'1998-12-01'
<<<  end print plan
```

In the execution above, remote unnest is not performed, but local unnest is performed. In other words, it reads the data satisfying conditions from part and lineitem to G1, G2 and G3, then performs semi join in the current driver server.

The following is an example of using REMOTE_UNNEST hint and NO_PUSHER_SUBQ hint together.

```
\EXPLAIN PLAN
SELECT p_name
  FROM part
 WHERE p_partkey IN ( SELECT /*+ REMOTE_UNNEST NO_PUSHER_SUBQ */
                              l_partkey
                       FROM lineitem
                      WHERE l_shipdate = date'1998-12-01'
                    );
>>>  start print plan
< Execution Plan >
================================================================================
| IDX|  NODE DESCRIPTION                                              |   ROWS |
--------------------------------------------------------------------------------
|  0 |  SELECT STATEMENT                                              |     18 |
|  1 |    QUERY BLOCK ("$QB_IDX_2")                                   |     18 |
|  2 |      MULTIPLE CLUSTER                            |     LOCAL/REMOTE 18 |
|  3 |        CLUSTER PUSHER ("_$NI_5")                               | 200000 |
|  4 |          PLAN BASED CLUSTER                      | LOCAL/REMOTE 200000 |
|  5 |            TABLE ACCESS ("PART")                               |  66675 |
|  6 |        SELECT STATEMENT                                        |      7 |
|  7 |          QUERY BLOCK ("$QB_IDX_2")                             |      7 |
|  8 |            SORT INSTANT                                        |      7 |
|  9 |              HASH JOIN (SEMI)                                  |      7 |
| 10 |                PUSHER TABLE ACCESS ("_$NI_5" AS _A2)           | 200000 |
| 11 |                HASH JOIN INSTANT (UNIQUE)                      |      7 |
| 12 |                  TABLE ACCESS ("LINEITEM" AS _A1)              |      7 |
================================================================================
     1  -  TARGET : _$NI_5.P_NAME
     2  -  SQL :
           SELECT /*+ KEEP_JOINED_TABLE
                      USE_HASH_IN( _A1, 396 )
                      FULL( _A2 )
                      FULL( _A1 )
                 */
                 "_A2"."P_PARTKEY", "_A2"."P_NAME"
            FROM ( "SESSION_SCHEMA"."_$NI_5"@LOCAL AS "_A2"
```

```
                SEMI JOIN
                "PUBLIC"."LINEITEM"@"G1N1"¦"G1N2"¦
                                    "G2N1"¦"G2N2"¦
                                    "G3N1"¦"G3N2"
                AS "_A1"
                ON  "_A1"."L_PARTKEY" = "_A2"."P_PARTKEY"
                AND "_A1"."L_SHIPDATE" = :_V0
              ) ALIAS "_A3"
        ORDER BY "_A2"."P_PARTKEY" ASC NULLS LAST
        TARGET DOMAIN : G1(G1N1,G1N2) 7 rows,
                        G2(G2N1,G2N2) 4 rows,
                        G3(G3N1,G3N2) 7 rows
        DISTINCT KEY GROUP
          KEY GROUP : _$NI_5.P_PARTKEY
 3  -  SQL : DECLARE INSTANT TABLE "SESSION_SCHEMA"."_$NI_5"
             ( "P_PARTKEY" NUMBER(10, 0), "P_NAME" VARCHAR(55 OCTETS) )
        COLUMN : PART.P_PARTKEY AS P_PARTKEY, PART.P_NAME AS P_NAME
        CLONED
        TARGET DOMAIN : G1(G1N1,G1N2) 200000 rows,
                        G2(G2N1,G2N2) 200000 rows,
                        G3(G3N1,G3N2) 200000 rows
 4  -  SQL :
        SELECT /*+ FULL( _A1 ) */
               "_A1"."P_PARTKEY", "_A1"."P_NAME"
          FROM "PUBLIC"."PART"@LOCAL AS "_A1"
        TARGET DOMAIN : G1(G1N1,G1N2) 66675 rows,
                        G2(G2N1,G2N2) 66664 rows,
                        G3(G3N1,G3N2) 66661 rows
 5  -  HASH SHARD ( # 3 )
        READ COLUMN : PART.P_PARTKEY, PART.P_NAME
 7  -  TARGET : _A2.P_PARTKEY, _A2.P_NAME
 8  -  SORT KEY : "_A2.P_PARTKEY ASC NULLS LAST"
        RECORD COLUMN : _A2.P_NAME
        READ KEY COLUMN : _A2.P_PARTKEY
        READ RECORD COLUMN : _A2.P_NAME
 9  -  JOINED COLUMN : _A2.P_PARTKEY, _A2.P_NAME
10  -  READ COLUMN : _A2.P_PARTKEY, _A2.P_NAME
11  -  HASH KEY : _A1.L_PARTKEY
        READ KEY COLUMN : _A1.L_PARTKEY
          HASH FILTER : _A1.L_PARTKEY = _A2.P_PARTKEY
        FETCH ONE ROW
```

```
    12  -  HASH SHARD ( # 3 )
           READ COLUMN : _A1.L_PARTKEY, _A1.L_SHIPDATE
             PHYSICAL FILTER : _A1.L_SHIPDATE = :_V0
<<<  end print plan
```

In the execution plan above, it can not accumulate a subquery in a pusher table, so it creates part as a clo ned pusher table, then performs remote join.

PUSHER_OUTQ

If PUSHER_OUTQ hint is specified, an optimizer transforms the subquery into the join statement which gu arantees the same result. Then, it accumulates a subquery of an outer table in a pusher table when perfor ming the join in a remote.

The following is an example of using PUSHER_OUTQ hint.

```
\EXPLAIN PLAN
SELECT p_name
  FROM part
 WHERE p_partkey IN ( SELECT /*+ PUSHER_OUTQ */
                              l_partkey
                       FROM lineitem
                      WHERE l_shipdate = date'1998-12-01'
                    );
< Execution Plan >
================================================================================
¦IDX¦  NODE DESCRIPTION                              ¦                ROWS ¦
--------------------------------------------------------------------------------
¦ 0 ¦  SELECT STATEMENT                              ¦                  18 ¦
¦ 1 ¦    QUERY BLOCK ("$QB_IDX_2")                   ¦                  18 ¦
¦ 2 ¦      HASH JOIN (SEMI)                          ¦                  18 ¦
¦ 3 ¦        PLAN BASED CLUSTER                      ¦ LOCAL/REMOTE 200000 ¦
¦ 4 ¦          TABLE ACCESS ("PART")                 ¦               66675 ¦
¦ 5 ¦        HASH JOIN INSTANT (UNIQUE)              ¦                  18 ¦
¦ 6 ¦          PLAN BASED CLUSTER                    ¦     LOCAL/REMOTE 18 ¦
¦ 7 ¦            TABLE ACCESS ("LINEITEM")           ¦                   7 ¦
================================================================================
     1  -  TARGET : PART.P_NAME
     2  -  JOINED COLUMN : PART.P_NAME
     3  -  SQL : SELECT /*+ FULL( _A1 ) */
                         "_A1"."P_PARTKEY", "_A1"."P_NAME"
                  FROM "PUBLIC"."PART"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 66675 rows,
```

```
                        G2(G2N1,G2N2) 66664 rows,
                        G3(G3N1,G3N2) 66661 rows
    4  -  HASH SHARD ( # 3 )
          READ COLUMN : PART.P_PARTKEY, PART.P_NAME
    5  -  HASH KEY : LINEITEM.L_PARTKEY
          READ KEY COLUMN : LINEITEM.L_PARTKEY
            HASH FILTER : LINEITEM.L_PARTKEY = PART.P_PARTKEY
          FETCH ONE ROW
    6  -  SQL : SELECT /*+ FULL( _A1 ) */
                       "_A1"."L_PARTKEY"
                  FROM "PUBLIC"."LINEITEM"@LOCAL AS "_A1"
                 WHERE "_A1"."L_SHIPDATE" = :_V0
          TARGET DOMAIN : G1(G1N1,G1N2) 7 rows,
                        G2(G2N1,G2N2) 4 rows,
                        G3(G3N1,G3N2) 7 rows
    7  -  HASH SHARD ( # 3 )
          READ COLUMN : LINEITEM.L_PARTKEY, LINEITEM.L_SHIPDATE
            PHYSICAL FILTER : LINEITEM.L_SHIPDATE = DATE'1998-12-01'
<<<  end print plan
```

PUSHER_OUTQ hint is applied when it is remote semi join.
If the cost of local semi join is better than that of remote semi join to which PUSHER_OUTQ was applied, then local semi join is selected. Therefore, whether the hint is applied can not be checked in the executio n plan.

Therefore, use REMOTE_UNNEST hint and PUSHER_OUTQ hint together as follows to check whether the hint is applied.

```
\EXPLAIN PLAN
SELECT p_name
  FROM part
 WHERE p_partkey IN ( SELECT /*+ REMOTE_UNNEST PUSHER_OUTQ */
                              l_partkey
                         FROM lineitem
                        WHERE l_shipdate = date'1998-12-01'
                      );
>>>  start print plan
< Execution Plan >
================================================================================
| IDX |  NODE DESCRIPTION                          |                ROWS |
--------------------------------------------------------------------------------
|   0 |  SELECT STATEMENT                          |                  18 |
```

```
┌  1 ┆    QUERY BLOCK ("$QB_IDX_2")                  ┆                 18 ┆
┆  2 ┆      MULTIPLE CLUSTER                         ┆     LOCAL/REMOTE 18 ┆
┆  3 ┆        CLUSTER PUSHER ("_$NI_5")              ┆             200000 ┆
┆  4 ┆          PLAN BASED CLUSTER                   ┆ LOCAL/REMOTE 200000 ┆
┆  5 ┆            TABLE ACCESS ("PART")                                ┆  66675 ┆
┆  6 ┆        SELECT STATEMENT                                         ┆      7 ┆
┆  7 ┆          QUERY BLOCK ("$QB_IDX_2")                              ┆      7 ┆
┆  8 ┆            SORT INSTANT                                         ┆      7 ┆
┆  9 ┆              HASH JOIN (SEMI)                                   ┆      7 ┆
┆ 10 ┆                PUSHER TABLE ACCESS ("_$NI_5" AS _A2)             ┆ 200000 ┆
┆ 11 ┆                HASH JOIN INSTANT (UNIQUE)                       ┆      7 ┆
┆ 12 ┆                  TABLE ACCESS ("LINEITEM" AS _A1)               ┆      7 ┆
================================================================================
     1  -  TARGET : _$NI_5.P_NAME
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE
                            USE_HASH_IN( _A1, 396 )
                            FULL( _A2 )
                            FULL( _A1 )
                        */
                         "_A2"."P_PARTKEY", "_A2"."P_NAME"
                    FROM ( "SESSION_SCHEMA"."_$NI_5"@LOCAL AS "_A2"
                           SEMI JOIN
                           "PUBLIC"."LINEITEM"@"G1N1"¦"G1N2"¦
                                                "G2N1"¦"G2N2"¦
                                                "G3N1"¦"G3N2" AS "_A1"
                           ON  "_A1"."L_PARTKEY" = "_A2"."P_PARTKEY"
                           AND "_A1"."L_SHIPDATE" = :_V0
                        ) ALIAS "_A3"
                ORDER BY "_A2"."P_PARTKEY" ASC NULLS LAST
           TARGET DOMAIN : G1(G1N1,G1N2) 7 rows,
                           G2(G2N1,G2N2) 4 rows,
                           G3(G3N1,G3N2) 7 rows
           DISTINCT KEY GROUP
             KEY GROUP : _$NI_5.P_PARTKEY
     3  -  SQL : DECLARE INSTANT TABLE "SESSION_SCHEMA"."_$NI_5"
                    ( "P_PARTKEY" NUMBER(10, 0), "P_NAME" VARCHAR(55 OCTETS) )
           COLUMN : PART.P_PARTKEY AS P_PARTKEY, PART.P_NAME AS P_NAME
           CLONED
           TARGET DOMAIN : G1(G1N1,G1N2) 200000 rows,
                           G2(G2N1,G2N2) 200000 rows,
                           G3(G3N1,G3N2) 200000 rows
```

```
     4  -  SQL : SELECT /*+ FULL( _A1 ) */
                         "_A1"."P_PARTKEY", "_A1"."P_NAME"
                    FROM "PUBLIC"."PART"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 66675 rows,
                           G2(G2N1,G2N2) 66664 rows,
                           G3(G3N1,G3N2) 66661 rows
     5  -  HASH SHARD ( # 3 )
           READ COLUMN : PART.P_PARTKEY, PART.P_NAME
     7  -  TARGET : _A2.P_PARTKEY, _A2.P_NAME
     8  -  SORT KEY : "_A2.P_PARTKEY ASC NULLS LAST"
           RECORD COLUMN : _A2.P_NAME
           READ KEY COLUMN : _A2.P_PARTKEY
           READ RECORD COLUMN : _A2.P_NAME
     9  -  JOINED COLUMN : _A2.P_PARTKEY, _A2.P_NAME
    10  -  READ COLUMN : _A2.P_PARTKEY, _A2.P_NAME
    11  -  HASH KEY : _A1.L_PARTKEY
           READ KEY COLUMN : _A1.L_PARTKEY
             HASH FILTER : _A1.L_PARTKEY = _A2.P_PARTKEY
           FETCH ONE ROW
    12  -  HASH SHARD ( # 3 )
           READ COLUMN : _A1.L_PARTKEY, _A1.L_SHIPDATE
             PHYSICAL FILTER : _A1.L_SHIPDATE = :_V0
<<<  end print plan
```

In the execution plan above, it performs remote semi join and builds part which is a table of an outer que ry as a pusher table.

NO_PUSHER_OUTQ

If NO_PUSHER_OUTQ hint is specified, an optimizer transforms the subquery into the join statement whic h guarantees the same result. Then, it should not build an outer table of a subquery as a pusher table wh en performing the join in a remote.

The following is an example of using NO_PUSHER_OUTQ hint.

```
\EXPLAIN PLAN
SELECT p_name
  FROM part
 WHERE p_partkey IN ( SELECT /*+ NO_PUSHER_OUTQ */
                              l_partkey
                        FROM lineitem
                       WHERE l_shipdate = date'1998-12-01'
                    );
```

```
>>>  start print plan
< Execution Plan >
================================================================================
| IDX |  NODE DESCRIPTION                                              | ROWS |
--------------------------------------------------------------------------------
|  0 |  SELECT STATEMENT                                               |    1 |
|  1 |    QUERY BLOCK ("$QB_IDX_2")                                    |    1 |
|  2 |      SINGLE CLUSTER                                 |  LOCAL/REMOTE 1 |
|  3 |        CLUSTER PUSHER ("_$NI_7")                                |   18 |
|  4 |          PLAN BASED CLUSTER                         | LOCAL/REMOTE 18 |
|  5 |            TABLE ACCESS ("LINEITEM")                            |    7 |
|  6 |        SELECT STATEMENT                                         |    1 |
|  7 |          QUERY BLOCK ("$QB_IDX_2")                              |    1 |
|  8 |            AGGREGATION BY HASH                                  |    1 |
|  9 |              NESTED JOIN (INVERTED SEMI)                        |    5 |
| 10 |                SORT JOIN INSTANT (UNIQUE)                       |    5 |
| 11 |                  PUSHER TABLE ACCESS ("_$NI_7" AS _A2)          |    5 |
| 12 |                INDEX ACCESS ("PART" AS _A1, "PART_PK_INDEX") |(5) 5 |
================================================================================
     1  -  TARGET : COUNT(*)
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE
                            USE_NL_IN( _A1 )
                            FULL( _A2 )
                            INDEX( _A1, "PUBLIC"."PART_PK_INDEX" )
                        */
                        COUNT(*)
                  FROM ( "PUBLIC"."PART"@"G1N1"|"G1N2"|
                                        "G2N1"|"G2N2"|
                                        "G3N1"|"G3N2" AS "_A1"
                         SEMI JOIN
                        "SESSION_SCHEMA"."_$NI_7"@LOCAL AS "_A2"
                         ON "_A1"."P_PARTKEY" = "_A2"."L_PARTKEY"
                       ) ALIAS "_A3"
           TARGET DOMAIN : G1(G1N1,G1N2) 1 rows,
                           G2(G2N1,G2N2) 1 rows,
                           G3(G3N1,G3N2) 1 rows
           RE-AGGREGATION
             AGGREGATION : SUM( COUNT(*) )
     3  -  SQL : DECLARE INSTANT TABLE "SESSION_SCHEMA"."_$NI_7"
                 ( "L_PARTKEY" NUMBER(10, 0) )
           COLUMN : LINEITEM.L_PARTKEY AS L_PARTKEY
```

```
          SHARDED : LINEITEM.L_PARTKEY
          TARGET DOMAIN : G1(G1N1,G1N2) 5 rows,
                          G2(G2N1,G2N2) 9 rows,
                          G3(G3N1,G3N2) 4 rows
     4  -  SQL : SELECT /*+ FULL( _A1 ) */
                        "_A1"."L_PARTKEY"
                   FROM "PUBLIC"."LINEITEM"@LOCAL AS "_A1"
                  WHERE "_A1"."L_SHIPDATE" = :_V0
          TARGET DOMAIN : G1(G1N1,G1N2) 7 rows,
                          G2(G2N1,G2N2) 4 rows,
                          G3(G3N1,G3N2) 7 rows
     5  -  HASH SHARD ( # 3 )
          READ COLUMN : LINEITEM.L_PARTKEY, LINEITEM.L_SHIPDATE
            PHYSICAL FILTER : LINEITEM.L_SHIPDATE = DATE'1998-12-01'
     7  -  TARGET : COUNT(*)
     8  -  AGGREGATION : COUNT(*)
     9  -  JOINED COLUMN : NOTHING
    10  -  SORT KEY : "_A2.L_PARTKEY ASC NULLS LAST"
          READ KEY COLUMN : _A2.L_PARTKEY
    11  -  READ COLUMN : _A2.L_PARTKEY
    12  -  HASH SHARD ( # 3 )
          READ INDEX COLUMN : _A1.P_PARTKEY
            MIN RANGE : _A1.P_PARTKEY = {_A2.L_PARTKEY}
            MAX RANGE : _A1.P_PARTKEY = {_A2.L_PARTKEY}
          FETCH ONE ROW
<<<  end print plan
```

In the execution plan above, it does not build part which is a table of an outer query as a pusher table, bu t it builds lineitem which is a table whose subquery is unnested as a pusher table.

〈unnest merge hints〉

If a subquery is unnested as a view, then the hint defines whether to merge the view.

MERGE_SUBQ

If a subquery is unnested as a view, then the hint merges the view.

The following is an example of using MERGE_SUBQ hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
```

```
  FROM orders
 WHERE o_orderkey IN (
                       SELECT   /*+ MERGE_SUBQ */
                                l_orderkey
                         FROM   lineitem
                       GROUP BY l_orderkey
                     )
;
no rows selected.
>>>  start print plan
< Execution Plan >
============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
|    2  |      INLINE_VIEW ("$V3")                                          |
|    3  |        QUERY BLOCK ("$QB_IDX_6")                                  |
|    4  |          GROUP HASH INSTANT                                       |
|    5  |            HASH JOIN (INNER JOIN)                                 |
|    6  |              TABLE ACCESS ("ORDERS")                              |
|    7  |              HASH JOIN INSTANT                                    |
|    8  |                INDEX ACCESS ("LINEITEM", "LINEITEM_PK_INDEX")     |
============================================================================
     1  -  TARGET : $V3.O_ORDERDATE, $V3.O_TOTALPRICE
     2  -  COLUMN : O_TOTALPRICE AS O_TOTALPRICE, O_ORDERDATE AS O_ORDERDATE
     3  -  TARGET : MAX( ORDERS.O_TOTALPRICE ) AS O_TOTALPRICE, MAX( ORDERS.O_ORDERDATE ) AS
O_ORDERDATE
     4  -  GROUP KEY : LINEITEM.L_ORDERKEY, ORDERS.$PHYSICAL_ROWID
           RECORD COLUMN : MAX( ORDERS.O_TOTALPRICE ), MAX( ORDERS.O_ORDERDATE )
           READ RECORD COLUMN : MAX( ORDERS.O_TOTALPRICE ), MAX( ORDERS.O_ORDERDATE )
     5  -  JOINED COLUMN : LINEITEM.L_ORDERKEY, ORDERS.$PHYSICAL_ROWID, ORDERS.O_TOTALPRICE,
ORDERS.O_ORDERDATE
     6  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
     7  -  HASH KEY : LINEITEM.L_ORDERKEY
           READ KEY COLUMN : LINEITEM.L_ORDERKEY
             HASH FILTER : LINEITEM.L_ORDERKEY = ORDERS.O_ORDERKEY
     8  -  READ INDEX COLUMN : LINEITEM.L_ORDERKEY
<<<  end print plan
```

In the execution plan above, if a subquery is unnested as a view, then the complex view merging is perfor

med for the view.

NO_MERGE_SUBQ

If a subquery is unnested as a view, then the hint does not merge the view.

The following is an example of using NO_MERGE_SUBQ hint.

```
\EXPLAIN PLAN
SELECT
       o_orderdate,
       o_totalprice
  FROM orders
 WHERE o_orderkey IN (
                       SELECT   /*+ NO_MERGE_SUBQ */
                                l_orderkey
                         FROM   lineitem
                       GROUP BY l_orderkey
                     )
;
>>>  start print plan
< Execution Plan >
================================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                      ¦
--------------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                     ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                        ¦
¦    2  ¦      NESTED JOIN (INNER JOIN)                                       ¦
¦    3  ¦        INLINE_VIEW ("$V4")                                          ¦
¦    4  ¦          QUERY BLOCK ("$QB_IDX_6")                                  ¦
¦    5  ¦            GROUP                                                    ¦
¦    6  ¦              INDEX ACCESS ("LINEITEM", "LINEITEM_PK_INDEX")         ¦
¦    7  ¦        INDEX ACCESS ("ORDERS", "ORDERS_PK_INDEX")                   ¦
================================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     2  -  JOINED COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_TOTALPRICE
     3  -  COLUMN : LINEITEM.L_ORDERKEY AS L_ORDERKEY
     4  -  TARGET : LINEITEM.L_ORDERKEY
     5  -  GROUP KEY : LINEITEM.L_ORDERKEY
     6  -  READ INDEX COLUMN : LINEITEM.L_ORDERKEY
     7  -  READ INDEX COLUMN : ORDERS.O_ORDERKEY
           READ TABLE COLUMN : ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE
             MIN RANGE : ORDERS.O_ORDERKEY = {$V4.L_ORDERKEY}
```

```
            MAX RANGE : ORDERS.O_ORDERKEY = {$V4.L_ORDERKEY}
          FETCH ONE ROW
<<<  end print plan
```

In the execution plan above, if a subquery is unnested as a view, then the nested join is performed for the view as it is.

〈transitive closure hints〉

Those hints determine whether to apply the join transitive closure method.
For more information about join transitive closure, refer to **Join Transitive Closure**.

TRANSITIVE_CLOSURE

If TRANSITIVE_CLOSURE hint is specified, join transitive closure is applied during the rewriter process.

The following is an example of using TRANSITIVE_CLOSURE hint.

```
\EXPLAIN PLAN
SELECT /*+ TRANSITIVE_CLOSURE */
       l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount
  FROM part,
       lineitem,
       partsupp
 WHERE ps_suppkey = l_suppkey
   AND ps_partkey = l_partkey
   AND p_partkey = l_partkey
   AND p_name like '%green%';
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                                          |   ROWS |
--------------------------------------------------------------------------------
| 0 |SELECT STATEMENT                                                | 319404 |
| 1 |  QUERY BLOCK ("$QB_IDX_2")                                     | 319404 |
| 2 |    NESTED JOIN (INNER JOIN)                                    | 319404 |
| 3 |      NESTED JOIN (INNER JOIN)                                  |  42656 |
| 4 |        TABLE ACCESS ("PART")                                   |  10664 |
| 5 |        INDEX ACCESS ("PARTSUPP", "PARTSUPP_PARTKEY_FK")        |  42656 |
| 6 |      INDEX ACCESS ("LINEITEM","LINEITEM_PARTKEY_SUPPKEY_FK")| 319404 |
================================================================================
     1  -  TARGET : ( LINEITEM.L_EXTENDEDPRICE * ( 1 - LINEITEM.L_DISCOUNT ) ) - (
```

```
PARTSUPP.PS_SUPPLYCOST * LINEITEM.L_QUANTITY ) AS AMOUNT
     2  -  JOINED COLUMN : LINEITEM.L_EXTENDEDPRICE, LINEITEM.L_DISCOUNT,
PARTSUPP.PS_SUPPLYCOST, LINEITEM.L_QUANTITY
     3  -  JOINED COLUMN : PART.P_PARTKEY, PARTSUPP.PS_SUPPKEY, PARTSUPP.PS_SUPPLYCOST
     4  -  READ COLUMN : PART.P_PARTKEY, PART.P_NAME
             LOGICAL FILTER : PART.P_NAME LIKE '%green%'
     5  -  READ INDEX COLUMN : PARTSUPP.PS_PARTKEY
           READ TABLE COLUMN : PARTSUPP.PS_SUPPKEY, PARTSUPP.PS_SUPPLYCOST
             MIN RANGE : PARTSUPP.PS_PARTKEY = {PART.P_PARTKEY}
             MAX RANGE : PARTSUPP.PS_PARTKEY = {PART.P_PARTKEY}
     6  -  READ INDEX COLUMN : LINEITEM.L_PARTKEY, LINEITEM.L_SUPPKEY
           READ TABLE COLUMN : LINEITEM.L_QUANTITY, LINEITEM.L_EXTENDEDPRICE,
LINEITEM.L_DISCOUNT
             MIN RANGE : LINEITEM.L_PARTKEY = {PART.P_PARTKEY} AND LINEITEM.L_SUPPKEY =
{PARTSUPP.PS_SUPPKEY}
             MAX RANGE : LINEITEM.L_PARTKEY = {PART.P_PARTKEY} AND LINEITEM.L_SUPPKEY =
{PARTSUPP.PS_SUPPKEY}
<<<  end print plan
```

A join condition does not exist in *partsupp* and *part* in the example query above, but the join condition (*ps_partkey = p_partkey*) exists in *partsupp* and *part* in the execution plan.

The user does not specify ps_partkey = p_partkey, but it is created through the relation such as ps_partkey = l_partkey AND p_partkey = l_partkey. The method creating a join condition by using another join condition in this way is called as join transitive closure. Therefore, part and partsupp can be joined first, so the join whose intermediate result are relatively fewer is preferencially performed so that the performance is upgraded.

NO_TRANSITIVE_CLOSURE

If NO_TRANSITIVE_CLOSURE hint is specified, join transitive closure is not applied during the rewriter process.

The following is an example of using NO_TRANSITIVE_CLOSURE hint.

```
\EXPLAIN PLAN
SELECT /*+ NO_TRANSITIVE_CLOSURE */
       l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount
  FROM part,
       lineitem,
       partsupp
 WHERE ps_suppkey = l_suppkey
   AND ps_partkey = l_partkey
```

```
   AND p_partkey = l_partkey
   AND p_name like '%green%';
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                                          |  ROWS |
--------------------------------------------------------------------------------
| 0 |SELECT STATEMENT                                                | 319404 |
| 1 |  QUERY BLOCK ("$QB_IDX_2")                                     | 319404 |
| 2 |    HASH JOIN (INNER JOIN)                                      | 319404 |
| 3 |      TABLE ACCESS ("PARTSUPP")                                 | 800000 |
| 4 |      HASH JOIN INSTANT                                         | 319404 |
| 5 |        NESTED JOIN (INNER JOIN)                                | 319404 |
| 6 |          TABLE ACCESS ("PART")                                 |  10664 |
| 7 |          INDEX ACCESS ("LINEITEM","LINEITEM_PARTKEY_SUPPKEY_FK")| 319404 |
================================================================================
     1  -  TARGET : ( LINEITEM.L_EXTENDEDPRICE * ( 1 - LINEITEM.L_DISCOUNT ) ) - (
PARTSUPP.PS_SUPPLYCOST * LINEITEM.L_QUANTITY ) AS AMOUNT
     2  -  JOINED COLUMN : LINEITEM.L_EXTENDEDPRICE, LINEITEM.L_DISCOUNT,
PARTSUPP.PS_SUPPLYCOST, LINEITEM.L_QUANTITY
     3  -  READ COLUMN : PARTSUPP.PS_PARTKEY, PARTSUPP.PS_SUPPKEY, PARTSUPP.PS_SUPPLYCOST
     4  -  HASH KEY : LINEITEM.L_PARTKEY, LINEITEM.L_SUPPKEY
           RECORD COLUMN : LINEITEM.L_EXTENDEDPRICE, LINEITEM.L_DISCOUNT, LINEITEM.L_QUANTITY
           READ KEY COLUMN : LINEITEM.L_PARTKEY, LINEITEM.L_SUPPKEY, LINEITEM.L_EXTENDEDPRICE,
LINEITEM.L_DISCOUNT, LINEITEM.L_QUANTITY
             HASH FILTER : LINEITEM.L_PARTKEY = PARTSUPP.PS_PARTKEY AND LINEITEM.L_SUPPKEY =
PARTSUPP.PS_SUPPKEY
     5  -  JOINED COLUMN : LINEITEM.L_PARTKEY, LINEITEM.L_SUPPKEY, LINEITEM.L_EXTENDEDPRICE,
LINEITEM.L_DISCOUNT, LINEITEM.L_QUANTITY
     6  -  READ COLUMN : PART.P_PARTKEY, PART.P_NAME
             LOGICAL FILTER : PART.P_NAME LIKE '%green%'
     7  -  READ INDEX COLUMN : LINEITEM.L_PARTKEY, LINEITEM.L_SUPPKEY
           READ TABLE COLUMN : LINEITEM.L_QUANTITY, LINEITEM.L_EXTENDEDPRICE,
LINEITEM.L_DISCOUNT
             MIN RANGE : LINEITEM.L_PARTKEY = {PART.P_PARTKEY}
             MAX RANGE : LINEITEM.L_PARTKEY = {PART.P_PARTKEY}
<<<  end print plan
```

In the example above, the example and the query are as same as those of TRANSITIVE_CLOSURE hint, bu t the execution plan is different from that of TRANSITIVE_CLOSURE hint. It is because the execution plan i s created with only the join condition described by the user.

If statistics information is correct, then the time to perform the query in the execution plan above may tak e longer than the time to perform the example of TRANSITIVE_CLOSURE hint. It is because its intermediat e results of join are more.

However, if the number of lineitem rows are much fewer due to an incorrect statistics information, then i t is more effected to create the execution plan with only the join condition specified by the user. Therefor e, use NO_TRANSITIVE_CLOSURE hint in this case.

〈view hints〉

〈view merge hints〉

Those hints determine whether to apply the view merging method during the rewriter process.

MERGE(view_name)

If MERGE(view_name) hint is specified, the view with view_name is merged with an outer query.

The following is an example of using MERGE(view_name) hint.

```
CREATE OR REPLACE VIEW v_nation
  (
     v_nationkey,
     v_nation_name,
     v_region_name
  )
AS SELECT n_nationkey,
          n_name,
          r_name
     FROM nation,
          region
    WHERE n_regionkey = r_regionkey;
\EXPLAIN PLAN
  SELECT /*+ MERGE(v_nation) */
         v_nation_name,
         count(*)
    FROM customer,
         v_nation
   WHERE c_nationkey = v_nationkey
     AND v_region_name = 'ASIA'
GROUP BY v_nation_name ;
>>>  start print plan
< Execution Plan >
```

```
=============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
|    2  |      GROUP HASH INSTANT                                           |
|    3  |        NESTED JOIN (INNER JOIN)                                   |
|    4  |          NESTED JOIN (INNER JOIN)                                 |
|    5  |            TABLE ACCESS ("REGION")                                |
|    6  |            INDEX ACCESS ("NATION", "NATION_REGIONKEY_FK")         |
|    7  |          INDEX ACCESS ("CUSTOMER", "CUSTOMER_NATIONKEY_FK")       |
=============================================================================
     1  -  TARGET : NATION.N_NAME, COUNT(*)
     2  -  GROUP KEY : NATION.N_NAME
           RECORD COLUMN : COUNT(*)
           READ KEY COLUMN : NATION.N_NAME
           READ RECORD COLUMN : COUNT(*)
     3  -  JOINED COLUMN : NATION.N_NAME
     4  -  JOINED COLUMN : NATION.N_NATIONKEY, NATION.N_NAME
     5  -  READ COLUMN : REGION.R_REGIONKEY, REGION.R_NAME
             PHYSICAL FILTER : REGION.R_NAME = 'ASIA'
     6  -  READ INDEX COLUMN : NATION.N_REGIONKEY
           READ TABLE COLUMN : NATION.N_NATIONKEY, NATION.N_NAME
             MIN RANGE : NATION.N_REGIONKEY = {REGION.R_REGIONKEY}
             MAX RANGE : NATION.N_REGIONKEY = {REGION.R_REGIONKEY}
     7  -  READ INDEX COLUMN : CUSTOMER.C_NATIONKEY
             MIN RANGE : CUSTOMER.C_NATIONKEY = {NATION.N_NATIONKEY}
             MAX RANGE : CUSTOMER.C_NATIONKEY = {NATION.N_NATIONKEY}
<<<  end print plan
```

In the execution plan above, v_nation is merged with an outer query.

NO_MERGE(view_name)

If NO_MERGE(view_name) hint is specified, the view with view_name is not merged with an outer query.

The following is an example of using NO_MERGE(view_name) hint.

```
CREATE OR REPLACE VIEW v_nation
  (
    v_nationkey,
    v_nation_name,
    v_region_name
```

```
  )
AS SELECT n_nationkey,
          n_name,
          r_name
     FROM nation,
          region
    WHERE n_regionkey = r_regionkey;
```

- The view merging method is not applied.

```
\EXPLAIN PLAN
  SELECT /*+ NO_MERGE(v_nation) */
         v_nation_name,
         count(*)
    FROM customer,
         v_nation
   WHERE c_nationkey = v_nationkey
     AND v_region_name = 'ASIA'
GROUP BY v_nation_name ;
>>>  start print plan
< Execution Plan >
================================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                      ¦
--------------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                      ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                         ¦
¦    2  ¦      GROUP HASH INSTANT                                              ¦
¦    3  ¦        NESTED JOIN (INNER JOIN)                                      ¦
¦    4  ¦          VIEW ("V_NATION")                                           ¦
¦    5  ¦            QUERY BLOCK ("$QB_IDX_7")                                 ¦
¦    6  ¦              NESTED JOIN (INNER JOIN)                                ¦
¦    7  ¦                TABLE ACCESS ("REGION")                               ¦
¦    8  ¦                INDEX ACCESS ("NATION", "NATION_REGIONKEY_FK")        ¦
¦    9  ¦          INDEX ACCESS ("CUSTOMER", "CUSTOMER_NATIONKEY_FK")          ¦
================================================================================
     1  -  TARGET : V_NATION.V_NATION_NAME, COUNT(*)
     2  -  GROUP KEY : V_NATION.V_NATION_NAME
           RECORD COLUMN : COUNT(*)
           READ KEY COLUMN : V_NATION.V_NATION_NAME
           READ RECORD COLUMN : COUNT(*)
     3  -  JOINED COLUMN : V_NATION.V_NATION_NAME
```

```
     4  -  COLUMN : V_NATIONKEY AS V_NATIONKEY, V_NATION_NAME AS V_NATION_NAME, V_REGION_NAME
AS V_REGION_NAME
     5  -  TARGET : NATION.N_NATIONKEY AS V_NATIONKEY, NATION.N_NAME AS V_NATION_NAME,
REGION.R_NAME AS V_REGION_NAME
     6  -  JOINED COLUMN : NATION.N_NATIONKEY, NATION.N_NAME, REGION.R_NAME
     7  -  READ COLUMN : REGION.R_REGIONKEY, REGION.R_NAME
             PHYSICAL FILTER : REGION.R_NAME = 'ASIA'
     8  -  READ INDEX COLUMN : NATION.N_REGIONKEY
           READ TABLE COLUMN : NATION.N_NATIONKEY, NATION.N_NAME
             MIN RANGE : NATION.N_REGIONKEY = {REGION.R_REGIONKEY}
             MAX RANGE : NATION.N_REGIONKEY = {REGION.R_REGIONKEY}
     9  -  READ INDEX COLUMN : CUSTOMER.C_NATIONKEY
             MIN RANGE : CUSTOMER.C_NATIONKEY = {V_NATION.V_NATIONKEY}
             MAX RANGE : CUSTOMER.C_NATIONKEY = {V_NATION.V_NATIONKEY}
<<<  end print plan
```

In the execution plan above, v_nation is not merged with an outer query, but it exists as it is in view form.

〈push view predicate hints〉

Those hints either push the view-related join predicate into the view or prevent the pushing.

PUSH_PRED

If PUSH_PRED hint is specified, it pushes all join predicates related to views listed in a from clause into the view.

The following is an example of using PUSH_PRED hint.

```
CREATE OR REPLACE VIEW v_nation
  (
     v_nationkey,
     v_nation_name,
     v_region_name
  )
AS SELECT n_nationkey,
          n_name,
          r_name
     FROM nation,
          region
    WHERE n_regionkey = r_regionkey;
\EXPLAIN PLAN
  SELECT /*+ PUSH_PRED */
         v_nation_name,
```

```
        count(*)
    FROM customer,
        v_nation
   WHERE c_nationkey = v_nationkey
     AND v_region_name = 'ASIA'
GROUP BY v_nation_name;
>>>  start print plan
< Execution Plan >
============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
|    2  |      GROUP HASH INSTANT                                           |
|    3  |        NESTED JOIN (INNER JOIN)                                   |
|    4  |          INDEX ACCESS ("CUSTOMER", "CUSTOMER_NATIONKEY_FK")       |
|    5  |          VIEW ("V_NATION")                                        |
|    6  |            QUERY BLOCK ("$QB_IDX_7")                              |
|    7  |              NESTED JOIN (INNER JOIN)                             |
|    8  |                INDEX ACCESS ("NATION", "NATION_PK_INDEX")          |
|    9  |                INDEX ACCESS ("REGION", "REGION_PK_INDEX")          |
============================================================================
     1  -  TARGET : V_NATION.V_NATION_NAME, COUNT(*)
     2  -  GROUP KEY : V_NATION.V_NATION_NAME
           RECORD COLUMN : COUNT(*)
           READ KEY COLUMN : V_NATION.V_NATION_NAME
           READ RECORD COLUMN : COUNT(*)
     3  -  JOINED COLUMN : V_NATION.V_NATION_NAME
     4  -  READ INDEX COLUMN : CUSTOMER.C_NATIONKEY
     5  -  COLUMN : V_NATIONKEY AS V_NATIONKEY, V_NATION_NAME AS V_NATION_NAME, V_REGION_NAME
AS V_REGION_NAME
     6  -  TARGET : NATION.N_NATIONKEY AS V_NATIONKEY, NATION.N_NAME AS V_NATION_NAME,
REGION.R_NAME AS V_REGION_NAME
     7  -  JOINED COLUMN : NATION.N_NATIONKEY, NATION.N_NAME, REGION.R_NAME
     8  -  READ INDEX COLUMN : NATION.N_NATIONKEY
           READ TABLE COLUMN : NATION.N_NAME, NATION.N_REGIONKEY
             MIN RANGE : NATION.N_NATIONKEY = {CUSTOMER.C_NATIONKEY}
             MAX RANGE : NATION.N_NATIONKEY = {CUSTOMER.C_NATIONKEY}
           FETCH ONE ROW
     9  -  READ INDEX COLUMN : REGION.R_REGIONKEY
           READ TABLE COLUMN : REGION.R_NAME
```

```
            MIN RANGE : REGION.R_REGIONKEY = {NATION.N_REGIONKEY}
            MAX RANGE : REGION.R_REGIONKEY = {NATION.N_REGIONKEY}
            PHYSICAL TABLE FILTER : REGION.R_NAME = 'ASIA'
          FETCH ONE ROW
<<<  end print plan
```

In the execution plan above, c_nationkey = v_nationkey is used by being pushed into the view.

NO_PUSH_PRED

If NO_PUSH_PRED hint is specified, it prevents pushing all join predicates related to views listed in a from clause into the view.

The following is an example of using NO_PUSH_PRED hint.

```
CREATE OR REPLACE VIEW v_nation
  (
     v_nationkey,
     v_nation_name,
     v_region_name
  )
AS SELECT n_nationkey,
          n_name,
          r_name
     FROM nation,
          region
    WHERE n_regionkey = r_regionkey;
\EXPLAIN PLAN
  SELECT /*+ NO_PUSH_PRED */
         v_nation_name,
         count(*)
    FROM customer,
         v_nation
   WHERE c_nationkey = v_nationkey
     AND v_region_name = 'ASIA'
GROUP BY v_nation_name;
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                  ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                  ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                     ¦
```

```
|    2  |      GROUP HASH INSTANT                                        |
|    3  |        NESTED JOIN (INNER JOIN)                                |
|    4  |          VIEW ("V_NATION")                                     |
|    5  |            QUERY BLOCK ("$QB_IDX_7")                           |
|    6  |              NESTED JOIN (INNER JOIN)                          |
|    7  |                TABLE ACCESS ("REGION")                         |
|    8  |                INDEX ACCESS ("NATION", "NATION_REGIONKEY_FK")   |
|    9  |          INDEX ACCESS ("CUSTOMER", "CUSTOMER_NATIONKEY_FK")    |
=========================================================================
     1  -  TARGET : V_NATION.V_NATION_NAME, COUNT(*)
     2  -  GROUP KEY : V_NATION.V_NATION_NAME
           RECORD COLUMN : COUNT(*)
           READ KEY COLUMN : V_NATION.V_NATION_NAME
           READ RECORD COLUMN : COUNT(*)
     3  -  JOINED COLUMN : V_NATION.V_NATION_NAME
     4  -  COLUMN : V_NATIONKEY AS V_NATIONKEY, V_NATION_NAME AS V_NATION_NAME, V_REGION_NAME
AS V_REGION_NAME
     5  -  TARGET : NATION.N_NATIONKEY AS V_NATIONKEY, NATION.N_NAME AS V_NATION_NAME,
REGION.R_NAME AS V_REGION_NAME
     6  -  JOINED COLUMN : NATION.N_NATIONKEY, NATION.N_NAME, REGION.R_NAME
     7  -  READ COLUMN : REGION.R_REGIONKEY, REGION.R_NAME
             PHYSICAL FILTER : REGION.R_NAME = 'ASIA'
     8  -  READ INDEX COLUMN : NATION.N_REGIONKEY
           READ TABLE COLUMN : NATION.N_NATIONKEY, NATION.N_NAME
             MIN RANGE : NATION.N_REGIONKEY = {REGION.R_REGIONKEY}
             MAX RANGE : NATION.N_REGIONKEY = {REGION.R_REGIONKEY}
     9  -  READ INDEX COLUMN : CUSTOMER.C_NATIONKEY
             MIN RANGE : CUSTOMER.C_NATIONKEY = {V_NATION.V_NATIONKEY}
             MAX RANGE : CUSTOMER.C_NATIONKEY = {V_NATION.V_NATIONKEY}
<<<  end print plan
```

In the execution plan above, c_nationkey = v_nationkey is used outside of the view.

PUSH_PRED(view_name[[,] view_name])

If PUSH_PRED(view_name[[,] view_name]) hint is specified, it pushes all join predicates related to vie ws listed ina from clause into the view.

The following is an example of using PUSH_PRED(view_name[[,] view_name]) hint.

```
CREATE OR REPLACE VIEW v_nation
  (
    v_nationkey,
```

```
     v_nation_name,
     v_region_name
  )
AS SELECT n_nationkey,
          n_name,
          r_name
     FROM nation,
          region
    WHERE n_regionkey = r_regionkey;
\EXPLAIN PLAN
  SELECT /*+ PUSH_PRED(v_nation) */
         v_nation_name,
         count(*)
    FROM customer,
         v_nation
   WHERE c_nationkey = v_nationkey
     AND v_region_name = 'ASIA'
GROUP BY v_nation_name;
>>>  start print plan
< Execution Plan >
============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                 ¦
----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                    ¦
¦    2  ¦      GROUP HASH INSTANT                                         ¦
¦    3  ¦        NESTED JOIN (INNER JOIN)                                 ¦
¦    4  ¦          INDEX ACCESS ("CUSTOMER", "CUSTOMER_NATIONKEY_FK")     ¦
¦    5  ¦          VIEW ("V_NATION")                                      ¦
¦    6  ¦            QUERY BLOCK ("$QB_IDX_7")                            ¦
¦    7  ¦              NESTED JOIN (INNER JOIN)                           ¦
¦    8  ¦                INDEX ACCESS ("NATION", "NATION_PK_INDEX")       ¦
¦    9  ¦                INDEX ACCESS ("REGION", "REGION_PK_INDEX")       ¦
============================================================================
     1  -  TARGET : V_NATION.V_NATION_NAME, COUNT(*)
     2  -  GROUP KEY : V_NATION.V_NATION_NAME
           RECORD COLUMN : COUNT(*)
           READ KEY COLUMN : V_NATION.V_NATION_NAME
           READ RECORD COLUMN : COUNT(*)
     3  -  JOINED COLUMN : V_NATION.V_NATION_NAME
     4  -  READ INDEX COLUMN : CUSTOMER.C_NATIONKEY
```

```
     5  -  COLUMN : V_NATIONKEY AS V_NATIONKEY, V_NATION_NAME AS V_NATION_NAME, V_REGION_NAME
AS V_REGION_NAME
     6  -  TARGET : NATION.N_NATIONKEY AS V_NATIONKEY, NATION.N_NAME AS V_NATION_NAME,
REGION.R_NAME AS V_REGION_NAME
     7  -  JOINED COLUMN : NATION.N_NATIONKEY, NATION.N_NAME, REGION.R_NAME
     8  -  READ INDEX COLUMN : NATION.N_NATIONKEY
           READ TABLE COLUMN : NATION.N_NAME, NATION.N_REGIONKEY
             MIN RANGE : NATION.N_NATIONKEY = {CUSTOMER.C_NATIONKEY}
             MAX RANGE : NATION.N_NATIONKEY = {CUSTOMER.C_NATIONKEY}
           FETCH ONE ROW
     9  -  READ INDEX COLUMN : REGION.R_REGIONKEY
           READ TABLE COLUMN : REGION.R_NAME
             MIN RANGE : REGION.R_REGIONKEY = {NATION.N_REGIONKEY}
             MAX RANGE : REGION.R_REGIONKEY = {NATION.N_REGIONKEY}
             PHYSICAL TABLE FILTER : REGION.R_NAME = 'ASIA'
           FETCH ONE ROW
<<<  end print plan
```

NO_PUSH_PRED(view_name[[,] view_name])

If NO_PUSH_PRED(view_name[[,] view_name]) hint is specified, it prevents pushing all join predicates related to views listed in a from clause into the view.

The following is an example of using NO_PUSH_PRED(view_name[[,] view_name]) hint.

```
CREATE OR REPLACE VIEW v_nation
  (
     v_nationkey,
     v_nation_name,
     v_region_name
  )
AS SELECT n_nationkey,
          n_name,
          r_name
     FROM nation,
          region
    WHERE n_regionkey = r_regionkey;
\EXPLAIN PLAN
  SELECT /*+ NO_PUSH_PRED(v_nation) */
         v_nation_name,
         count(*)
    FROM customer,
```

```
         v_nation
   WHERE c_nationkey = v_nationkey
     AND v_region_name = 'ASIA'
GROUP BY v_nation_name;
>>>  start print plan
< Execution Plan >
================================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                     ¦
--------------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                    ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                       ¦
¦    2  ¦      GROUP HASH INSTANT                                            ¦
¦    3  ¦        NESTED JOIN (INNER JOIN)                                    ¦
¦    4  ¦          VIEW ("V_NATION")                                         ¦
¦    5  ¦            QUERY BLOCK ("$QB_IDX_7")                               ¦
¦    6  ¦              NESTED JOIN (INNER JOIN)                              ¦
¦    7  ¦                TABLE ACCESS ("REGION")                             ¦
¦    8  ¦                INDEX ACCESS ("NATION", "NATION_REGIONKEY_FK")      ¦
¦    9  ¦          INDEX ACCESS ("CUSTOMER", "CUSTOMER_NATIONKEY_FK")        ¦
================================================================================
     1  -  TARGET : V_NATION.V_NATION_NAME, COUNT(*)
     2  -  GROUP KEY : V_NATION.V_NATION_NAME
           RECORD COLUMN : COUNT(*)
           READ KEY COLUMN : V_NATION.V_NATION_NAME
           READ RECORD COLUMN : COUNT(*)
     3  -  JOINED COLUMN : V_NATION.V_NATION_NAME
     4  -  COLUMN : V_NATIONKEY AS V_NATIONKEY, V_NATION_NAME AS V_NATION_NAME, V_REGION_NAME
AS V_REGION_NAME
     5  -  TARGET : NATION.N_NATIONKEY AS V_NATIONKEY, NATION.N_NAME AS V_NATION_NAME,
REGION.R_NAME AS V_REGION_NAME
     6  -  JOINED COLUMN : NATION.N_NATIONKEY, NATION.N_NAME, REGION.R_NAME
     7  -  READ COLUMN : REGION.R_REGIONKEY, REGION.R_NAME
             PHYSICAL FILTER : REGION.R_NAME = 'ASIA'
     8  -  READ INDEX COLUMN : NATION.N_REGIONKEY
           READ TABLE COLUMN : NATION.N_NATIONKEY, NATION.N_NAME
             MIN RANGE : NATION.N_REGIONKEY = {REGION.R_REGIONKEY}
             MAX RANGE : NATION.N_REGIONKEY = {REGION.R_REGIONKEY}
     9  -  READ INDEX COLUMN : CUSTOMER.C_NATIONKEY
            MIN RANGE : CUSTOMER.C_NATIONKEY = {V_NATION.V_NATIONKEY}
            MAX RANGE : CUSTOMER.C_NATIONKEY = {V_NATION.V_NATIONKEY}
<<<  end print plan
```

Operation Hint

Those hints are applied per each relation.

〈access path hints〉

It defines how to access a single table.

FULL(table_name)

If FULL(table_name) hint is specified, then an optimizer performs the table full scan for the specified tabl e.
Only one table_name can be specified, and it should exist in 〈from clause〉.

The following is an example of using FULL(table_name) hint.

```
SELECT count(*) FROM nation;
COUNT(*)
--------
      25
1 row selected.
\EXPLAIN PLAN
SELECT /*+ FULL( nation ) */
       n_name
  FROM nation
 WHERE n_nationkey < 10;
>>>  start print plan
< Execution Plan >
============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                  ¦ ROWS ¦
----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                  ¦   10 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                     ¦   10 ¦
¦    2  ¦      TABLE ACCESS ("NATION")                     ¦   10 ¦
============================================================================
     1  -  TARGET : NATION.N_NAME
     2  -  READ COLUMN : NATION.N_NATIONKEY, NATION.N_NAME
             PHYSICAL FILTER : NATION.N_NATIONKEY < 10
<<<  end print plan
```

In the example above, n_nationkey is a primary key column so the index access is available. However the

performance of the table access is better than that of the index access when selecting ten results from a s mall table whose nation table has 25 rows only.

INDEX(table_name [,] [index_name [[,] index_name]])

If INDEX(table_name [,] [index_name [[,] index_name]]) hint is specified, then an optimizer perfor ms the index scan for the specified table. In this case, the index with the best cost is selected among liste d indexes. If index_name is not specified, then the index with the best cost is selected among all indexes i n the table.
One or more table_name can be specified, and it should exist in <from clause>.

An index name can be specified one or more, or it can be omitted. And it should be index_name which e xists in the table corresponding to table_name.

The following is an example of using INDEX(table_name) hint.

```
\EXPLAIN PLAN
  SELECT /*+ INDEX( nation ) */
         n_nationkey,
         n_name
    FROM nation
   WHERE n_nationkey < 10
ORDER BY n_nationkey;

>>>  start print plan
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                                      | ROWS |
-----------------------------------------------------------------------------
| 0 |  SELECT STATEMENT                                          |  10  |
| 1 |    QUERY BLOCK ("$QB_IDX_2")                               |  10  |
| 2 |      INDEX ACCESS ("NATION", "NATION_PK_INDEX")            |  10  |
=============================================================================
     1  -  TARGET : NATION.N_NATIONKEY, NATION.N_NAME
     2  -  READ INDEX COLUMN : NATION.N_NATIONKEY
           READ TABLE COLUMN : NATION.N_NAME
             MAX RANGE : NATION.N_NATIONKEY < 10
<<<  end print plan
```

In the example above, a nation table is a small table which has 25 rows, and only 10 rows are selected a mong those. Therefore, the performance of the table access is better than that of the index access. Howe ver, the result of an index access is as if it is ordered by even though the order by is not separately perfor

med, so the order by can be omitted. Therefore, an index access may be better in this case.

NO_INDEX(table_name [,] [index_name [[,] index_name]])

If NO_INDEX(table_name [,] [index_name [[,] index_name]]) hint is specified, then an optimizer pr events using the specified index. If index_name is not specified, then none of the index is used. In other words, it does not perform the index scan.
Only one table_name can be specified, and it should exist in <from clause>.

An index name can be specified one or more, or it can be omitted. And it should be index_name which e xists in the table corresponding to table_name.

The following is an example of using NO_INDEX(table_name) hint.

```
\EXPLAIN PLAN
  SELECT /*+ NO_INDEX( nation ) */
         n_nationkey,
         n_name
    FROM nation
   WHERE n_nationkey < 10
ORDER BY n_nationkey;
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                  ¦ ROWS ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                  ¦  10  ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                     ¦  10  ¦
¦    2  ¦      SORT INSTANT                                ¦  10  ¦
¦    3  ¦        TABLE ACCESS ("NATION")                   ¦  10  ¦
=============================================================================
     1  -  TARGET : NATION.N_NATIONKEY, NATION.N_NAME
     2  -  SORT KEY : "NATION.N_NATIONKEY ASC NULLS LAST"
           RECORD COLUMN : NATION.N_NAME
           READ KEY COLUMN : NATION.N_NATIONKEY
           READ RECORD COLUMN : NATION.N_NAME
     3  -  READ COLUMN : NATION.N_NATIONKEY, NATION.N_NAME
             PHYSICAL FILTER : NATION.N_NATIONKEY < 10
<<<  end print plan
```

In the example above, the table access is performed because using any index is prevented.

INDEX_FORWARD(table_name [,] [index_name [[,] index_name]])

INDEX_FORWARD(table_name [,] [index_name [[,] index_name]]) hint is as same as INDEX(table_name [,] [index_name[[,] index_name]]) hint.
It performs the forward scan for the index. Therefore, if the index was created in an ascending order, then it outputs the result in an ascending order. If the index was created in a descending order, then it outputs the result in a descending order.

The following is an example of using INDEX_FORWARD(table_name) hint.

```
\EXPLAIN PLAN
  SELECT /*+ INDEX_FORWARD( nation ) */
         n_nationkey,
         n_name
    FROM nation
   WHERE n_nationkey < 10
ORDER BY n_nationkey ASC;
>>>  start print plan
< Execution Plan >
=====================================================================
|  IDX  |  NODE DESCRIPTION                                 | ROWS |
---------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                 |  10  |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                      |  10  |
|    2  |      INDEX ACCESS ("NATION", "NATION_PK_INDEX")   |  10  |
=====================================================================
     1  -  TARGET : NATION.N_NATIONKEY, NATION.N_NAME
     2  -  READ INDEX COLUMN : NATION.N_NATIONKEY
           READ TABLE COLUMN : NATION.N_NAME
             MAX RANGE : NATION.N_NATIONKEY < 10
<<<  end print plan
```

In the example above, if NATION_PK_INDEX was created in an ascending order, then separate sorting for ORDER BY is not required when performing the forward scan for the index.

INDEX_BACKWARD(table_name [,] [index_name [[,] index_name]])

It performs the backward scan for the index. Therefore, if the index was created in an ascending order, then it outputs the result in a descending order. If the index was created in a descending order, then it outputs the result in an ascending order.
The syntax rule of INDEX_BACKWARD(table_name [,] [index_name [[,] index_name]]) is as same as that of INDEX(table_name [,] [index_name[[,] index_name]]) hint.

The following is an example of using INDEX_BACKWARD(table_name) hint.

```
\EXPLAIN PLAN
  SELECT /*+ INDEX_BACKWARD( nation ) */
         n_nationkey,
         n_name
    FROM nation
   WHERE n_nationkey < 10
ORDER BY n_nationkey DESC;
>>>  start print plan
< Execution Plan >
=================================================================
|  IDX  |  NODE DESCRIPTION                              | ROWS |
-----------------------------------------------------------------
|    0  |  SELECT STATEMENT                              |  10  |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                   |  10  |
|    2  |      INDEX ACCESS ("NATION", "NATION_PK_INDEX")|  10  |
=================================================================
     1  -  TARGET : NATION.N_NATIONKEY, NATION.N_NAME
     2  -  READ INDEX COLUMN : NATION.N_NATIONKEY
           READ TABLE COLUMN : NATION.N_NAME
             MAX RANGE : NATION.N_NATIONKEY < 10
<<<  end print plan
```

In the example above, if NATION_PK_INDEX was created in an ascending order, then separate sorting for ORDER BY is not required when performing the backward scan for the index.

INDEX_ASC(table_name [,] [index_name [[,] index_name]])

INDEX_ASC(table_name [,] [index_name [[,] index_name]]) hint is as same as INDEX(table_name [,] [index_name[[,] index_name]]) hint.
It performs the forward scan for the index. Therefore, if the index was created in an ascending order, the n it outputs the result in an ascending order. If the index was created in a descending order, then it outp uts the result in a descending order.

The following is an example of using INDEX_ASC(table_name) hint.

```
\EXPLAIN PLAN
  SELECT /*+ INDEX_ASC( nation ) */
         n_nationkey,
         n_name
    FROM nation
```

```
   WHERE n_nationkey < 10
ORDER BY n_nationkey ASC;
>>>  start print plan
< Execution Plan >
=================================================================
|  IDX  |  NODE DESCRIPTION                              | ROWS |
-----------------------------------------------------------------
|    0  |  SELECT STATEMENT                              |  10  |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                   |  10  |
|    2  |      INDEX ACCESS ("NATION", "NATION_PK_INDEX")|  10  |
=================================================================
     1  -  TARGET : NATION.N_NATIONKEY, NATION.N_NAME
     2  -  READ INDEX COLUMN : NATION.N_NATIONKEY
           READ TABLE COLUMN : NATION.N_NAME
             MAX RANGE : NATION.N_NATIONKEY < 10
<<<  end print plan
```

In the example above, if NATION_PK_INDEX was created in an ascending order, then separate sorting for ORDER BY is not required when performing the forward scan for the index.

INDEX_DESC(table_name [,] [index_name [[,] index_name]])

It performs the backward scan for the index. Therefore, if the index was created in an ascending order, th en it outputs the result in a descending order. If the index was created in a descending order, then it outp uts the result in an ascending order.
The syntax rule of INDEX_DESC(table_name [,] [index_name [[,] index_name]]) is as same as that o f INDEX(table_name [,] [index_name[[,] index_name]]) hint.

The following is an example of using INDEX_DESC(table_name) hint.

```
\EXPLAIN PLAN
  SELECT /*+ INDEX_DESC( nation ) */
         n_nationkey,
         n_name
    FROM nation
   WHERE n_nationkey < 10
ORDER BY n_nationkey DESC;
>>>  start print plan
< Execution Plan >
=================================================================
|  IDX  |  NODE DESCRIPTION                              | ROWS |
-----------------------------------------------------------------
```

```
¦    0  ¦  SELECT STATEMENT                                  ¦  10  ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                     ¦  10  ¦
¦    2  ¦      INDEX ACCESS ("NATION", "NATION_PK_INDEX")  ¦  10  ¦
=================================================================
     1  -  TARGET : NATION.N_NATIONKEY, NATION.N_NAME
     2  -  READ INDEX COLUMN : NATION.N_NATIONKEY
           READ TABLE COLUMN : NATION.N_NAME
             MAX RANGE : NATION.N_NATIONKEY < 10
<<<  end print plan
```

In the example above, if NATION_PK_INDEX was created in an ascending order, then separate sorting for ORDER BY is not required when performing the backward scan for the index.

INDEX_COMBINE(table_name [,] [index_name [[,] index_name]])

If INDEX_COMBINE(table_name [,] [index_name [[,] index_name]]) hint is specified, then an optimizer separates OR statements in the specified table, performs the index scan and merges the results. In this case, the index with the best cost is selected among listed indexes when determining each index scan. If index_name is not specified, then the index with the best cost is selected among all indexes in the table. Therefore, each different index can be used according to OR statements.

When specifying INDEX_COMBINE hint, a table name should exists in <from clause> and the index name should be the one existing in the table.

The or statement should exist in the condition to scan the table for performing INDEX_COMBINE hint. If the or statement does not exist, then the optimizer ignores the hint.

The following is an example of using INDEX_COMBINE(table_name) hint.

```
\EXPLAIN PLAN
SELECT /*+ INDEX_COMBINE( orders ) */
       o_orderstatus
  FROM orders
 WHERE o_orderkey = 1 OR o_custkey = 1;
>>>  start print plan
< Execution Plan >
=========================================================================
¦  IDX  ¦  NODE DESCRIPTION                                    ¦ ROWS ¦
-------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                    ¦   7  ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                       ¦   7  ¦
¦    2  ¦      CONCAT (Compare RID)                          ¦   7  ¦
¦    3  ¦        INDEX ACCESS ("ORDERS", "ORDERS_PK_INDEX")  ¦   1  ¦
¦    4  ¦        INDEX ACCESS ("ORDERS", "ORDERS_CUSTKEY_FK")¦   6  ¦
```

```
================================================================================
     1  -  TARGET : ORDERS.O_ORDERSTATUS
     2  -  CONCAT COLUMN : ORDERS.$PHYSICAL_ROWID, ORDERS.O_ORDERSTATUS
     3  -  READ INDEX COLUMN : ORDERS.O_ORDERKEY
           READ TABLE COLUMN : ORDERS.O_ORDERSTATUS
             MIN RANGE : ORDERS.O_ORDERKEY = 1
             MAX RANGE : ORDERS.O_ORDERKEY = 1
           FETCH ONE ROW
     4  -  READ INDEX COLUMN : ORDERS.O_CUSTKEY
           READ TABLE COLUMN : ORDERS.O_ORDERSTATUS
             MIN RANGE : ORDERS.O_CUSTKEY = 1
             MAX RANGE : ORDERS.O_CUSTKEY = 1
<<<  end print plan
```

In the example above, the index scan is not available with the entire o_orderkey = 1 OR o_custkey = 1 co ndition, so the performance may become slow. Therefore, separate OR statement and perform the index scan for each condition (o_orderkey = 1 and o_custkey = 1), then merge the results to upgrade the perfo rmance.

IN_KEY_RANGE(table_name [,] [index_name [[,] index_name]])

If IN_KEY_RANGE(table_name [,] [index_name [[,] index_name]]) hint is specified, then an optimiz er performs the IN key range scan for the specified table. In this case, the index with the best cost is select ed among specified indexes. If index_name is not specified, then the index with the best cost is selected a mong all indexes in the table.
However, a filter which is capable of the IN key range is required for applying IN_KEY_RANGE(table_nam e [,] [index_name [[,] index_name]]) hint.

Filter conditions for IN key range are as follows.

e.g. (col1, col2) IN ((val1, val2), (val3, val4))

- *IN* or *=ANY List Function Filter* should exist in WHERE clause.
- col1 and col2 should be base columns. In other words, they should not be operations nor are functions.
- (val1, val3) corresponding to col1 can be converted to a single data type,
  (val2, val4) corresponding to col2 can be converted to a single data type.

The following is an example of using IN_KEY_RANGE hint.

```
\EXPLAIN PLAN
  SELECT /*+ IN_KEY_RANGE( orders ) */
         o_orderstatus
    FROM orders
```

```
   WHERE o_orderkey > 500 AND o_custkey IN ( 1, 10, 100, 1000, 10000 );
< Execution Plan >
============================================================================
|  IDX  |  NODE DESCRIPTION                                        | ROWS |
----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                        |   91 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                             |   91 |
|    2  |      INDEX ACCESS ("ORDERS", "ORDERS_CUSTKEY_FK")        |   91 |
============================================================================
     1  -  TARGET : ORDERS.O_ORDERSTATUS
     2  -  READ INDEX COLUMN : ORDERS.O_CUSTKEY
           READ TABLE COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_ORDERSTATUS
           IN KEY RANGE
             MIN RANGE : ORDERS.O_CUSTKEY = ?
             MAX RANGE : ORDERS.O_CUSTKEY = ?
             PHYSICAL TABLE FILTER : ORDERS.O_ORDERKEY > 500
<<<  end print plan
```

In the execution plan above, the IN key range scan is performed.

ROWID(table_name)

If ROWID(table_name) hint is specified, then an optimizer performs the rowid scan for the specified tabl e. The table name should exist in <from clause> when specifying ROWID hint.

The equal condition using ROWID should exist in the table to apply ROWID hint. If this condition does not exist, then an optimizer ignores this hint.

The following is an example of using ROWID hint.

```
\EXPLAIN PLAN
  SELECT /*+ ROWID( orders ) */
         o_orderstatus
    FROM orders
   WHERE rowid = 'AAAAAAAAYfAAACAAACGjAAA';
>>>  start print plan
< Execution Plan >
============================================================================
|  IDX  |  NODE DESCRIPTION                                        | ROWS |
----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                        |    1 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                             |    1 |
```

```
|    2  |      ROWID ACCESS ("ORDERS")                              |    1 |
=========================================================================

     1  -  TARGET : ORDERS.O_ORDERSTATUS
     2  -  READ COLUMN : ORDERS.O_ORDERSTATUS
             ROWID FILTER : ORDERS.ROWID = 'AAAAAAAAYfAAACAAACGjAAA'
<<<  end print plan
```

〈join hints〉

〈join order hints〉

Those hints define how to perform the join ordering.

ORDERED

If ORDERED hint is specified, then an optimizer orders to perform joining in an order described in 〈from clause〉 when performing the join ordering.

If a user knows how many number of rows satisfying the join conditions and the best join order, then the user can decrease the join ordering cost by using ORDERED hint.

The following is an example of using ORDERED hint.

```
\EXPLAIN PLAN
  SELECT  /*+ ORDERED */
          l_orderkey,
          ROUND( sum(l_extendedprice*(1-l_discount)), 2) as revenue,
          o_orderdate,
          o_shippriority
   FROM   customer,
          orders,
          lineitem
  WHERE   c_mktsegment = 'BUILDING'
    AND   c_custkey = o_custkey
    AND   l_orderkey = o_orderkey
    AND   o_orderdate < date '1995-03-15'
    AND   l_shipdate > date '1995-03-15'
GROUP BY  l_orderkey,
          o_orderdate,
          o_shippriority;
>>>  start print plan
< Execution Plan >
=========================================================================
```

```
| IDX | NODE DESCRIPTION                                          |  ROWS |
-------------------------------------------------------------------------
| 0 | SELECT STATEMENT                                          | 11620 |
| 1 |   QUERY BLOCK ("$QB_IDX_2")                               | 11620 |
| 2 |     GROUP HASH INSTANT                                    | 11620 |
| 3 |       NESTED JOIN (INNER JOIN)                            | 30519 |
| 4 |         NESTED JOIN (INNER JOIN)                          |147126 |
| 5 |           TABLE ACCESS ("CUSTOMER")                       | 30142 |
| 6 |           INDEX ACCESS ("ORDERS", "ORDERS_CUSTKEY_FK")    |147126 |
| 7 |         INDEX ACCESS ("LINEITEM", "LINEITEM_ORDERKEY_FK") | 30519 |
=========================================================================
     1  -  TARGET : LINEITEM.L_ORDERKEY, ROUND(SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 -
LINEITEM.L_DISCOUNT ) ),2) AS REVENUE, ORDERS.O_ORDERDATE, ORDERS.O_SHIPPRIORITY
     2  -  GROUP KEY : LINEITEM.L_ORDERKEY, ORDERS.O_ORDERDATE, ORDERS.O_SHIPPRIORITY
           RECORD COLUMN : SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 - LINEITEM.L_DISCOUNT ) )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY, ORDERS.O_ORDERDATE, ORDERS.O_SHIPPRIORITY
           READ RECORD COLUMN : SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 - LINEITEM.L_DISCOUNT ) )
     3  -  JOINED COLUMN : LINEITEM.L_ORDERKEY, ORDERS.O_ORDERDATE, ORDERS.O_SHIPPRIORITY,
LINEITEM.L_EXTENDEDPRICE, LINEITEM.L_DISCOUNT
     4  -  JOINED COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_ORDERDATE, ORDERS.O_SHIPPRIORITY
     5  -  READ COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_MKTSEGMENT
             PHYSICAL FILTER : CUSTOMER.C_MKTSEGMENT = 'BUILDING'
     6  -  READ INDEX COLUMN : ORDERS.O_CUSTKEY
           READ TABLE COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_ORDERDATE, ORDERS.O_SHIPPRIORITY
             MIN RANGE : ORDERS.O_CUSTKEY = {CUSTOMER.C_CUSTKEY}
             MAX RANGE : ORDERS.O_CUSTKEY = {CUSTOMER.C_CUSTKEY}
             PHYSICAL TABLE FILTER : ORDERS.O_ORDERDATE < DATE'1995-03-15'
     7  -  READ INDEX COLUMN : LINEITEM.L_ORDERKEY
           READ TABLE COLUMN : LINEITEM.L_EXTENDEDPRICE, LINEITEM.L_DISCOUNT,
LINEITEM.L_SHIPDATE
             MIN RANGE : LINEITEM.L_ORDERKEY = {ORDERS.O_ORDERKEY}
             MAX RANGE : LINEITEM.L_ORDERKEY = {ORDERS.O_ORDERKEY}
             PHYSICAL TABLE FILTER : LINEITEM.L_SHIPDATE > DATE'1995-03-15'
<<<  end print plan
```

ORDERING(table_name [, table_name [, table_name [LEFT | RIGHT]]])

If ORDERING(table_name [, table_name [, table_name [LEFT | RIGHT]]]) hint is specified, then an opt imizer orders to perform joining in an order described in the hint when performing the join ordering.

A locating option can not be specified for the first and the second tables, but it can be specified from the third table. Location options are LEFT and RIGHT. LEFT places the table on the left node (outer node) of t

he join, and RIGHT places the table on the right node (inner node) of the join. If a locating option is not u sed, then the location is determined by the cost estimation.

The following is an example of using ORDERING hint.

```
\EXPLAIN PLAN
  SELECT  /*+ ORDERING( customer, orders, lineitem RIGHT ) */
          l_orderkey,
          ROUND( sum(l_extendedprice*(1-l_discount)), 2) as revenue,
          o_orderdate,
          o_shippriority
   FROM   lineitem,
          orders,
          customer
  WHERE   c_mktsegment = 'BUILDING'
    AND   c_custkey = o_custkey
    AND   l_orderkey = o_orderkey
    AND   o_orderdate < date '1995-03-15'
    AND   l_shipdate > date '1995-03-15'
GROUP BY  l_orderkey,
          o_orderdate,
          o_shippriority;
>>>  start print plan
< Execution Plan >
================================================================================
¦  IDX  ¦  NODE DESCRIPTION                                          ¦   ROWS ¦
--------------------------------------------------------------------------------
¦ 0 ¦  SELECT STATEMENT                                             ¦  11620 ¦
¦ 1 ¦    QUERY BLOCK ("$QB_IDX_2")                                  ¦  11620 ¦
¦ 2 ¦      GROUP HASH INSTANT                                       ¦  11620 ¦
¦ 3 ¦        NESTED JOIN (INNER JOIN)                               ¦  30519 ¦
¦ 4 ¦          NESTED JOIN (INNER JOIN)                             ¦ 147126 ¦
¦ 5 ¦            TABLE ACCESS ("CUSTOMER")                          ¦  30142 ¦
¦ 6 ¦            INDEX ACCESS ("ORDERS", "ORDERS_CUSTKEY_FK")       ¦ 147126 ¦
¦ 7 ¦          INDEX ACCESS ("LINEITEM", "LINEITEM_ORDERKEY_FK")    ¦  30519 ¦
================================================================================
     1  -  TARGET : LINEITEM.L_ORDERKEY, ROUND(SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 -
LINEITEM.L_DISCOUNT ) ),2) AS REVENUE, ORDERS.O_ORDERDATE, ORDERS.O_SHIPPRIORITY
     2  -  GROUP KEY : LINEITEM.L_ORDERKEY, ORDERS.O_ORDERDATE, ORDERS.O_SHIPPRIORITY
           RECORD COLUMN : SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 - LINEITEM.L_DISCOUNT ) )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY, ORDERS.O_ORDERDATE, ORDERS.O_SHIPPRIORITY
           READ RECORD COLUMN : SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 - LINEITEM.L_DISCOUNT ) )
```

```
     3  -  JOINED COLUMN : LINEITEM.L_ORDERKEY, ORDERS.O_ORDERDATE, ORDERS.O_SHIPPRIORITY,
LINEITEM.L_EXTENDEDPRICE, LINEITEM.L_DISCOUNT
     4  -  JOINED COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_ORDERDATE, ORDERS.O_SHIPPRIORITY
     5  -  READ COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_MKTSEGMENT
             PHYSICAL FILTER : CUSTOMER.C_MKTSEGMENT = 'BUILDING'
     6  -  READ INDEX COLUMN : ORDERS.O_CUSTKEY
           READ TABLE COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_ORDERDATE, ORDERS.O_SHIPPRIORITY
             MIN RANGE : ORDERS.O_CUSTKEY = {CUSTOMER.C_CUSTKEY}
             MAX RANGE : ORDERS.O_CUSTKEY = {CUSTOMER.C_CUSTKEY}
             PHYSICAL TABLE FILTER : ORDERS.O_ORDERDATE < DATE'1995-03-15'
     7  -  READ INDEX COLUMN : LINEITEM.L_ORDERKEY
           READ TABLE COLUMN : LINEITEM.L_EXTENDEDPRICE, LINEITEM.L_DISCOUNT,
LINEITEM.L_SHIPDATE
             MIN RANGE : LINEITEM.L_ORDERKEY = {ORDERS.O_ORDERKEY}
             MAX RANGE : LINEITEM.L_ORDERKEY = {ORDERS.O_ORDERKEY}
             PHYSICAL TABLE FILTER : LINEITEM.L_SHIPDATE > DATE'1995-03-15'
<<<  end print plan
```

LEADING(table_name [[,] table_name])

If LEADING(table_name [[,] table_name]) hint is specified, then an optimizer orders to perform joinin g in an order described in the hint when performing the join ordering. Unlike an ORDERING hint, LEADIN G(table_name [[,] table_name]) hint can not specify the location of tables, but it can only specify the order of tables participating in the join ordering.

The following is an example of using LEADING hint.

```
\EXPLAIN PLAN
  SELECT  /*+ LEADING( customer, orders, lineitem ) */
          l_orderkey,
          ROUND( sum(l_extendedprice*(1-l_discount)), 2) as revenue,
          o_orderdate,
          o_shippriority
   FROM   lineitem,
          orders,
          customer
  WHERE   c_mktsegment = 'BUILDING'
    AND   c_custkey = o_custkey
    AND   l_orderkey = o_orderkey
    AND   o_orderdate < date '1995-03-15'
    AND   l_shipdate > date '1995-03-15'
GROUP BY  l_orderkey,
```

```
          o_orderdate,
          o_shippriority;
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                                          |   ROWS |
--------------------------------------------------------------------------------
| 0 |  SELECT STATEMENT                                              |  11620 |
| 1 |    QUERY BLOCK ("$QB_IDX_2")                                   |  11620 |
| 2 |      GROUP HASH INSTANT                                        |  11620 |
| 3 |        NESTED JOIN (INNER JOIN)                                |  30519 |
| 4 |          NESTED JOIN (INNER JOIN)                              | 147126 |
| 5 |            TABLE ACCESS ("CUSTOMER")                           |  30142 |
| 6 |            INDEX ACCESS ("ORDERS", "ORDERS_CUSTKEY_FK")        | 147126 |
| 7 |          INDEX ACCESS ("LINEITEM", "LINEITEM_ORDERKEY_FK")     |  30519 |
================================================================================
 1  -  TARGET : LINEITEM.L_ORDERKEY, ROUND(SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 -
LINEITEM.L_DISCOUNT ) ),2) AS REVENUE, ORDERS.O_ORDERDATE, ORDERS.O_SHIPPRIORITY
     2  -  GROUP KEY : LINEITEM.L_ORDERKEY, ORDERS.O_ORDERDATE, ORDERS.O_SHIPPRIORITY
           RECORD COLUMN : SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 - LINEITEM.L_DISCOUNT ) )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY, ORDERS.O_ORDERDATE, ORDERS.O_SHIPPRIORITY
           READ RECORD COLUMN : SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 - LINEITEM.L_DISCOUNT ) )
     3  -  JOINED COLUMN : LINEITEM.L_ORDERKEY, ORDERS.O_ORDERDATE, ORDERS.O_SHIPPRIORITY,
LINEITEM.L_EXTENDEDPRICE, LINEITEM.L_DISCOUNT
     4  -  JOINED COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_ORDERDATE, ORDERS.O_SHIPPRIORITY
     5  -  READ COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_MKTSEGMENT
             PHYSICAL FILTER : CUSTOMER.C_MKTSEGMENT = 'BUILDING'
     6  -  READ INDEX COLUMN : ORDERS.O_CUSTKEY
           READ TABLE COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_ORDERDATE, ORDERS.O_SHIPPRIORITY
             MIN RANGE : ORDERS.O_CUSTKEY = {CUSTOMER.C_CUSTKEY}
             MAX RANGE : ORDERS.O_CUSTKEY = {CUSTOMER.C_CUSTKEY}
             PHYSICAL TABLE FILTER : ORDERS.O_ORDERDATE < DATE'1995-03-15'
     7  -  READ INDEX COLUMN : LINEITEM.L_ORDERKEY
           READ TABLE COLUMN : LINEITEM.L_EXTENDEDPRICE, LINEITEM.L_DISCOUNT,
LINEITEM.L_SHIPDATE
             MIN RANGE : LINEITEM.L_ORDERKEY = {ORDERS.O_ORDERKEY}
             MAX RANGE : LINEITEM.L_ORDERKEY = {ORDERS.O_ORDERKEY}
             PHYSICAL TABLE FILTER : LINEITEM.L_SHIPDATE > DATE'1995-03-15'
<<<  end print plan
```

〈join operation hints〉

USE_HASH(table_name [[,] table_name])

If USE_HASH(table_name [[,] table_name]) hint is specified, then an optimizer selects hash join when determining the join operation of the join in which table_name participates.

One or more table should be specified, and the same tables can not be specified more than two. If 〈join operation hints〉 already exists in the specified table, then the hint is ignored. Also, if each different join operation hint is specified in two tables participating in join, then the hint specified in a right node(inner node) is preferentially applied.

The hint is applied only when a equi-join condition exists even though USE_HASH hint is specified. If a join condition which is capable of hash join does not exist, then an optimizer selects the best join operation through a cost estimation.

The following is an example of using USE_HASH(table_name [[,] table_name]) hint.

```
\EXPLAIN PLAN
  SELECT  /*+ USE_HASH( orders ) */
          c_name,
          o_orderdate,
          o_orderstatus
    FROM  customer,
          orders
   WHERE  c_custkey = o_custkey
     AND  o_orderdate >= date '1995-03-15'
     AND  o_orderdate < date '1995-03-15' + interval '1' month;
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                                        |   ROWS |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                        |  19343 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                             |  19343 |
|    2  |      HASH JOIN (INNER JOIN)                              |  19343 |
|    3  |        TABLE ACCESS ("CUSTOMER")                         | 150000 |
|    4  |        HASH JOIN INSTANT                                 |  19343 |
|    5  |          TABLE ACCESS ("ORDERS")                         |  19343 |
================================================================================
     1  -  TARGET : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     2  -  JOINED COLUMN : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     3  -  READ COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_NAME
     4  -  HASH KEY : ORDERS.O_CUSTKEY
```

```
           RECORD COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
           READ KEY COLUMN : ORDERS.O_CUSTKEY, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
             HASH FILTER : ORDERS.O_CUSTKEY = CUSTOMER.C_CUSTKEY
     5  -  READ COLUMN : ORDERS.O_CUSTKEY, ORDERS.O_ORDERSTATUS, ORDERS.O_ORDERDATE
             PHYSICAL FILTER : ORDERS.O_ORDERDATE < DATE'1995-03-15' + CAST( '1' AS
INTERVAL(MONTH) ) AND ORDERS.O_ORDERDATE >= DATE'1995-03-15'
<<<  end print plan
```

USE_HASH(table_name [[,] table_name] , hash_bucket_count)

USE_HASH(table_name [[,] table_name] , hash_bucket_count) hint is as same as UNNEST_HASH hint, but in addition to that it can specify the hash bucket count.

If statistics information is incorrect, then hash bucket count of hash instants created for hash join may be too many or too little so it may degrade the performance. In this case, specify the hash bucket count by u sing the hint to prevent the performance from degrading.

The following is an example of using USE_HASH(table_name [[,] table_name], hash_bucket_count) hi nt.

```
SELECT COUNT(DISTINCT o_custkey)
  FROM orders
 WHERE o_orderdate >= date '1995-03-15'
   AND o_orderdate < date '1995-03-15' + interval '1' month;
COUNT(DISTINCT O_CUSTKEY)
-------------------------
                    17430
1 row selected.
\EXPLAIN PLAN VERBOSE
  SELECT  /*+ USE_HASH( orders, 17430 ) */
          c_name,
          o_orderdate,
          o_orderstatus
    FROM  customer,
          orders
   WHERE  c_custkey = o_custkey
     AND  o_orderdate >= date '1995-03-15'
     AND  o_orderdate < date '1995-03-15' + interval '1' month;
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                              |  ROWS | Ellipsis  |
--------------------------------------------------------------------------------
```

```
|  0  |  SELECT STATEMENT                                |  19343 | ... |
|  1  |    QUERY BLOCK ("$QB_IDX_2")                     |  19343 | ... |
|  2  |      HASH JOIN (INNER JOIN)                      |  19343 | ... |
|  3  |        TABLE ACCESS ("CUSTOMER")                 | 150000 | ... |
|  4  |        HASH JOIN INSTANT                         |  19343 | ... |
|  5  |          TABLE ACCESS ("ORDERS")                 |  19343 | ... |
=============================================================================
     1  -  TARGET : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     2  -  JOINED COLUMN : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     3  -  READ COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_NAME
     4  -  HASH KEY : ORDERS.O_CUSTKEY
           RECORD COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
           READ KEY COLUMN : ORDERS.O_CUSTKEY, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
             HASH FILTER : ORDERS.O_CUSTKEY = CUSTOMER.C_CUSTKEY
           HASH BUCKET COUNT : 17430
     5  -  READ COLUMN : ORDERS.O_CUSTKEY, ORDERS.O_ORDERSTATUS, ORDERS.O_ORDERDATE
             PHYSICAL FILTER : ORDERS.O_ORDERDATE < DATE'1995-03-15' + CAST( '1' AS
INTERVAL(MONTH) ) AND ORDERS.O_ORDERDATE >= DATE'1995-03-15'
<<<  end print plan
```

In the example above, the hash bucket count in an output of orders participating in the join is specified a s many as the number of distinct values.

USE_HASH_IN(alias)

If USE_HASH_IN(alias) hint is specified, then an optimizer selects hash join when determining the join o peration of the join in which the alias participates. Then it places the alias on the right(inner) of the join.

If <join operation hints> already exists in the specified alias, then the hint is ignored. Also, if each differen t join operation hint is specified in two tables participating in join, then the hint specified in a right node(i nner node) is preferentially applied.

A table name, a table alias name, a view name, a view alias name and a join alias name can be an alias.

The following is an example of using USE_HASH_IN(alias) hint.

```
--# hash join
\EXPLAIN PLAN
  SELECT  /*+ USE_HASH_IN( orders ) */
          c_name,
          o_orderdate,
          o_orderstatus
    FROM  customer,
```

```
          orders
   WHERE  c_custkey = o_custkey
     AND  o_orderdate >= date '1995-03-15'
     AND  o_orderdate < date '1995-03-15' + interval '1' month;
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                                        |   ROWS |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                        |  19343 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                             |  19343 |
|    2  |      HASH JOIN (INNER JOIN)                              |  19343 |
|    3  |        TABLE ACCESS ("CUSTOMER")                         | 150000 |
|    4  |        HASH JOIN INSTANT                                 |  19343 |
|    5  |          TABLE ACCESS ("ORDERS")                         |  19343 |
================================================================================
     1  -  TARGET : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     2  -  JOINED COLUMN : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     3  -  READ COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_NAME
     4  -  HASH KEY : ORDERS.O_CUSTKEY
           RECORD COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
           READ KEY COLUMN : ORDERS.O_CUSTKEY, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
             HASH FILTER : ORDERS.O_CUSTKEY = CUSTOMER.C_CUSTKEY
     5  -  READ COLUMN : ORDERS.O_CUSTKEY, ORDERS.O_ORDERSTATUS, ORDERS.O_ORDERDATE
             PHYSICAL FILTER : ORDERS.O_ORDERDATE < DATE'1995-03-15' + CAST( '1' AS
INTERVAL(MONTH) ) AND ORDERS.O_ORDERDATE >= DATE'1995-03-15'
<<<  end print plan
```

The following is an example of using USE_HASH_IN(alias) hint by using join alias.

```
\EXPLAIN PLAN
  SELECT  /*+ USE_HASH_IN( j1 )  */
          l_orderkey,
          ROUND( sum(l_extendedprice*(1-l_discount)), 2) as revenue,
          o_orderdate,
          o_shippriority
   FROM   ( customer INNER JOIN orders ON  c_mktsegment = 'BUILDING'
                                       AND c_custkey = o_custkey
                                       AND o_orderdate < date '1995-03-15'
          ) ALIAS j1
          INNER JOIN lineitem ON  l_orderkey = o_orderkey
```

```
                              AND l_shipdate > date '1995-03-15'
GROUP BY  l_orderkey,
          o_orderdate,
          o_shippriority;
>>>  start print plan
< Execution Plan >
============================================================================
| IDX |  NODE DESCRIPTION                                        |    ROWS |
----------------------------------------------------------------------------
| 0 |  SELECT STATEMENT                                          |   11620 |
| 1 |    QUERY BLOCK ("$QB_IDX_2")                               |   11620 |
| 2 |      GROUP HASH INSTANT                                    |   11620 |
| 3 |        HASH JOIN (INNER JOIN)                              |   30519 |
| 4 |          TABLE ACCESS ("LINEITEM")                         | 3241776 |
| 5 |          HASH JOIN INSTANT                                 |   30519 |
| 6 |            NESTED JOIN (INNER JOIN)                        |  147126 |
| 7 |              TABLE ACCESS ("CUSTOMER")                     |   30142 |
| 8 |              INDEX ACCESS ("ORDERS", "ORDERS_CUSTKEY_FK") |  147126 |
============================================================================
     1  -  TARGET : LINEITEM.L_ORDERKEY, ROUND(SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 -
LINEITEM.L_DISCOUNT ) ),2) AS REVENUE, ORDERS.O_ORDERDATE, ORDERS.O_SHIPPRIORITY
     2  -  GROUP KEY : LINEITEM.L_ORDERKEY, ORDERS.O_ORDERDATE, ORDERS.O_SHIPPRIORITY
           RECORD COLUMN : SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 - LINEITEM.L_DISCOUNT ) )
           READ KEY COLUMN : LINEITEM.L_ORDERKEY, ORDERS.O_ORDERDATE, ORDERS.O_SHIPPRIORITY
           READ RECORD COLUMN : SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 - LINEITEM.L_DISCOUNT ) )
     3  -  JOINED COLUMN : LINEITEM.L_ORDERKEY, ORDERS.O_ORDERDATE, ORDERS.O_SHIPPRIORITY,
LINEITEM.L_EXTENDEDPRICE, LINEITEM.L_DISCOUNT
     4  -  READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_EXTENDEDPRICE, LINEITEM.L_DISCOUNT,
LINEITEM.L_SHIPDATE
             PHYSICAL FILTER : LINEITEM.L_SHIPDATE > DATE'1995-03-15'
     5  -  HASH KEY : ORDERS.O_ORDERKEY
           RECORD COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_SHIPPRIORITY
           READ KEY COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_ORDERDATE, ORDERS.O_SHIPPRIORITY
             HASH FILTER : ORDERS.O_ORDERKEY = LINEITEM.L_ORDERKEY
     6  -  JOINED COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_ORDERDATE, ORDERS.O_SHIPPRIORITY
     7  -  READ COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_MKTSEGMENT
             PHYSICAL FILTER : CUSTOMER.C_MKTSEGMENT = 'BUILDING'
     8  -  READ INDEX COLUMN : ORDERS.O_CUSTKEY
           READ TABLE COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_ORDERDATE, ORDERS.O_SHIPPRIORITY
             MIN RANGE : ORDERS.O_CUSTKEY = {CUSTOMER.C_CUSTKEY}
             MAX RANGE : ORDERS.O_CUSTKEY = {CUSTOMER.C_CUSTKEY}
```

```
            PHYSICAL TABLE FILTER : ORDERS.O_ORDERDATE < DATE'1995-03-15'
<<<  end print plan
```

In the execution plan above, join alias j10| is located on the right of the hash join.

USE_HASH_OUT(alias)

If USE_HASH_OUT(alias) hint is specified, then an optimizer selects hash join when determining the join operation of the join in which the alias participates. Then it places the alias on the left(outer) of the join.

If <join operation hints> already exists in the specified alias, then the hint is ignored. Also, if each differen t join operation hint is specified in two tables participating in join, then the hint specified in a right node(i nner node) is preferentially applied.

A table name, a table alias name, a view name, a view alias name and a join alias name can be an alias.

The following is an example of using USE_HASH_OUT(alias) hint.

```
\EXPLAIN PLAN
  SELECT  /*+ USE_HASH_OUT( orders ) */
          c_name,
          o_orderdate,
          o_orderstatus
    FROM  customer,
          orders
   WHERE  c_custkey = o_custkey
     AND  o_orderdate >= date '1995-03-15'
     AND  o_orderdate < date '1995-03-15' + interval '1' month;
>>>  start print plan
< Execution Plan >
=============================================================================¦  IDX  ¦  NODE
DESCRIPTION                                        ¦   ROWS
¦-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                          ¦  19343 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                             ¦  19343 ¦
¦    2  ¦      HASH JOIN (INNER JOIN)                              ¦  19343 ¦
¦    3  ¦        TABLE ACCESS ("ORDERS")                           ¦  19343 ¦
¦    4  ¦        HASH JOIN INSTANT                                 ¦  19343 ¦
¦    5  ¦          TABLE ACCESS ("CUSTOMER")                       ¦ 150000 ¦
=============================================================================
     1  -  TARGET : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     2  -  JOINED COLUMN : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     3  -  READ COLUMN : ORDERS.O_CUSTKEY, ORDERS.O_ORDERSTATUS, ORDERS.O_ORDERDATE
             PHYSICAL FILTER : ORDERS.O_ORDERDATE < DATE'1995-03-15' + CAST( '1' AS
```

```
INTERVAL(MONTH) ) AND ORDERS.O_ORDERDATE >= DATE'1995-03-15'
     4  -  HASH KEY : CUSTOMER.C_CUSTKEY
           RECORD COLUMN : CUSTOMER.C_NAME
           READ KEY COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_NAME
             HASH FILTER : CUSTOMER.C_CUSTKEY = ORDERS.O_CUSTKEY
           FETCH ONE ROW
     5  -  READ COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_NAME
<<<  end print plan
```

NO_USE_HASH(table_name [[,] table_name])

If NO_USE_HASH(table_name [[,] table_name]) hint is specified, then an optimizer excludes hash join when determining the join operation of the join in which table_name participates. Therefore, it selects th e join operation whose cost estimation is the best among other join operations except for hash join.

One or more table should be specified, and the same tables can not be specified more than two. If <join operation hints> already exists in the specified table, then the hint is ignored. Also, if each different join o peration hint is specified in two tables participating in join, then the hint specified in a right node(inner n ode) is preferentially applied.

The following is an example of using NO_USE_HASH(table_name [[,] table_name]) hint.

```
\EXPLAIN PLAN
  SELECT  /*+ NO_USE_HASH( orders ) */
          c_name,
          o_orderdate,
          o_orderstatus
    FROM  customer,
          orders
   WHERE  c_custkey = o_custkey
     AND  o_orderdate >= date '1995-03-15'
     AND  o_orderdate < date '1995-03-15' + interval '1' month;
>>>  start print plan
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                                          | ROWS |
-----------------------------------------------------------------------------
|  0  |  SELECT STATEMENT                                          | 19343 |
|  1  |    QUERY BLOCK ("$QB_IDX_2")                               | 19343 |
|  2  |      NESTED JOIN (INNER JOIN)                              | 19343 |
|  3  |        TABLE ACCESS ("ORDERS")                             | 19343 |
|  4  |        INDEX ACCESS ("CUSTOMER", "CUSTOMER_PK_INDEX")      | 19343 |
=============================================================================
```

```
     1  -  TARGET : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     2  -  JOINED COLUMN : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     3  -  READ COLUMN : ORDERS.O_CUSTKEY, ORDERS.O_ORDERSTATUS, ORDERS.O_ORDERDATE
             PHYSICAL FILTER : ORDERS.O_ORDERDATE < DATE'1995-03-15' + CAST( '1' AS
INTERVAL(MONTH) ) AND ORDERS.O_ORDERDATE >= DATE'1995-03-15'
     4  -  READ INDEX COLUMN : CUSTOMER.C_CUSTKEY
           READ TABLE COLUMN : CUSTOMER.C_NAME
             MIN RANGE : CUSTOMER.C_CUSTKEY = {ORDERS.O_CUSTKEY}
             MAX RANGE : CUSTOMER.C_CUSTKEY = {ORDERS.O_CUSTKEY}
           FETCH ONE ROW
<<<  end print plan
```

USE_MERGE(table_name [[,] table_name])

If USE_MERGE(table_name [[,] table_name]) hint is specified, then an optimizer selects merge join w hen determining the join operation of the join in which table_name participates.

One or more table should be specified, and the same tables can not be specified more than two. If <join operation hints> already exists in the specified table, then the hint is ignored. Also, if each different join o peration hint is specified in two tables participating in join, then the hint specified in a right node(inner n ode) is preferentially applied.

The following is an example of using USE_MERGE(table_name [[,] table_name]) hint.

```
\EXPLAIN PLAN
  SELECT  /*+ USE_MERGE( orders ) */
          c_name,
          o_orderdate,
          o_orderstatus
    FROM  customer,
          orders
   WHERE  c_custkey = o_custkey
     AND  o_orderdate >= date '1995-03-15'
     AND  o_orderdate < date '1995-03-15' + interval '1' month;
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                                          |   ROWS |
--------------------------------------------------------------------------------
|  0  |  SELECT STATEMENT                                            |  19343 |
|  1  |    QUERY BLOCK ("$QB_IDX_2")                                 |  19343 |
|  2  |      MERGE JOIN (INNER JOIN)                                 |  19343 |
|  3  |        INDEX ACCESS ("CUSTOMER", "CUSTOMER_PK_INDEX")        | 150000 |
```

```
|  4  |        SORT JOIN INSTANT                                  |  19343 |
|  5  |          TABLE ACCESS ("ORDERS")                          |  19343 |
=============================================================================
     1  -  TARGET : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     2  -  JOINED COLUMN : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
             ON FILTER (Equi) : CUSTOMER.C_CUSTKEY = ORDERS.O_CUSTKEY
     3  -  READ INDEX COLUMN : CUSTOMER.C_CUSTKEY
           READ TABLE COLUMN : CUSTOMER.C_NAME
     4  -  SORT KEY : "ORDERS.O_CUSTKEY ASC NULLS LAST"
           RECORD COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
           READ KEY COLUMN : ORDERS.O_CUSTKEY
           READ RECORD COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
             MIN RANGE : ORDERS.O_CUSTKEY >= {CUSTOMER.C_CUSTKEY}
             MAX RANGE : ORDERS.O_CUSTKEY IS NOT NULL
     5  -  READ COLUMN : ORDERS.O_CUSTKEY, ORDERS.O_ORDERSTATUS, ORDERS.O_ORDERDATE
             PHYSICAL FILTER : ORDERS.O_ORDERDATE < DATE'1995-03-15' + CAST( '1' AS
INTERVAL(MONTH) ) AND ORDERS.O_ORDERDATE >= DATE'1995-03-15'
<<<  end print plan
```

USE_MERGE_IN(alias)

If USE_MERGE_IN(alias) hint is specified, then an optimizer selects merge join when determining the join operation of the join in which the alias participates. Then it places the alias on the right(inner) of the join.

If <join operation hints> already exists in the specified alias, then the hint is ignored. Also, if each different join operation hint is specified in two tables participating in join, then the hint specified in a right node(inner node) is preferentially applied.

A table name, a table alias name, a view name, a view alias name and a join alias name can be an alias.

The following is an example of using USE_MERGE_IN(alias) hint.

```
\EXPLAIN PLAN
  SELECT  /*+ USE_MERGE_IN( orders ) */
          c_name,
          o_orderdate,
          o_orderstatus
    FROM  customer,
          orders
   WHERE  c_custkey = o_custkey
     AND  o_orderdate >= date '1995-03-15'
     AND  o_orderdate < date '1995-03-15' + interval '1' month;
```

```
< Execution Plan >
============================================================================
|  IDX  |  NODE DESCRIPTION                                      |   ROWS |
----------------------------------------------------------------------------
|  0  |  SELECT STATEMENT                                        |  19343 |
|  1  |    QUERY BLOCK ("$QB_IDX_2")                             |  19343 |
|  2  |      MERGE JOIN (INNER JOIN)                             |  19343 |
|  3  |        INDEX ACCESS ("CUSTOMER", "CUSTOMER_PK_INDEX")    | 150000 |
|  4  |        SORT JOIN INSTANT                                 |  19343 |
|  5  |          TABLE ACCESS ("ORDERS")                         |  19343 |
============================================================================
     1  -  TARGET : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     2  -  JOINED COLUMN : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
             ON FILTER (Equi) : CUSTOMER.C_CUSTKEY = ORDERS.O_CUSTKEY
     3  -  READ INDEX COLUMN : CUSTOMER.C_CUSTKEY
           READ TABLE COLUMN : CUSTOMER.C_NAME
     4  -  SORT KEY : "ORDERS.O_CUSTKEY ASC NULLS LAST"
           RECORD COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
           READ KEY COLUMN : ORDERS.O_CUSTKEY
           READ RECORD COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
             MIN RANGE : ORDERS.O_CUSTKEY >= {CUSTOMER.C_CUSTKEY}
             MAX RANGE : ORDERS.O_CUSTKEY IS NOT NULL
     5  -  READ COLUMN : ORDERS.O_CUSTKEY, ORDERS.O_ORDERSTATUS, ORDERS.O_ORDERDATE
             PHYSICAL FILTER : ORDERS.O_ORDERDATE < DATE'1995-03-15' + CAST( '1' AS
INTERVAL(MONTH) ) AND ORDERS.O_ORDERDATE >= DATE'1995-03-15'
<<<  end print plan
```

USE_MERGE_OUT(alias)

If USE_MERGE_OUT(alias) hint is specified, then an optimizer selects merge join when determining the j oin operation of the join in which the alias participates. Then it places the alias on the left(outer)of the joi n.

If <join operation hints> already exists in the specified alias, then the hint is ignored. Also, if each differen t join operation hint is specified in two tables participating in join, then the hint specified in a right node(i nner node) is preferentially applied.

The following is an example of using USE_MERGE_OUT(alias) hint.

```
\EXPLAIN PLAN
  SELECT  /*+ USE_MERGE_OUT( orders ) */
          c_name,
          o_orderdate,
```

```
          o_orderstatus
    FROM  customer,
          orders
   WHERE  c_custkey = o_custkey
     AND  o_orderdate >= date '1995-03-15'
     AND  o_orderdate < date '1995-03-15' + interval '1' month;
< Execution Plan >
============================================================================
|  IDX  |  NODE DESCRIPTION                                        |   ROWS |
----------------------------------------------------------------------------
|  0  |  SELECT STATEMENT                                          |  19343 |
|  1  |    QUERY BLOCK ("$QB_IDX_2")                               |  19343 |
|  2  |      MERGE JOIN (INNER JOIN)                               |  19343 |
|  3  |        SORT JOIN INSTANT                                   |  19343 |
|  4  |          TABLE ACCESS ("ORDERS")                           |  19343 |
|  5  |        INDEX ACCESS ("CUSTOMER", "CUSTOMER_PK_INDEX")      | 149991 |
============================================================================
     1  -  TARGET : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     2  -  JOINED COLUMN : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
             ON FILTER (Equi) : ORDERS.O_CUSTKEY = CUSTOMER.C_CUSTKEY
     3  -  SORT KEY : "ORDERS.O_CUSTKEY ASC NULLS LAST"
           RECORD COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
           READ KEY COLUMN : ORDERS.O_CUSTKEY
           READ RECORD COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     4  -  READ COLUMN : ORDERS.O_CUSTKEY, ORDERS.O_ORDERSTATUS, ORDERS.O_ORDERDATE
             PHYSICAL FILTER : ORDERS.O_ORDERDATE < DATE'1995-03-15' + CAST( '1' AS
INTERVAL(MONTH) ) AND ORDERS.O_ORDERDATE >= DATE'1995-03-15'
     5  -  READ INDEX COLUMN : CUSTOMER.C_CUSTKEY
           READ TABLE COLUMN : CUSTOMER.C_NAME
             MIN RANGE : CUSTOMER.C_CUSTKEY >= {ORDERS.O_CUSTKEY}
             MAX RANGE : CUSTOMER.C_CUSTKEY IS NOT NULL
<<<  end print plan
```

NO_USE_MERGE(table_name [[,] table_name])

If NO_USE_MERGE(table_name [[,] table_name]) hint is specified, then an optimizer excludes merge join when determining the join operation of the join in which table_name participates. Therefore, it selects the join operation whose cost estimation is the best among other join operations except for merge join.

One or more table should be specified, and the same tables can not be specified more than two. If <join operation hints> already exists in the specified table, then the hint is ignored. Also, if each different join operation hint is specified in two tables participating in join, then the hint specified in a right node(inner n

ode) is preferentially applied.

The following is an example of using NO_USE_MERGE(table_name [[,] table_name]) hint.

```
\EXPLAIN PLAN
  SELECT  /*+ NO_USE_MERGE( orders ) */
          c_name,
          o_orderdate,
          o_orderstatus
    FROM  customer,
          orders
   WHERE  c_custkey = o_custkey
     AND  o_orderdate >= date '1995-03-15'
     AND  o_orderdate < date '1995-03-15' + interval '1' month;
< Execution Plan >
================================================================================
¦  IDX  ¦  NODE DESCRIPTION                                          ¦ ROWS ¦
--------------------------------------------------------------------------------
¦  0  ¦  SELECT STATEMENT                                           ¦ 19343 ¦
¦  1  ¦    QUERY BLOCK ("$QB_IDX_2")                              ¦ 19343 ¦
¦  2  ¦      NESTED JOIN (INNER JOIN)                             ¦ 19343 ¦
¦  3  ¦        TABLE ACCESS ("ORDERS")                            ¦ 19343 ¦
¦  4  ¦        INDEX ACCESS ("CUSTOMER", "CUSTOMER_PK_INDEX")     ¦ 19343 ¦
================================================================================
     1  -  TARGET : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     2  -  JOINED COLUMN : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     3  -  READ COLUMN : ORDERS.O_CUSTKEY, ORDERS.O_ORDERSTATUS, ORDERS.O_ORDERDATE
             PHYSICAL FILTER : ORDERS.O_ORDERDATE < DATE'1995-03-15' + CAST( '1' AS
INTERVAL(MONTH) ) AND ORDERS.O_ORDERDATE >= DATE'1995-03-15'
     4  -  READ INDEX COLUMN : CUSTOMER.C_CUSTKEY
           READ TABLE COLUMN : CUSTOMER.C_NAME
             MIN RANGE : CUSTOMER.C_CUSTKEY = {ORDERS.O_CUSTKEY}
             MAX RANGE : CUSTOMER.C_CUSTKEY = {ORDERS.O_CUSTKEY}
           FETCH ONE ROW
<<<  end print plan
```

USE_NL(table_name [[,] table_name])

If USE_NL(table_name [[,] table_name]) hint is specified, then an optimizer selects nested loop join w hen determining the join operation of the join in which table_name participates.

One or more table should be specified, and the same tables can not be specified more than two. If <join operation hints> already exists in the specified table, then the hint is ignored. Also, if each different join o

peration hint is specified in two tables participating in join, then the hint specified in a right node(inner n ode) is preferentially applied.

The following is an example of using USE_NL(table_name [[,] table_name]) hint.

```
\EXPLAIN PLAN
  SELECT  /*+ USE_NL( orders ) */
          c_name,
          o_orderdate,
          o_orderstatus
    FROM  customer,
          orders
   WHERE  c_custkey = o_custkey
     AND  o_orderdate >= date '1995-03-15'
     AND  o_orderdate < date '1995-03-15' + interval '1' month;
< Execution Plan >
============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                        ¦ ROWS ¦
----------------------------------------------------------------------------
¦  0  ¦  SELECT STATEMENT                                         ¦ 19343 ¦
¦  1  ¦    QUERY BLOCK ("$QB_IDX_2")                            ¦ 19343 ¦
¦  2  ¦      NESTED JOIN (INNER JOIN)                           ¦ 19343 ¦
¦  3  ¦        TABLE ACCESS ("ORDERS")                          ¦ 19343 ¦
¦  4  ¦        INDEX ACCESS ("CUSTOMER", "CUSTOMER_PK_INDEX")   ¦ 19343 ¦
============================================================================
     1  -  TARGET : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     2  -  JOINED COLUMN : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     3  -  READ COLUMN : ORDERS.O_CUSTKEY, ORDERS.O_ORDERSTATUS, ORDERS.O_ORDERDATE
             PHYSICAL FILTER : ORDERS.O_ORDERDATE < DATE'1995-03-15' + CAST( '1' AS
INTERVAL(MONTH) ) AND ORDERS.O_ORDERDATE >= DATE'1995-03-15'
     4  -  READ INDEX COLUMN : CUSTOMER.C_CUSTKEY
           READ TABLE COLUMN : CUSTOMER.C_NAME
             MIN RANGE : CUSTOMER.C_CUSTKEY = {ORDERS.O_CUSTKEY}
             MAX RANGE : CUSTOMER.C_CUSTKEY = {ORDERS.O_CUSTKEY}
           FETCH ONE ROW
<<<  end print plan
```

USE_NL_IN(alias)

If USE_NL_IN(alias) hint is specified, then an optimizer selects nested loop join when determining the joi n operation of the join in which the alias participates. Then it places the alias on the right(inner) of the joi n.

A table name, a table alias name, a view name, a view alias name and a join alias name can be an alias.

The following is an example of using USE_NL_IN(alias) hint.

```
\EXPLAIN PLAN
  SELECT  /*+ USE_NL_IN( orders ) */
          c_name,
          o_orderdate,
          o_orderstatus
    FROM  customer,
          orders
   WHERE  c_custkey = o_custkey
     AND  o_orderdate >= date '1995-03-15'
     AND  o_orderdate < date '1995-03-15' + interval '1' month;
>>>  start print plan
< Execution Plan >
================================================================================
¦ IDX ¦  NODE DESCRIPTION                                          ¦   ROWS ¦
--------------------------------------------------------------------------------
¦  0  ¦  SELECT STATEMENT                                          ¦  19343 ¦
¦  1  ¦    QUERY BLOCK ("$QB_IDX_2")                             ¦  19343 ¦
¦  2  ¦      NESTED JOIN (INNER JOIN)                            ¦  19343 ¦
¦  3  ¦        TABLE ACCESS ("CUSTOMER")                         ¦ 150000 ¦
¦  4  ¦        INDEX ACCESS ("ORDERS", "ORDERS_CUSTKEY_FK")      ¦  19343 ¦
================================================================================
     1  -  TARGET : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     2  -  JOINED COLUMN : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     3  -  READ COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_NAME
     4  -  READ INDEX COLUMN : ORDERS.O_CUSTKEY
           READ TABLE COLUMN : ORDERS.O_ORDERSTATUS, ORDERS.O_ORDERDATE
             MIN RANGE : ORDERS.O_CUSTKEY = {CUSTOMER.C_CUSTKEY}
             MAX RANGE : ORDERS.O_CUSTKEY = {CUSTOMER.C_CUSTKEY}
             PHYSICAL TABLE FILTER : ORDERS.O_ORDERDATE < DATE'1995-03-15' + CAST( '1' AS
INTERVAL(MONTH) ) AND ORDERS.O_ORDERDATE >= DATE'1995-03-15'
<<<  end print plan
```

USE_NL_OUT(alias)

If USE_NL_OUT(alias) hint is specified, then an optimizer selects nested loop join when determining the j oin operation of the join in which the alias participates. Then it places the alias on the left(outer) of the jo in.

A table name, a table alias name, a view name, a view alias name and a join alias name can be an alias.

The following is an example of using USE_NL_OUT(alias) hint.

```
\EXPLAIN PLAN
  SELECT  /*+ USE_NL_OUT( orders ) */
          c_name,
          o_orderdate,
          o_orderstatus
    FROM  customer,
          orders
   WHERE  c_custkey = o_custkey
     AND  o_orderdate >= date '1995-03-15'
     AND  o_orderdate < date '1995-03-15' + interval '1' month;
< Execution Plan >
----------------------------------------------------------------------------
|  IDX  |  NODE DESCRIPTION                                          | ROWS |
----------------------------------------------------------------------------
|  0  |  SELECT STATEMENT                                           | 19343 |
|  1  |    QUERY BLOCK ("$QB_IDX_2")                                | 19343 |
|  2  |      NESTED JOIN (INNER JOIN)                               | 19343 |
|  3  |        TABLE ACCESS ("ORDERS")                              | 19343 |
|  4  |        INDEX ACCESS ("CUSTOMER", "CUSTOMER_PK_INDEX")       | 19343 |
----------------------------------------------------------------------------
     1  -  TARGET : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     2  -  JOINED COLUMN : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     3  -  READ COLUMN : ORDERS.O_CUSTKEY, ORDERS.O_ORDERSTATUS, ORDERS.O_ORDERDATE
             PHYSICAL FILTER : ORDERS.O_ORDERDATE < DATE'1995-03-15' + CAST( '1' AS
INTERVAL(MONTH) ) AND ORDERS.O_ORDERDATE >= DATE'1995-03-15'
     4  -  READ INDEX COLUMN : CUSTOMER.C_CUSTKEY
           READ TABLE COLUMN : CUSTOMER.C_NAME
             MIN RANGE : CUSTOMER.C_CUSTKEY = {ORDERS.O_CUSTKEY}
             MAX RANGE : CUSTOMER.C_CUSTKEY = {ORDERS.O_CUSTKEY}
           FETCH ONE ROW
<<<  end print plan
```

NO_USE_NL(table_name [[,] table_name])

If NO_USE_NL(table_name [[,] table_name]) hint is specified, then an optimizer excludes nested loop join when determining the join operation of the join in which table_name participates. Therefore, it selec ts the join operation whose cost estimation is the best among other join operations except for nested loo p join.

The following is an example of using NO_USE_NL(table_name [[,] table_name]) hint.

```
\EXPLAIN PLAN
  SELECT  /*+ NO_USE_NL( orders ) */
          c_name,
          o_orderdate,
          o_orderstatus
    FROM  customer,
          orders
   WHERE  c_custkey = o_custkey
     AND  o_orderdate >= date '1995-03-15'
     AND  o_orderdate < date '1995-03-15' + interval '1' month;
>>>  start print plan
< Execution Plan >
================================================================================
¦ IDX ¦  NODE DESCRIPTION                                          ¦   ROWS ¦
--------------------------------------------------------------------------------
¦  0  ¦  SELECT STATEMENT                                          ¦  19343 ¦
¦  1  ¦    QUERY BLOCK ("$QB_IDX_2")                             ¦  19343 ¦
¦  2  ¦      HASH JOIN (INNER JOIN)                              ¦  19343 ¦
¦  3  ¦        TABLE ACCESS ("CUSTOMER")                         ¦ 150000 ¦
¦  4  ¦        HASH JOIN INSTANT                                 ¦  19343 ¦
¦  5  ¦          TABLE ACCESS ("ORDERS")                         ¦  19343 ¦
================================================================================
     1  -  TARGET : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     2  -  JOINED COLUMN : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     3  -  READ COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_NAME
     4  -  HASH KEY : ORDERS.O_CUSTKEY
           RECORD COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
           READ KEY COLUMN : ORDERS.O_CUSTKEY, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
             HASH FILTER : ORDERS.O_CUSTKEY = CUSTOMER.C_CUSTKEY
     5  -  READ COLUMN : ORDERS.O_CUSTKEY, ORDERS.O_ORDERSTATUS, ORDERS.O_ORDERDATE
             PHYSICAL FILTER : ORDERS.O_ORDERDATE < DATE'1995-03-15' + CAST( '1' AS
INTERVAL(MONTH) ) AND ORDERS.O_ORDERDATE >= DATE'1995-03-15'
<<<  end print plan
```

USE_INL(table_name [[,] table_name])

If USE_INL(table_name [[,] table_name]) hint is specified, then an optimizer selects instant nested loo p join when determining the join operation of the join in which table_name participates.

One or more table should be specified, and the same tables can not be specified more than two. If <join operation hints> already exists in the specified table, then the hint is ignored. Also, if each different join o peration hint is specified in two tables participating in join, then the hint specified in a right node(inner n

ode) is preferentially applied.

The following is an example of using USE_INL(table_name [[,] table_name]) hint.

```
\EXPLAIN PLAN
  SELECT  /*+ USE_INL( orders ) */
          c_name,
          o_orderdate,
          o_orderstatus
    FROM  customer,
          orders
   WHERE  c_custkey = o_custkey
     AND  o_orderdate >= date '1995-03-15'
     AND  o_orderdate < date '1995-03-15' + interval '1' month;
>>>  start print plan
< Execution Plan >
================================================================================
¦ IDX ¦  NODE DESCRIPTION                                          ¦   ROWS ¦
--------------------------------------------------------------------------------
¦  0  ¦  SELECT STATEMENT                                          ¦  19343 ¦
¦  1  ¦    QUERY BLOCK ("$QB_IDX_2")                             ¦  19343 ¦
¦  2  ¦      NESTED JOIN (INNER JOIN)                            ¦  19343 ¦
¦  3  ¦        TABLE ACCESS ("ORDERS")                           ¦  19343 ¦
¦  4  ¦        SORT JOIN INSTANT                                 ¦  19343 ¦
¦  5  ¦          TABLE ACCESS ("CUSTOMER")                       ¦ 150000 ¦
================================================================================
     1  -  TARGET : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     2  -  JOINED COLUMN : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     3  -  READ COLUMN : ORDERS.O_CUSTKEY, ORDERS.O_ORDERSTATUS, ORDERS.O_ORDERDATE
             PHYSICAL FILTER : ORDERS.O_ORDERDATE < DATE'1995-03-15' + CAST( '1' AS
INTERVAL(MONTH) ) AND ORDERS.O_ORDERDATE >= DATE'1995-03-15'
     4  -  SORT KEY : "CUSTOMER.C_CUSTKEY ASC NULLS LAST"
           RECORD COLUMN : CUSTOMER.C_NAME
           READ KEY COLUMN : CUSTOMER.C_CUSTKEY
           READ RECORD COLUMN : CUSTOMER.C_NAME
             MIN RANGE : CUSTOMER.C_CUSTKEY = {ORDERS.O_CUSTKEY}
             MAX RANGE : CUSTOMER.C_CUSTKEY = {ORDERS.O_CUSTKEY}
     5  -  READ COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_NAME
<<<  end print plan
```

USE_INL_IN(alias)

If USE_INL_IN(alias) hint is specified, then an optimizer selects instant nested loop join when determining the join operation of the join in which the alias participates. Then it places the alias on the right(inner) of the join.

A table name, a table alias name, a view name, a view alias name and a join alias name can be an alias.

The following is an example of using USE_INL_IN(alias) hint.

```
\EXPLAIN PLAN
  SELECT  /*+ USE_INL_IN( orders ) */
          c_name,
          o_orderdate,
          o_orderstatus
    FROM  customer,
          orders
   WHERE  c_custkey = o_custkey
     AND  o_orderdate >= date '1995-03-15'
     AND  o_orderdate < date '1995-03-15' + interval '1' month;
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                                          |   ROWS |
--------------------------------------------------------------------------------
|  0  |  SELECT STATEMENT                                            |  19343 |
|  1  |    QUERY BLOCK ("$QB_IDX_2")                                 |  19343 |
|  2  |      NESTED JOIN (INNER JOIN)                                |  19343 |
|  3  |        TABLE ACCESS ("CUSTOMER")                             | 150000 |
|  4  |        SORT JOIN INSTANT                                     |  19343 |
|  5  |          TABLE ACCESS ("ORDERS")                             |  19343 |
================================================================================
     1  -  TARGET : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     2  -  JOINED COLUMN : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     3  -  READ COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_NAME
     4  -  SORT KEY : "ORDERS.O_CUSTKEY ASC NULLS LAST"
           RECORD COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
           READ KEY COLUMN : ORDERS.O_CUSTKEY
           READ RECORD COLUMN : ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
             MIN RANGE : ORDERS.O_CUSTKEY = {CUSTOMER.C_CUSTKEY}
             MAX RANGE : ORDERS.O_CUSTKEY = {CUSTOMER.C_CUSTKEY}
     5  -  READ COLUMN : ORDERS.O_CUSTKEY, ORDERS.O_ORDERSTATUS, ORDERS.O_ORDERDATE
             PHYSICAL FILTER : ORDERS.O_ORDERDATE < DATE'1995-03-15' + CAST( '1' AS
```

```
INTERVAL(MONTH) ) AND ORDERS.O_ORDERDATE >= DATE'1995-03-15'
<<<  end print plan
```

USE_INL_OUT(alias)

If USE_INL_OUT(alias) hint is specified, then an optimizer selects instant nested loop join when determin ing the join operation of the join in which the alias participates. Then it places the alias on the left(outer) of the join.

A table name, a table alias name, a view name, a view alias name and a join alias name can be an alias.

The following is an example of using USE_INL_OUT(alias) hint.

```
\EXPLAIN PLAN
  SELECT  /*+ USE_INL_OUT( orders ) */
          c_name,
          o_orderdate,
          o_orderstatus
    FROM  customer,
          orders
   WHERE  c_custkey = o_custkey
     AND  o_orderdate >= date '1995-03-15'
     AND  o_orderdate < date '1995-03-15' + interval '1' month;
>>>  start print plan
< Execution Plan >
================================================================================
| IDX |  NODE DESCRIPTION                                          |   ROWS |
--------------------------------------------------------------------------------
|  0  |  SELECT STATEMENT                                          |  19343 |
|  1  |    QUERY BLOCK ("$QB_IDX_2")                               |  19343 |
|  2  |      NESTED JOIN (INNER JOIN)                              |  19343 |
|  3  |        TABLE ACCESS ("ORDERS")                             |  19343 |
|  4  |        SORT JOIN INSTANT                                   |  19343 |
|  5  |          TABLE ACCESS ("CUSTOMER")                         | 150000 |
================================================================================
     1  -  TARGET : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     2  -  JOINED COLUMN : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     3  -  READ COLUMN : ORDERS.O_CUSTKEY, ORDERS.O_ORDERSTATUS, ORDERS.O_ORDERDATE
             PHYSICAL FILTER : ORDERS.O_ORDERDATE < DATE'1995-03-15' + CAST( '1' AS
INTERVAL(MONTH) ) AND ORDERS.O_ORDERDATE >= DATE'1995-03-15'
     4  -  SORT KEY : "CUSTOMER.C_CUSTKEY ASC NULLS LAST"
           RECORD COLUMN : CUSTOMER.C_NAME
           READ KEY COLUMN : CUSTOMER.C_CUSTKEY
```

```
           READ RECORD COLUMN : CUSTOMER.C_NAME
             MIN RANGE : CUSTOMER.C_CUSTKEY = {ORDERS.O_CUSTKEY}
             MAX RANGE : CUSTOMER.C_CUSTKEY = {ORDERS.O_CUSTKEY}
     5  -  READ COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_NAME
<<<  end print plan
```

NO_USE_INL(table_name [[,] table_name])

If NO_USE_INL(table_name [[,] table_name]) hint is specified, then an optimizer excludes instant nested loop join when determining the join operation of the join in which table_name participates. Therefore, it selects the join operation whose cost estimation is the best among other join operations except for instant nested loop join.

One or more table should be specified, and the same tables can not be specified more than two. If <join operation hints> already exists in the specified table, then the hint is ignored. Also, if each different join operation hint is specified in two tables participating in join, then the hint specified in a right node(inner node) is preferentially applied.

The following is an example of using NO_USE_INL(table_name [[,] table_name]) hint.

```
\EXPLAIN PLAN
  SELECT  /*+ NO_USE_INL( orders ) */
          c_name,
          o_orderdate,
          o_orderstatus
    FROM  customer,
          orders
   WHERE  c_custkey = o_custkey
     AND  o_orderdate >= date '1995-03-15'
     AND  o_orderdate < date '1995-03-15' + interval '1' month;
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                                          | ROWS |
-----------------------------------------------------------------------------
|  0  |  SELECT STATEMENT                                           | 19343 |
|  1  |    QUERY BLOCK ("$QB_IDX_2")                                | 19343 |
|  2  |      NESTED JOIN (INNER JOIN)                               | 19343 |
|  3  |        TABLE ACCESS ("ORDERS")                              | 19343 |
|  4  |        INDEX ACCESS ("CUSTOMER", "CUSTOMER_PK_INDEX")       | 19343 |
=============================================================================
     1  -  TARGET : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     2  -  JOINED COLUMN : CUSTOMER.C_NAME, ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     3  -  READ COLUMN : ORDERS.O_CUSTKEY, ORDERS.O_ORDERSTATUS, ORDERS.O_ORDERDATE
```

```
            PHYSICAL FILTER : ORDERS.O_ORDERDATE < DATE'1995-03-15' + CAST( '1' AS
INTERVAL(MONTH) ) AND ORDERS.O_ORDERDATE >= DATE'1995-03-15'
     4  -  READ INDEX COLUMN : CUSTOMER.C_CUSTKEY
           READ TABLE COLUMN : CUSTOMER.C_NAME
             MIN RANGE : CUSTOMER.C_CUSTKEY = {ORDERS.O_CUSTKEY}
             MAX RANGE : CUSTOMER.C_CUSTKEY = {ORDERS.O_CUSTKEY}
           FETCH ONE ROW
<<<  end print plan
```

USE_JOIN_COMBINE(alias)

If USE_JOIN_COMBINE(alias) hint is specified, then an optimizer selects join combine when determining the join operation of the join in which the alias participates.

A table name, a table alias name, a view name, a view alias name and a join alias name can be an alias.

The following is an example of using USE_JOIN_COMBINE(alias) hint.

```
\EXPLAIN PLAN
SELECT /*+ USE_JOIN_COMBINE(part) */
        sum(l_extendedprice * (1 - l_discount) ) as revenue
  FROM lineitem,
       part
 WHERE
       (
            p_partkey = l_partkey
        and p_brand = 'Brand#12'
        and p_container in ( 'SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
        and l_quantity >= 1 and l_quantity <= 1 + 10
        and p_size between 1 and 5
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON'
       )
       or
       (
            p_partkey = l_partkey
        and p_brand = 'Brand#23'
        and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
        and l_quantity >= 10 and l_quantity <= 10 + 10
        and p_size between 1 and 10
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON'
       )
```

```
      or
      (
            p_partkey = l_partkey
        and p_brand = 'Brand#34'
        and p_container in ( 'LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
        and l_quantity >= 20 and l_quantity <= 20 + 10
        and p_size between 1 and 15
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON'
      );
>>>  start print plan
< Execution Plan >
============================================================================
|IDX|  NODE DESCRIPTION                                                      |
----------------------------------------------------------------------------
| 0 |  SELECT STATEMENT                                                      |
| 1 |    QUERY BLOCK ("$QB_IDX_2")                                           |
| 2 |      AGGREGATION BY HASH                                               |
| 3 |        CONCAT (Compare Nothing)                                        |
| 4 |          NESTED JOIN (INNER JOIN)                                      |
| 5 |            TABLE ACCESS ("PART")                                       |
| 6 |            INDEX ACCESS ("LINEITEM", "LINEITEM_PARTKEY_SUPPKEY_FK")    |
| 7 |          NESTED JOIN (INNER JOIN)                                      |
| 8 |            TABLE ACCESS ("PART")                                       |
| 9 |            INDEX ACCESS ("LINEITEM", "LINEITEM_PARTKEY_SUPPKEY_FK")    |
|10 |          NESTED JOIN (INNER JOIN)                                      |
|11 |            TABLE ACCESS ("PART")                                       |
|12 |            INDEX ACCESS ("LINEITEM", "LINEITEM_PARTKEY_SUPPKEY_FK")    |
============================================================================
     1  -  TARGET : SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 - LINEITEM.L_DISCOUNT ) ) AS REVENUE
     2  -  AGGREGATION : SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 - LINEITEM.L_DISCOUNT ) )
     3  -  CONCAT COLUMN : LINEITEM.L_EXTENDEDPRICE, LINEITEM.L_DISCOUNT
     4  -  JOINED COLUMN : LINEITEM.L_EXTENDEDPRICE, LINEITEM.L_DISCOUNT
     5  -  READ COLUMN : PART.P_PARTKEY, PART.P_BRAND, PART.P_SIZE, PART.P_CONTAINER
             PHYSICAL FILTER : PART.P_BRAND = 'Brand#12' AND PART.P_SIZE <= 5 AND PART.P_SIZE
>= 1 AND ( PART.P_CONTAINER ) IN ( 'SM CASE', 'SM BOX', 'SM PACK', 'SM PKG' )
     6  -  READ INDEX COLUMN : LINEITEM.L_PARTKEY
           READ TABLE COLUMN : LINEITEM.L_QUANTITY, LINEITEM.L_EXTENDEDPRICE,
LINEITEM.L_DISCOUNT, LINEITEM.L_SHIPINSTRUCT, LINEITEM.L_SHIPMODE
             MIN RANGE : LINEITEM.L_PARTKEY = {PART.P_PARTKEY}
             MAX RANGE : LINEITEM.L_PARTKEY = {PART.P_PARTKEY}
```

```
            PHYSICAL TABLE FILTER : LINEITEM.L_QUANTITY <= 1 + 10 AND LINEITEM.L_QUANTITY >=
1 AND LINEITEM.L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND ( LINEITEM.L_SHIPMODE ) IN ( 'AIR',
'AIR REG' )
     7  -  JOINED COLUMN : LINEITEM.L_EXTENDEDPRICE, LINEITEM.L_DISCOUNT
     8  -  READ COLUMN : PART.P_PARTKEY, PART.P_BRAND, PART.P_SIZE, PART.P_CONTAINER
            PHYSICAL FILTER : PART.P_BRAND = 'Brand#23' AND PART.P_SIZE <= 10 AND PART.P_SIZE
>= 1 AND ( PART.P_CONTAINER ) IN ( 'MED BAG', 'MED BOX', 'MED PKG', 'MED PACK' )
     9  -  READ INDEX COLUMN : LINEITEM.L_PARTKEY
           READ TABLE COLUMN : LINEITEM.L_QUANTITY, LINEITEM.L_EXTENDEDPRICE,
LINEITEM.L_DISCOUNT, LINEITEM.L_SHIPINSTRUCT, LINEITEM.L_SHIPMODE
            MIN RANGE : LINEITEM.L_PARTKEY = {PART.P_PARTKEY}
            MAX RANGE : LINEITEM.L_PARTKEY = {PART.P_PARTKEY}
            PHYSICAL TABLE FILTER : LINEITEM.L_QUANTITY <= 10 + 10 AND LINEITEM.L_QUANTITY >=
10 AND LINEITEM.L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND ( LINEITEM.L_SHIPMODE ) IN ( 'AIR',
'AIR REG' )
    10  -  JOINED COLUMN : LINEITEM.L_EXTENDEDPRICE, LINEITEM.L_DISCOUNT
    11  -  READ COLUMN : PART.P_PARTKEY, PART.P_BRAND, PART.P_SIZE, PART.P_CONTAINER
            PHYSICAL FILTER : PART.P_BRAND = 'Brand#34' AND PART.P_SIZE <= 15 AND PART.P_SIZE
>= 1 AND ( PART.P_CONTAINER ) IN ( 'LG CASE', 'LG BOX', 'LG PACK', 'LG PKG' )
    12  -  READ INDEX COLUMN : LINEITEM.L_PARTKEY
           READ TABLE COLUMN : LINEITEM.L_QUANTITY, LINEITEM.L_EXTENDEDPRICE,
LINEITEM.L_DISCOUNT, LINEITEM.L_SHIPINSTRUCT, LINEITEM.L_SHIPMODE
            MIN RANGE : LINEITEM.L_PARTKEY = {PART.P_PARTKEY}
            MAX RANGE : LINEITEM.L_PARTKEY = {PART.P_PARTKEY}
            PHYSICAL TABLE FILTER : LINEITEM.L_QUANTITY <= 20 + 10 AND LINEITEM.L_QUANTITY >=
20 AND LINEITEM.L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND ( LINEITEM.L_SHIPMODE ) IN ( 'AIR',
'AIR REG' )
<<<  end print plan
```

NO_USE_JOIN_COMBINE(alias)

If NO_USE_JOIN_COMBINE(alias) hint is specified, then an optimizer excludes join combine when deter mining the join operation of the join in which the alias participates.

A table name, a table alias name, a view name, a view alias name and a join alias name can be an alias.

The following is an example of using NO_USE_JOIN_COMBINE(alias) hint.

```
\EXPLAIN PLAN
SELECT /*+ NO_USE_JOIN_COMBINE(part) */
        sum(l_extendedprice * (1 - l_discount) ) as revenue
  FROM lineitem,
       part
```

```
 WHERE
       (
            p_partkey = l_partkey
        and p_brand = 'Brand#12'
        and p_container in ( 'SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
        and l_quantity >= 1 and l_quantity <= 1 + 10
        and p_size between 1 and 5
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON'
       )
       or
       (
            p_partkey = l_partkey
        and p_brand = 'Brand#23'
        and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
        and l_quantity >= 10 and l_quantity <= 10 + 10
        and p_size between 1 and 10
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON'
       )
      or
      (
            p_partkey = l_partkey
        and p_brand = 'Brand#34'
        and p_container in ( 'LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
        and l_quantity >= 20 and l_quantity <= 20 + 10
        and p_size between 1 and 15
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON'
      );
>>>  start print plan
< Execution Plan >
================================================================================
¦IDX¦  NODE DESCRIPTION                                                        ¦
--------------------------------------------------------------------------------
¦ 0 ¦  SELECT STATEMENT                                                        ¦
¦ 1 ¦    QUERY BLOCK ("$QB_IDX_2")                                             ¦
¦ 2 ¦      AGGREGATION BY HASH                                                 ¦
¦ 3 ¦        NESTED JOIN (INNER JOIN)                                          ¦
¦ 4 ¦          TABLE ACCESS ("PART")                                           ¦
¦ 5 ¦          CONCAT (Compare Nothing)                                        ¦
```

```
¦ 6 ¦            INDEX ACCESS ("LINEITEM", "LINEITEM_PARTKEY_SUPPKEY_FK") ¦
¦ 7 ¦            INDEX ACCESS ("LINEITEM", "LINEITEM_PARTKEY_SUPPKEY_FK") ¦
¦ 8 ¦            INDEX ACCESS ("LINEITEM", "LINEITEM_PARTKEY_SUPPKEY_FK") ¦
=========================================================================
     1  -  TARGET : SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 - LINEITEM.L_DISCOUNT ) ) AS REVENUE
     2  -  AGGREGATION : SUM( LINEITEM.L_EXTENDEDPRICE * ( 1 - LINEITEM.L_DISCOUNT ) )
     3  -  JOINED COLUMN : LINEITEM.L_EXTENDEDPRICE, LINEITEM.L_DISCOUNT
     4  -  READ COLUMN : PART.P_PARTKEY, PART.P_BRAND, PART.P_SIZE, PART.P_CONTAINER
     5  -  CONCAT COLUMN : LINEITEM.L_EXTENDEDPRICE, LINEITEM.L_DISCOUNT
     6  -  READ INDEX COLUMN : LINEITEM.L_PARTKEY
           READ TABLE COLUMN : LINEITEM.L_QUANTITY, LINEITEM.L_EXTENDEDPRICE,
LINEITEM.L_DISCOUNT, LINEITEM.L_SHIPINSTRUCT, LINEITEM.L_SHIPMODE
             MIN RANGE : LINEITEM.L_PARTKEY = {PART.P_PARTKEY}
             MAX RANGE : LINEITEM.L_PARTKEY = {PART.P_PARTKEY}
             PHYSICAL TABLE FILTER : LINEITEM.L_QUANTITY <= 1 + 10 AND LINEITEM.L_QUANTITY >=
1 AND LINEITEM.L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND ( LINEITEM.L_SHIPMODE ) IN ( 'AIR',
'AIR REG' )
             CONSTANT FILTER : {PART.P_BRAND} = 'Brand#12' AND ( {PART.P_CONTAINER} ) IN ( 'SM
CASE', 'SM BOX', 'SM PACK', 'SM PKG' ) AND {PART.P_SIZE} <= 5 AND {PART.P_SIZE} >= 1
     7  -  READ INDEX COLUMN : LINEITEM.L_PARTKEY
           READ TABLE COLUMN : LINEITEM.L_QUANTITY, LINEITEM.L_EXTENDEDPRICE,
LINEITEM.L_DISCOUNT, LINEITEM.L_SHIPINSTRUCT, LINEITEM.L_SHIPMODE
             MIN RANGE : LINEITEM.L_PARTKEY = {PART.P_PARTKEY}
             MAX RANGE : LINEITEM.L_PARTKEY = {PART.P_PARTKEY}
             PHYSICAL TABLE FILTER : LINEITEM.L_QUANTITY <= 10 + 10 AND LINEITEM.L_QUANTITY >=
10 AND LINEITEM.L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND ( LINEITEM.L_SHIPMODE ) IN ( 'AIR',
'AIR REG' )
             CONSTANT FILTER : {PART.P_BRAND} = 'Brand#23' AND ( {PART.P_CONTAINER} ) IN (
'MED BAG', 'MED BOX', 'MED PKG', 'MED PACK' ) AND {PART.P_SIZE} <= 10 AND {PART.P_SIZE} >= 1
     8  -  READ INDEX COLUMN : LINEITEM.L_PARTKEY
           READ TABLE COLUMN : LINEITEM.L_QUANTITY, LINEITEM.L_EXTENDEDPRICE,
LINEITEM.L_DISCOUNT, LINEITEM.L_SHIPINSTRUCT, LINEITEM.L_SHIPMODE
             MIN RANGE : LINEITEM.L_PARTKEY = {PART.P_PARTKEY}
             MAX RANGE : LINEITEM.L_PARTKEY = {PART.P_PARTKEY}
             PHYSICAL TABLE FILTER : LINEITEM.L_QUANTITY <= 20 + 10 AND LINEITEM.L_QUANTITY >=
20 AND LINEITEM.L_SHIPINSTRUCT = 'DELIVER IN PERSON' AND ( LINEITEM.L_SHIPMODE ) IN ( 'AIR',
'AIR REG' )
             CONSTANT FILTER : {PART.P_BRAND} = 'Brand#34' AND ( {PART.P_CONTAINER} ) IN ( 'LG
CASE', 'LG BOX', 'LG PACK', 'LG PKG' ) AND {PART.P_SIZE} <= 15 AND {PART.P_SIZE} >= 1
<<<  end print plan
```

〈join driver hints〉

Those hints are available when performing the join in the cluster system. They are ignored in the standalone system.

LOCAL_JOIN(alias)

It performs the join in the current server when an alias participates in the join. If both a left child and a right child are sharded tables, it brings all subordinate rows to a local, then performs the join.

The following is an example of using LOCAL_JOIN hint.

```
\EXPLAIN PLAN
SELECT /*+ LOCAL_JOIN(lineitem) */
       l_orderkey, p_partkey
  FROM part, lineitem
 WHERE p_partkey = l_partkey;
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                          |                ROWS |
--------------------------------------------------------------------------------
| 0 |  SELECT STATEMENT                              |                2528 |
| 1 |    QUERY BLOCK ("$QB_IDX_2")                   |                2528 |
| 2 |      HASH JOIN (INNER JOIN)                    |                2528 |
| 3 |        PLAN BASED CLUSTER                      | LOCAL/REMOTE 200000 |
| 4 |          INDEX ACCESS ("PART", "PART_PK_INDEX")|       (66675) 66675 |
| 5 |        HASH JOIN INSTANT                       |                2528 |
| 6 |          PLAN BASED CLUSTER                    |   LOCAL/REMOTE 2528 |
| 7 |            TABLE ACCESS ("LINEITEM")           |                 840 |
================================================================================
     1  -  TARGET : LINEITEM.L_ORDERKEY, PART.P_PARTKEY
     2  -  JOINED COLUMN : LINEITEM.L_ORDERKEY, PART.P_PARTKEY
     3  -  SQL : SELECT /*+ INDEX( _A1, "PUBLIC"."PART_PK_INDEX" ) */
                       "_A1"."P_PARTKEY"
                  FROM "PUBLIC"."PART"@LOCAL AS "_A1"
           TARGET DOMAIN : G1(G1N1,G1N2) 66675 rows,
                           G2(G2N1,G2N2) 66664 rows,
                           G3(G3N1,G3N2) 66661 rows
     4  -  HASH SHARD ( # 3 )
           READ INDEX COLUMN : PART.P_PARTKEY
     5  -  HASH KEY : LINEITEM.L_PARTKEY
           RECORD COLUMN : LINEITEM.L_ORDERKEY
```

```
          READ KEY COLUMN : LINEITEM.L_PARTKEY, LINEITEM.L_ORDERKEY
            HASH FILTER : LINEITEM.L_PARTKEY = PART.P_PARTKEY
    6  -  SQL : SELECT /*+ FULL( _A1 ) */
                       "_A1"."L_ORDERKEY", "_A1"."L_PARTKEY"
                  FROM "PUBLIC"."LINEITEM"@LOCAL AS "_A1"
                 WHERE "_A1"."L_SHIPDATE" = :_V0
          TARGET DOMAIN : G1(G1N1,G1N2) 840 rows,
                          G2(G2N1,G2N2) 824 rows,
                          G3(G3N1,G3N2) 864 rows
    7  -  HASH SHARD ( # 3 )
          READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_PARTKEY, LINEITEM.L_SHIPDATE
            PHYSICAL FILTER : LINEITEM.L_SHIPDATE = DATE'1995-03-15'
<<<  end print plan
```

Both part and lineitem are hash sharded tables. In the execution plan above, it brings both tables to G1, G2 and G3, then performs the join in a local.

REMOTE_JOIN(alias)

It performs the join in each server when an alias participates in the join.

The following is an example of using REMOTE_JOIN hint.

```
\EXPLAIN PLAN
SELECT /*+ REMOTE_JOIN(lineitem) */
       l_orderkey, p_partkey
  FROM part, lineitem
 WHERE p_partkey = l_partkey
   AND l_shipdate = date '1995-03-15';
>>>  start print plan
< Execution Plan >
================================================================================
| IDX |NODE DESCRIPTION                                        |      ROWS |
--------------------------------------------------------------------------------
|  0 |SELECT STATEMENT                                         |      2528 |
|  1 |  QUERY BLOCK ("$QB_IDX_2")                              |      2528 |
|  2 |    SINGLE CLUSTER                                 | LOCAL/REMOTE 2528 |
|  3 |      CLUSTER PUSHER ("_$NI_5")                          |      2528 |
|  4 |        PLAN BASED CLUSTER                         | LOCAL/REMOTE 2528 |
|  5 |          TABLE ACCESS ("LINEITEM")                      |       840 |
|  6 |      SELECT STATEMENT                                   |       857 |
|  7 |        QUERY BLOCK ("$QB_IDX_2")                        |       857 |
|  8 |          NESTED JOIN (INNER JOIN)                       |       857 |
```

```
┆  9 ┆            PUSHER TABLE ACCESS ("_$NI_5" AS _A2)        ┆       857 ┆
┆ 10 ┆            INDEX ACCESS ("PART" AS _A1, "PART_PK_INDEX")┆ (857) 857 ┆
================================================================================
     1  -  TARGET : _$NI_5.L_ORDERKEY, PART.P_PARTKEY
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE
                            USE_NL_IN( _A1 )
                            FULL( _A2 )
                            INDEX( _A1, "PUBLIC"."PART_PK_INDEX" )
                        */
                        "_A2"."L_ORDERKEY", "_A1"."P_PARTKEY"
                    FROM ( "SESSION_SCHEMA"."_$NI_5"@LOCAL AS "_A2"
                           INNER JOIN
                           "PUBLIC"."PART"@LOCAL AS "_A1"
                           ON true
                        ) ALIAS "_A3"
                   WHERE "_A1"."P_PARTKEY" = "_A2"."L_PARTKEY"
           TARGET DOMAIN : G1(G1N1,G1N2) 857 rows,
                           G2(G2N1,G2N2) 845 rows,
                           G3(G3N1,G3N2) 826 rows
     3  -  SQL : DECLARE INSTANT TABLE "SESSION_SCHEMA"."_$NI_5"
                    ( "L_PARTKEY" NUMBER(10, 0), "L_ORDERKEY" NUMBER(10, 0) )
           COLUMN : LINEITEM.L_PARTKEY AS L_PARTKEY, LINEITEM.L_ORDERKEY AS L_ORDERKEY
           SHARDED : LINEITEM.L_PARTKEY
           TARGET DOMAIN : G1(G1N1,G1N2) 857 rows,
                           G2(G2N1,G2N2) 845 rows,
                           G3(G3N1,G3N2) 826 rows
     4  -  SQL : SELECT /*+ FULL( _A1 ) */
                        "_A1"."L_ORDERKEY", "_A1"."L_PARTKEY"
                    FROM "PUBLIC"."LINEITEM"@LOCAL AS "_A1"
                   WHERE "_A1"."L_SHIPDATE" = :_V0
           TARGET DOMAIN : G1(G1N1,G1N2) 840 rows,
                           G2(G2N1,G2N2) 824 rows,
                           G3(G3N1,G3N2) 864 rows
     5  -  HASH SHARD ( # 3 )
           READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_PARTKEY, LINEITEM.L_SHIPDATE
             PHYSICAL FILTER : LINEITEM.L_SHIPDATE = DATE'1995-03-15'
     7  -  TARGET : _A2.L_ORDERKEY, _A1.P_PARTKEY
     8  -  JOINED COLUMN : _A2.L_ORDERKEY, _A1.P_PARTKEY
             CONSTANT FILTER : TRUE
     9  -  READ COLUMN : _A2.L_PARTKEY, _A2.L_ORDERKEY
    10  -  HASH SHARD ( # 3 )
```

```
           READ INDEX COLUMN : _A1.P_PARTKEY
             MIN RANGE : _A1.P_PARTKEY = {_A2.L_PARTKEY}
             MAX RANGE : _A1.P_PARTKEY = {_A2.L_PARTKEY}
           FETCH ONE ROW
<<<  end print plan
```

In the execution plan above, it performs the remote join by creating lineitem as a pusher table whose sha rd key is l_partkey, then transferring SQL to each server.

〈join pusher hints〉

Those hints are used when performing the remote join in a cluster system. It is ignored in a standalone sy stem.

PUSHER(alias)

It builds a table or a view corresponding to an alias as a pusher table when an alias participates in a remot e join.
The hint is not applied when the remote join is available without a pusher table.

The following is an example of using PUSHER hint.

```
\EXPLAIN PLAN
SELECT /*+ PUSHER(lineitem) */
       l_orderkey, p_partkey
  FROM part, lineitem
 WHERE p_partkey = l_partkey
   AND l_shipdate = date '1995-03-15';
>>>  start print plan
< Execution Plan >
================================================================================
¦  IDX  ¦  NODE DESCRIPTION                                          ¦ ROWS ¦
--------------------------------------------------------------------------------
¦  0 ¦  SELECT STATEMENT                                             ¦ 2528 ¦
¦  1 ¦    QUERY BLOCK ("$QB_IDX_2")                                 ¦ 2528 ¦
¦  2 ¦      SINGLE CLUSTER                                ¦ LOCAL/REMOTE 2528 ¦
¦  3 ¦        CLUSTER PUSHER ("_$NI_5")                             ¦ 2528 ¦
¦  4 ¦          PLAN BASED CLUSTER                        ¦ LOCAL/REMOTE 2528 ¦
¦  5 ¦            TABLE ACCESS ("LINEITEM")                          ¦ 840 ¦
¦  6 ¦        SELECT STATEMENT                                       ¦ 857 ¦
¦  7 ¦          QUERY BLOCK ("$QB_IDX_2")                            ¦ 857 ¦
¦  8 ¦            NESTED JOIN (INNER JOIN)                           ¦ 857 ¦
¦  9 ¦              PUSHER TABLE ACCESS ("_$NI_5" AS _A2)            ¦ 857 ¦
```

```
| 10 |             INDEX ACCESS ("PART" AS _A1, "PART_PK_INDEX")    | 857 |
=============================================================================
     1  -  TARGET : _$NI_5.L_ORDERKEY, PART.P_PARTKEY
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE
                            USE_NL_IN( _A1 )
                            FULL( _A2 )
                            INDEX( _A1, "PUBLIC"."PART_PK_INDEX" )
                        */
                        "_A2"."L_ORDERKEY", "_A1"."P_PARTKEY"
                  FROM ( "SESSION_SCHEMA"."_$NI_5"@LOCAL AS "_A2"
                         INNER JOIN
                         "PUBLIC"."PART"@LOCAL AS "_A1"
                         ON true
                       ) ALIAS "_A3"
                WHERE "_A1"."P_PARTKEY" = "_A2"."L_PARTKEY"
           TARGET DOMAIN : G1(G1N1,G1N2) 857 rows,
                           G2(G2N1,G2N2) 845 rows,
                           G3(G3N1,G3N2) 826 rows
     3  -  SQL : DECLARE INSTANT TABLE "SESSION_SCHEMA"."_$NI_5"
                ( "L_PARTKEY" NUMBER(10, 0), "L_ORDERKEY" NUMBER(10, 0) )
           COLUMN : LINEITEM.L_PARTKEY AS L_PARTKEY, LINEITEM.L_ORDERKEY AS L_ORDERKEY
           SHARDED : LINEITEM.L_PARTKEY
           TARGET DOMAIN : G1(G1N1,G1N2) 857 rows,
                           G2(G2N1,G2N2) 845 rows,
                           G3(G3N1,G3N2) 826 rows
     4  -  SQL : SELECT /*+ FULL( _A1 ) */
                        "_A1"."L_ORDERKEY", "_A1"."L_PARTKEY"
                  FROM "PUBLIC"."LINEITEM"@LOCAL AS "_A1"
                WHERE "_A1"."L_SHIPDATE" = :_V0
           TARGET DOMAIN : G1(G1N1,G1N2) 840 rows,
                           G2(G2N1,G2N2) 824 rows,
                           G3(G3N1,G3N2) 864 rows
     5  -  HASH SHARD ( # 3 )
           READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_PARTKEY, LINEITEM.L_SHIPDATE
             PHYSICAL FILTER : LINEITEM.L_SHIPDATE = DATE'1995-03-15'
     7  -  TARGET : _A2.L_ORDERKEY, _A1.P_PARTKEY
     8  -  JOINED COLUMN : _A2.L_ORDERKEY, _A1.P_PARTKEY
             CONSTANT FILTER : TRUE
     9  -  READ COLUMN : _A2.L_PARTKEY, _A2.L_ORDERKEY
    10  -  HASH SHARD ( # 3 )
           READ INDEX COLUMN : _A1.P_PARTKEY
```

```
            MIN RANGE : _A1.P_PARTKEY = {_A2.L_PARTKEY}
            MAX RANGE : _A1.P_PARTKEY = {_A2.L_PARTKEY}
          FETCH ONE ROW
<<<  end print plan
```

In the execution plan above, it performs the remote join by creating lineitem as a pusher table whose sha rd key is l_partkey, then transferring SQL to each server.

NO_PUSHER(alias)

It does not build a table or a view corresponding to an alias as a pusher table when an alias participates in the join.

If the remote join cost of when building a sibling of an alias as a pusher table is big, then a local join can be selected.
In this case, if using it together with REMOTE_JOIN hint, then it performs the remote join by building a si bling of an alias as a pusher table.

The following is an example of using NO_PUSHER hint.

```
\EXPLAIN PLAN
SELECT /*+ NO_PUSHER(lineitem) */
       l_orderkey, p_partkey
  FROM part, lineitem
 WHERE p_partkey = l_partkey
   AND l_shipdate = date '1995-03-15';
>>>  start print plan
< Execution Plan >
================================================================================
¦IDX¦  NODE DESCRIPTION                              ¦                ROWS ¦
--------------------------------------------------------------------------------
¦ 0 ¦ SELECT STATEMENT                               ¦                2528 ¦
¦ 1 ¦   QUERY BLOCK ("$QB_IDX_2")                    ¦                2528 ¦
¦ 2 ¦     HASH JOIN (INNER JOIN)                     ¦                2528 ¦
¦ 3 ¦       PLAN BASED CLUSTER                       ¦ LOCAL/REMOTE 200000 ¦
¦ 4 ¦         INDEX ACCESS ("PART", "PART_PK_INDEX") ¦     (66675)   66675 ¦
¦ 5 ¦       HASH JOIN INSTANT                        ¦                2528 ¦
¦ 6 ¦         PLAN BASED CLUSTER                     ¦ LOCAL/REMOTE   2528 ¦
¦ 7 ¦           TABLE ACCESS ("LINEITEM")            ¦                 840 ¦
================================================================================
     1  -  TARGET : LINEITEM.L_ORDERKEY, PART.P_PARTKEY
     2  -  JOINED COLUMN : LINEITEM.L_ORDERKEY, PART.P_PARTKEY
     3  -  SQL : SELECT /*+ INDEX( _A1, "PUBLIC"."PART_PK_INDEX" ) */
```

```
                    "_A1"."P_PARTKEY"
                FROM "PUBLIC"."PART"@LOCAL AS "_A1"
        TARGET DOMAIN : G1(G1N1,G1N2) 66675 rows,
                        G2(G2N1,G2N2) 66664 rows,
                        G3(G3N1,G3N2) 66661 rows
  4  -  HASH SHARD ( # 3 )
        READ INDEX COLUMN : PART.P_PARTKEY
  5  -  HASH KEY : LINEITEM.L_PARTKEY
        RECORD COLUMN : LINEITEM.L_ORDERKEY
        READ KEY COLUMN : LINEITEM.L_PARTKEY, LINEITEM.L_ORDERKEY
          HASH FILTER : LINEITEM.L_PARTKEY = PART.P_PARTKEY
  6  -  SQL : SELECT /*+ FULL( _A1 ) */
                    "_A1"."L_ORDERKEY", "_A1"."L_PARTKEY"
                FROM "PUBLIC"."LINEITEM"@LOCAL AS "_A1"
               WHERE "_A1"."L_SHIPDATE" = :_V0
        TARGET DOMAIN : G1(G1N1,G1N2) 840 rows,
                        G2(G2N1,G2N2) 824 rows,
                        G3(G3N1,G3N2) 864 rows
  7  -  HASH SHARD ( # 3 )
        READ COLUMN : LINEITEM.L_ORDERKEY, LINEITEM.L_PARTKEY, LINEITEM.L_SHIPDATE
          PHYSICAL FILTER : LINEITEM.L_SHIPDATE = DATE'1995-03-15'
<<<  end print plan
```

In the execution plan above, the local join is performed. If the remote join cost of when building a sibling of an alias as a pusher table is big, then a local join can be selected as follows.

```
\EXPLAIN PLAN
SELECT /*+ REMOTE_JOIN(lineitem) NO_PUSHER(lineitem) */
       l_orderkey, p_partkey
  FROM part, lineitem
 WHERE p_partkey = l_partkey
   AND l_shipdate = date '1995-03-15';
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                          |               ROWS |
--------------------------------------------------------------------------------
|  0 |  SELECT STATEMENT                             |               2528 |
|  1 |    QUERY BLOCK ("$QB_IDX_2")                  |               2528 |
|  2 |      SINGLE CLUSTER                           |LOCAL/REMOTE   2528 |
|  3 |        CLUSTER PUSHER ("_$NI_5")              |             200000 |
```

```
┆  4 ┆          PLAN BASED CLUSTER                    ┆LOCAL/REMOTE 200000 ┆
┆  5 ┆            INDEX ACCESS ("PART", "PART_PK_INDEX")    ┆(66675) 66675 ┆
┆  6 ┆        SELECT STATEMENT                                ┆    840 ┆
┆  7 ┆          QUERY BLOCK ("$QB_IDX_2")                     ┆    840 ┆
┆  8 ┆            HASH JOIN (INNER JOIN)                      ┆    840 ┆
┆  9 ┆              PUSHER TABLE ACCESS ("_$NI_5" AS _A2)     ┆ 200000 ┆
┆ 10 ┆              HASH JOIN INSTANT                         ┆    840 ┆
┆ 11 ┆                TABLE ACCESS ("LINEITEM" AS _A1)         ┆    840 ┆
================================================================================
     1  -  TARGET : LINEITEM.L_ORDERKEY, _$NI_5.P_PARTKEY
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE
                            USE_HASH_IN( _A1, 396 )
                            FULL( _A2 )
                            FULL( _A1 )
                        */
                        "_A1"."L_ORDERKEY", "_A2"."P_PARTKEY"
                   FROM ( "SESSION_SCHEMA"."_$NI_5"@LOCAL AS "_A2"
                          INNER JOIN
                          "PUBLIC"."LINEITEM"@LOCAL AS "_A1"
                          ON "_A1"."L_PARTKEY" = "_A2"."P_PARTKEY"
                        ) ALIAS "_A3"
                  WHERE "_A1"."L_SHIPDATE" = :_V0
             TARGET DOMAIN : G1(G1N1,G1N2) 840 rows,
                             G2(G2N1,G2N2) 824 rows,
                             G3(G3N1,G3N2) 864 rows
     3  -  SQL : DECLARE INSTANT TABLE "SESSION_SCHEMA"."_$NI_5"
                  ( "P_PARTKEY" NUMBER(10, 0) )
             COLUMN : PART.P_PARTKEY AS P_PARTKEY
             CLONED
             TARGET DOMAIN : G1(G1N1,G1N2) 200000 rows,
                             G2(G2N1,G2N2) 200000 rows,
                             G3(G3N1,G3N2) 200000 rows
     4  -  SQL : SELECT /*+ INDEX( _A1, "PUBLIC"."PART_PK_INDEX" ) */
                         "_A1"."P_PARTKEY"
                    FROM "PUBLIC"."PART"@LOCAL AS "_A1"
             TARGET DOMAIN : G1(G1N1,G1N2) 66675 rows,
                             G2(G2N1,G2N2) 66664 rows,
                             G3(G3N1,G3N2) 66661 rows
     5  -  HASH SHARD ( # 3 )
             READ INDEX COLUMN : PART.P_PARTKEY
     7  -  TARGET : _A1.L_ORDERKEY, _A2.P_PARTKEY
```

```
     8  -  JOINED COLUMN : _A1.L_ORDERKEY, _A2.P_PARTKEY
     9  -  READ COLUMN : _A2.P_PARTKEY
    10  -  HASH KEY : _A1.L_PARTKEY
           RECORD COLUMN : _A1.L_ORDERKEY
           READ KEY COLUMN : _A1.L_PARTKEY, _A1.L_ORDERKEY
             HASH FILTER : _A1.L_PARTKEY = _A2.P_PARTKEY
    11  -  HASH SHARD ( # 3 )
           READ COLUMN : _A1.L_ORDERKEY, _A1.L_PARTKEY, _A1.L_SHIPDATE
             PHYSICAL FILTER : _A1.L_SHIPDATE = :_V0
<<<  end print plan
```

In the execution plan above, it performs the remote join by building a sibling of an alias as a pusher table.

〈group hints〉

Those hints are related to *group by* processing, so they are valid only when *group by* clause exists.

〈group operation hints〉

USE_GROUP_HASH

If USE_GROUP_HASH hint is specified, then an optimizer uses a hash instant to process *group by*.

Generally, a hash instant is used to process *group by*. However, if rows on the subordinate nodes ascend by being sorted for the *group by* key column, then hash instants are not accumulated. Instead, *group by* i s processed by comparing the *group by* key column values of rows.

An optimizer does not accumulate hash instants when performing the cost estimation but guides the sub ordinate node to use an index. However, if the statistics information is incorrect, then the cost of this met hod can be more expensive. Therefore, use USE_GROUP_HASH hint in this case.

The following is an example of using USE_GROUP_HASH hint. Compare the execution plan before and af ter using the hint to figure out the process of USE_GROUP_HASH hint.

```
\EXPLAIN PLAN
   SELECT
          c_custkey,
          count(o_orderkey) as c_count
     FROM  customer LEFT OUTER JOIN orders
          ON  c_custkey = o_custkey
          AND o_comment not like '%special%requests%'
          AND c_mktsegment = 'BUILDING'
 GROUP BY   c_custkey;
>>>  start print plan
```

```
< Execution Plan >
============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
|    2  |      GROUP                                                        |
|    3  |        HASH JOIN (LEFT OUTER JOIN)                                |
|    4  |          INDEX ACCESS ("CUSTOMER", "CUSTOMER_PK_INDEX")           |
|    5  |          HASH JOIN INSTANT                                        |
|    6  |            TABLE ACCESS ("ORDERS")                                |
============================================================================
     1  -  TARGET : CUSTOMER.C_CUSTKEY, COUNT( ORDERS.O_ORDERKEY ) AS C_COUNT
     2  -  GROUP KEY : CUSTOMER.C_CUSTKEY
           RECORD COLUMN : COUNT( ORDERS.O_ORDERKEY )
           READ KEY COLUMN : CUSTOMER.C_CUSTKEY
           READ RECORD COLUMN : COUNT( ORDERS.O_ORDERKEY )
     3  -  JOINED COLUMN : CUSTOMER.C_CUSTKEY, ORDERS.O_ORDERKEY
     4  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_CUSTKEY, ORDERS.O_COMMENT
             LOGICAL FILTER : ORDERS.O_COMMENT NOT LIKE '%special%requests%'
     5  -  HASH KEY : CUSTOMER.C_CUSTKEY
           READ KEY COLUMN : CUSTOMER.C_CUSTKEY
             HASH FILTER : CUSTOMER.C_CUSTKEY = ORDERS.O_CUSTKEY
     6  -  READ INDEX COLUMN : CUSTOMER.C_CUSTKEY
<<<  end print plan
\EXPLAIN PLAN
   SELECT  /*+ USE_GROUP_HASH */
           c_custkey,
           count(o_orderkey) as c_count
     FROM  customer LEFT OUTER JOIN orders
           ON  c_custkey = o_custkey
           AND o_comment not like '%special%requests%'
           AND c_mktsegment = 'BUILDING'
 GROUP BY   c_custkey;
>>>  start print plan
< Execution Plan >
============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
```

```
|    2  |      GROUP HASH INSTANT                                  |
|    3  |        HASH JOIN (LEFT OUTER JOIN)                             |
|    4  |          TABLE ACCESS ("CUSTOMER")                       |
|    5  |          HASH JOIN INSTANT                                      |
|    6  |            TABLE ACCESS ("ORDERS")                              |
=========================================================================
     1  -  TARGET : CUSTOMER.C_CUSTKEY, COUNT( ORDERS.O_ORDERKEY ) AS C_COUNT
     2  -  GROUP KEY : CUSTOMER.C_CUSTKEY
           RECORD COLUMN : COUNT( ORDERS.O_ORDERKEY )
           READ KEY COLUMN : CUSTOMER.C_CUSTKEY
           READ RECORD COLUMN : COUNT( ORDERS.O_ORDERKEY )
     3  -  JOINED COLUMN : CUSTOMER.C_CUSTKEY, ORDERS.O_ORDERKEY
     4  -  READ COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_MKTSEGMENT
     5  -  HASH KEY : ORDERS.O_CUSTKEY
           RECORD COLUMN : ORDERS.O_ORDERKEY
           READ KEY COLUMN : ORDERS.O_CUSTKEY, ORDERS.O_ORDERKEY
             HASH FILTER : ORDERS.O_CUSTKEY = CUSTOMER.C_CUSTKEY
             LOGICAL FILTER : {CUSTOMER.C_MKTSEGMENT} = 'BUILDING'
     6  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_CUSTKEY, ORDERS.O_COMMENT
             LOGICAL FILTER : ORDERS.O_COMMENT NOT LIKE '%special%requests%'
<<<  end print plan
```

USE_GROUP_HASH(hash_bucket_count)

USE_GROUP_HASH(hash_bucket_count) hint is as same as USE_GROUP_HASH hint, but in addition to t hat it can specify the hash bucket count.

If statistics information is incorrect, then then hash bucket count of hash instants created for GROUP BY may be too many or too little so it may degrade the performance. In this case, specify the hash bucket co unt by using the hint to prevent the performance from degrading.

The following is an example of using USE_GROUP_HASH(hash_bucket_count) hint.

```
\EXPLAIN PLAN VERBOSE
   SELECT  /*+ USE_GROUP_HASH(15000) */
           c_custkey,
           count(o_orderkey) as c_count
     FROM  customer LEFT OUTER JOIN orders
           ON  c_custkey = o_custkey
           AND o_comment not like '%special%requests%'
           AND c_mktsegment = 'BUILDING'
 GROUP BY   c_custkey;
>>>  start print plan
```

```
< Execution Plan >
============================================================================
¦IDX¦  NODE DESCRIPTION                                  ¦   ROWS  ¦ Ellipsis ¦
----------------------------------------------------------------------------
¦ 0 ¦  SELECT STATEMENT                                  ¦  150000 ¦ ... ¦
¦ 1 ¦    QUERY BLOCK ("$QB_IDX_2")                       ¦  150000 ¦ ... ¦
¦ 2 ¦      GROUP HASH INSTANT                            ¦  150000 ¦ ... ¦
¦ 3 ¦        HASH JOIN (INVERTED LEFT OUTER JOIN)        ¦  430506 ¦ ... ¦
¦ 4 ¦          TABLE ACCESS ("ORDERS")                   ¦ 1483918 ¦ ... ¦
¦ 5 ¦          HASH JOIN INSTANT                         ¦  430506 ¦ ... ¦
¦ 6 ¦            TABLE ACCESS ("CUSTOMER")               ¦  150000 ¦ ... ¦
============================================================================
     1  -  TARGET : CUSTOMER.C_CUSTKEY, COUNT( ORDERS.O_ORDERKEY ) AS C_COUNT
     2  -  GROUP KEY : CUSTOMER.C_CUSTKEY
           RECORD COLUMN : COUNT( ORDERS.O_ORDERKEY )
           READ KEY COLUMN : CUSTOMER.C_CUSTKEY
           READ RECORD COLUMN : COUNT( ORDERS.O_ORDERKEY )
           HASH BUCKET COUNT : 15000
     3  -  JOINED COLUMN : CUSTOMER.C_CUSTKEY, ORDERS.O_ORDERKEY
     4  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_CUSTKEY, ORDERS.O_COMMENT
             LOGICAL FILTER : ORDERS.O_COMMENT NOT LIKE '%special%requests%'
     5  -  HASH KEY : CUSTOMER.C_CUSTKEY
           RECORD COLUMN : CUSTOMER.C_MKTSEGMENT
           READ KEY COLUMN : CUSTOMER.C_CUSTKEY
           READ RECORD COLUMN : CUSTOMER.C_MKTSEGMENT
             HASH FILTER : CUSTOMER.C_CUSTKEY = ORDERS.O_CUSTKEY
             PHYSICAL FILTER : CUSTOMER.C_MKTSEGMENT = 'BUILDING'
           HASH BUCKET COUNT : 150000
     6  -  READ COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_MKTSEGMENT
<<<  end print plan
```

In the example above, the hash bucket count is specified as many as the number of expected output row s of GROUP HASH INSTANT.

NO_USE_GROUP_HASH

If NO_USE_GROUP_HASH hint is specified, then an optimizer processes *group by* without using a hash in stant.

In other words, if the intermediate result is sorted for group key and ascends from the subordinate plan, t hen it is used. Otherwise, it processes *group by* by using the sort instant.

```
\EXPLAIN PLAN
   SELECT  /*+ NO_USE_GROUP_HASH */
           c_custkey,
           count(o_orderkey) as c_count
     FROM  customer LEFT OUTER JOIN orders
           ON  c_custkey = o_custkey
           AND o_comment not like '%special%requests%'
           AND c_mktsegment = 'BUILDING'
 GROUP BY   c_custkey;
>>>  start print plan
< Execution Plan >
================================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                     ¦
--------------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                     ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                        ¦
¦  2    ¦      GROUP SORT INSTANT                                         ¦
¦    3  ¦        HASH JOIN (INVERTED LEFT OUTER JOIN)                         ¦
¦    4  ¦          TABLE ACCESS ("ORDERS")                                    ¦
¦    5  ¦          HASH JOIN INSTANT                                          ¦
¦    6  ¦            TABLE ACCESS ("CUSTOMER")                                ¦
================================================================================
     1  -  TARGET : CUSTOMER.C_CUSTKEY, COUNT( ORDERS.O_ORDERKEY ) AS C_COUNT
     2  -  GROUP KEY : CUSTOMER.C_CUSTKEY
           RECORD COLUMN : COUNT( ORDERS.O_ORDERKEY )
           READ KEY COLUMN : CUSTOMER.C_CUSTKEY
           READ RECORD COLUMN : COUNT( ORDERS.O_ORDERKEY )
     3  -  JOINED COLUMN : CUSTOMER.C_CUSTKEY, ORDERS.O_ORDERKEY
     4  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_CUSTKEY, ORDERS.O_COMMENT
             LOGICAL FILTER : ORDERS.O_COMMENT NOT LIKE '%special%requests%'
     5  -  HASH KEY : CUSTOMER.C_CUSTKEY
           RECORD COLUMN : CUSTOMER.C_MKTSEGMENT
           READ KEY COLUMN : CUSTOMER.C_CUSTKEY
           READ RECORD COLUMN : CUSTOMER.C_MKTSEGMENT
             HASH FILTER : CUSTOMER.C_CUSTKEY = ORDERS.O_CUSTKEY
             PHYSICAL FILTER : CUSTOMER.C_MKTSEGMENT = 'BUILDING'
     6  -  READ COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_MKTSEGMENT
<<<  end print plan
```

USE_GROUP_SORT

If USE_GROUP_SORT hint is specified, then an optimizer processes *group by* by using a sort instant.

```
\EXPLAIN PLAN
   SELECT  /*+ USE_GROUP_SORT */
           c_custkey,
           count(o_orderkey) as c_count
     FROM  customer LEFT OUTER JOIN orders
           ON  c_custkey = o_custkey
           AND o_comment not like '%special%requests%'
           AND c_mktsegment = 'BUILDING'
 GROUP BY   c_custkey;
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                  ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                  ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                     ¦
¦  2    ¦      GROUP SORT INSTANT                                  ¦
¦    3  ¦        HASH JOIN (INVERTED LEFT OUTER JOIN)                        ¦
¦    4  ¦          TABLE ACCESS ("ORDERS")                                   ¦
¦    5  ¦          HASH JOIN INSTANT                                         ¦
¦    6  ¦            TABLE ACCESS ("CUSTOMER")                               ¦
=============================================================================
     1  -  TARGET : CUSTOMER.C_CUSTKEY, COUNT( ORDERS.O_ORDERKEY ) AS C_COUNT
     2  -  GROUP KEY : CUSTOMER.C_CUSTKEY
           RECORD COLUMN : COUNT( ORDERS.O_ORDERKEY )
           READ KEY COLUMN : CUSTOMER.C_CUSTKEY
           READ RECORD COLUMN : COUNT( ORDERS.O_ORDERKEY )
     3  -  JOINED COLUMN : CUSTOMER.C_CUSTKEY, ORDERS.O_ORDERKEY
     4  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_CUSTKEY, ORDERS.O_COMMENT
             LOGICAL FILTER : ORDERS.O_COMMENT NOT LIKE '%special%requests%'
     5  -  HASH KEY : CUSTOMER.C_CUSTKEY
           RECORD COLUMN : CUSTOMER.C_MKTSEGMENT
           READ KEY COLUMN : CUSTOMER.C_CUSTKEY
           READ RECORD COLUMN : CUSTOMER.C_MKTSEGMENT
             HASH FILTER : CUSTOMER.C_CUSTKEY = ORDERS.O_CUSTKEY
             PHYSICAL FILTER : CUSTOMER.C_MKTSEGMENT = 'BUILDING'
     6  -  READ COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_MKTSEGMENT
<<<  end print plan
```

NO_USE_GROUP_SORT

If NO_USE_GROUP_SORT hint is specified, then an optimizer processes group by without using a sort inst ant.

```
\EXPLAIN PLAN
   SELECT  /*+ NO_USE_GROUP_SORT */
           c_custkey,
           count(o_orderkey) as c_count
     FROM  customer LEFT OUTER JOIN orders
           ON  c_custkey = o_custkey
           AND o_comment not like '%special%requests%'
           AND c_mktsegment = 'BUILDING'
 GROUP BY   c_custkey;
>>>  start print plan
< Execution Plan >
================================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                     ¦
--------------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                     ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                        ¦
¦   2   ¦    GROUP HASH INSTANT                                          ¦
¦    3  ¦        HASH JOIN (INVERTED LEFT OUTER JOIN)                         ¦
¦    4  ¦          TABLE ACCESS ("ORDERS")                                    ¦
¦    5  ¦          HASH JOIN INSTANT                                          ¦
¦    6  ¦            TABLE ACCESS ("CUSTOMER")                                ¦
================================================================================
     1  -  TARGET : CUSTOMER.C_CUSTKEY, COUNT( ORDERS.O_ORDERKEY ) AS C_COUNT
     2  -  GROUP KEY : CUSTOMER.C_CUSTKEY
           RECORD COLUMN : COUNT( ORDERS.O_ORDERKEY )
           READ KEY COLUMN : CUSTOMER.C_CUSTKEY
           READ RECORD COLUMN : COUNT( ORDERS.O_ORDERKEY )
     3  -  JOINED COLUMN : CUSTOMER.C_CUSTKEY, ORDERS.O_ORDERKEY
     4  -  READ COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_CUSTKEY, ORDERS.O_COMMENT
             LOGICAL FILTER : ORDERS.O_COMMENT NOT LIKE '%special%requests%'
     5  -  HASH KEY : CUSTOMER.C_CUSTKEY
           RECORD COLUMN : CUSTOMER.C_MKTSEGMENT
           READ KEY COLUMN : CUSTOMER.C_CUSTKEY
           READ RECORD COLUMN : CUSTOMER.C_MKTSEGMENT
             HASH FILTER : CUSTOMER.C_CUSTKEY = ORDERS.O_CUSTKEY
             PHYSICAL FILTER : CUSTOMER.C_MKTSEGMENT = 'BUILDING'
     6  -  READ COLUMN : CUSTOMER.C_CUSTKEY, CUSTOMER.C_MKTSEGMENT
```

```
<<<  end print plan
```

〈group driver hints〉

Those hints are available when performing *group by* clause in the cluster system. They are ignored in the standalone system.

LOCAL_GROUP

It performs *group by* in the current server.

The following is an example of using LOCAL_GROUP hint.

```
\EXPLAIN PLAN
SELECT /*+ LOCAL_GROUP */
       c_custkey, COUNT(o_orderkey)
  FROM customer, orders
 WHERE c_custkey = o_custkey
   AND o_orderdate >= date '1993-07-01'
   AND c_nationkey = 1
   AND o_orderstatus = 'F'
   AND o_orderpriority = '2-HIGH'
GROUP BY c_custkey;
>>>  start print plan
< Execution Plan >
================================================================================
¦IDX¦  NODE DESCRIPTION                                                  ¦ ROWS ¦
--------------------------------------------------------------------------------
¦ 0 ¦  SELECT STATEMENT                                                  ¦ 2094 ¦
¦ 1 ¦    QUERY BLOCK ("$QB_IDX_2")                                       ¦ 2094 ¦
¦ 2 ¦      GROUP HASH INSTANT                                            ¦ 2094 ¦
¦ 3 ¦        PLAN BASED CLUSTER                          ¦LOCAL/REMOTE 3049 ¦
¦ 4 ¦          NESTED JOIN (INNER JOIN)                                  ¦ 1004 ¦
¦ 5 ¦            INDEX ACCESS ("CUSTOMER", "CUSTOMER_NATIONKEY_FK") ¦ 5975 ¦
¦ 6 ¦            INDEX ACCESS ("ORDERS", "ORDERS_CUSTKEY_FK")       ¦ 1004 ¦
================================================================================
     1  -  TARGET : CUSTOMER.C_CUSTKEY, COUNT( ORDERS.O_ORDERKEY )
     2  -  GROUP KEY : CUSTOMER.C_CUSTKEY
           RECORD COLUMN : COUNT( ORDERS.O_ORDERKEY )
           READ KEY COLUMN : CUSTOMER.C_CUSTKEY
           READ RECORD COLUMN : COUNT( ORDERS.O_ORDERKEY )
     3  -  SQL : SELECT /*+ KEEP_JOINED_TABLE
                            USE_NL_IN( _A1 )
```

```
                        INDEX( _A2, "PUBLIC"."CUSTOMER_NATIONKEY_FK" )
                        INDEX( _A1, "PUBLIC"."ORDERS_CUSTKEY_FK" )
                    */
                    "_A2"."C_CUSTKEY", "_A1"."O_ORDERKEY"
                FROM ( "PUBLIC"."CUSTOMER"@LOCAL AS "_A2"
                       INNER JOIN
                       "PUBLIC"."ORDERS"@LOCAL AS "_A1" ON true
                    ) ALIAS "_A3"
              WHERE "_A2"."C_NATIONKEY" = :_V0
                AND "_A1"."O_CUSTKEY" = "_A2"."C_CUSTKEY"
                AND "_A1"."O_ORDERSTATUS" = :_V1
                AND "_A1"."O_ORDERDATE" >= :_V2
                AND "_A1"."O_ORDERPRIORITY" = :_V3
        TARGET DOMAIN : G1(G1N1,G1N2) 1004 rows,
                        G2(G2N1,G2N2) 995 rows,
                        G3(G3N1,G3N2) 1050 rows
    4  -  JOINED COLUMN : CUSTOMER.C_CUSTKEY, ORDERS.O_ORDERKEY
    5  -  CLONED
        READ INDEX COLUMN : CUSTOMER.C_NATIONKEY
        READ TABLE COLUMN : CUSTOMER.C_CUSTKEY
          MIN RANGE : CUSTOMER.C_NATIONKEY = 1
          MAX RANGE : CUSTOMER.C_NATIONKEY = 1
    6  -  HASH SHARD ( # 3 )
        READ INDEX COLUMN : ORDERS.O_CUSTKEY
        READ TABLE COLUMN : ORDERS.O_ORDERKEY, ORDERS.O_ORDERSTATUS, ORDERS.O_ORDERDATE,
ORDERS.O_ORDERPRIORITY
          MIN RANGE : ORDERS.O_CUSTKEY = {CUSTOMER.C_CUSTKEY}
          MAX RANGE : ORDERS.O_CUSTKEY = {CUSTOMER.C_CUSTKEY}
          PHYSICAL TABLE FILTER : ORDERS.O_ORDERSTATUS = 'F' AND ORDERS.O_ORDERDATE >=
DATE'1993-07-01' AND ORDERS.O_ORDERPRIORITY = '2-HIGH'
<<<  end print plan
```

In the execution plan above, it brings all join results from G1, G2 and G3, then performs *group by* in a loc al.

Generally, if it is a sharded table, it is recommended to process it with remote group if possible. It is beca use that it can process grouping in parallel and the intermediate results to be transferred are decreased.

However, if the target rows of group by is few, so the parallel processing effect is not expected, then it is recommended to process *group by* in a local.

REMOTE_GROUP

It performs *group by* in a server of each group.

The following is an example of using REMOTE_GROUP hint.

```
\EXPLAIN PLAN
SELECT /*+ REMOTE_GROUP */
       c_custkey, COUNT(o_orderkey)
  FROM customer, orders
 WHERE c_custkey = o_custkey
GROUP BY c_custkey;
>>>  start print plan
< Execution Plan >
================================================================================
| IDX |  NODE DESCRIPTION                                          |   ROWS |
--------------------------------------------------------------------------------
| 0 |  SELECT STATEMENT                                            |  99996 |
| 1 |    QUERY BLOCK ("$QB_IDX_2")                                 |  99996 |
| 2 |      SINGLE CLUSTER                                | LOCAL/REMOTE 99996 |
| 3 |        SELECT STATEMENT                                      |  98218 |
| 4 |          QUERY BLOCK ("$QB_IDX_2")                           |  98218 |
| 5 |            GROUP HASH INSTANT                                |  98218 |
| 6 |              HASH JOIN (INNER JOIN)                          | 500004 |
| 7 |                TABLE ACCESS ("ORDERS" AS _A2)                | 500004 |
| 8 |                HASH JOIN INSTANT                             | 500004 |
| 9 |                  INDEX ACCESS ("CUSTOMER" AS _A1")           | 150000 |
================================================================================
     1  -  TARGET : CUSTOMER.C_CUSTKEY, COUNT( ORDERS.O_ORDERKEY )
     2  -  SQL : SELECT /*+ USE_GROUP_HASH(150000)
                            KEEP_JOINED_TABLE
                            USE_HASH_IN( _A1, 150000 )
                            FULL( _A2 )
                            INDEX( _A1, "PUBLIC"."CUSTOMER_PK_INDEX" )
                        */
                        "_A1"."C_CUSTKEY", COUNT( "_A2"."O_ORDERKEY" )
                   FROM ( "PUBLIC"."ORDERS"@LOCAL AS "_A2"
                           INNER JOIN
                          "PUBLIC"."CUSTOMER"@LOCAL AS "_A1"
                          ON "_A1"."C_CUSTKEY" = "_A2"."O_CUSTKEY"
                        ) ALIAS "_A3"
                  GROUP BY "_A1"."C_CUSTKEY"
```

```
          TARGET DOMAIN : G1(G1N1,G1N2) 98218 rows,
                          G2(G2N1,G2N2) 98174 rows,
                          G3(G3N1,G3N2) 98138 rows
          RE-GROUPING
            GROUP KEY : CUSTOMER.C_CUSTKEY
            AGGREGATION : SUM( COUNT( ORDERS.O_ORDERKEY ) )
    4  -  TARGET : _A1.C_CUSTKEY, COUNT( _A2.O_ORDERKEY )
    5  -  GROUP KEY : _A1.C_CUSTKEY
          RECORD COLUMN : COUNT( _A2.O_ORDERKEY )
          READ KEY COLUMN : _A1.C_CUSTKEY
          READ RECORD COLUMN : COUNT( _A2.O_ORDERKEY )
    6  -  JOINED COLUMN : _A1.C_CUSTKEY, _A2.O_ORDERKEY
    7  -  HASH SHARD ( # 3 )
          READ COLUMN : _A2.O_ORDERKEY, _A2.O_CUSTKEY
    8  -  HASH KEY : _A1.C_CUSTKEY
          READ KEY COLUMN : _A1.C_CUSTKEY
            HASH FILTER : _A1.C_CUSTKEY = _A2.O_CUSTKEY
          FETCH ONE ROW
    9  -  CLONED
          READ INDEX COLUMN : _A1.C_CUSTKEY
<<<  end print plan
```

In the execution plan above, it processes group by in each server. After processing the join in each server, it groups 500,000 data and outputs 100,000 intermediate results. In this way, it can process the groupin g in parallel and decrease intermediate results to be brought to the network.

If processing the example above in a local, then it should bring 1,500,000 join results and perform group by for 1,500,000 results. In this case, it cost for bring 1,500,000 data to the network and for grouping 1, 500,000 data at once.

〈group cluster regrouping hints〉

Those hints are available when performing *group by* in the cluster system. They are ignored in the standal one system.

MERGE_GROUP

MERGE_GROUP hint is available when it satisfies the following conditions.

- Group by clause exists.
- It can be performed with remote group.
- The intermediate results ascend from the subordinate node of a group in a state that the order for th e group by key column is guaranteed.

Using MERGE_GROUP hint guarantees the order even after fetching the intermediate results to the driver.

The following is an example of using MERGE_GROUP hint.

```
\EXPLAIN PLAN
SELECT /*+ MERGE_GROUP */
       o_custkey
  FROM orders
 WHERE o_custkey > 0
GROUP BY o_custkey;
O_CUSTKEY
---------
        1
        2
        4
        5
        7
        8
       10
       11
      ...
   149992
   149993
   149995
   149996
   149998
   149999
99996 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
¦IDX¦  NODE DESCRIPTION                                ¦               ROWS ¦
--------------------------------------------------------------------------------
¦ 0 ¦  SELECT STATEMENT                                ¦              99996 ¦
¦ 1 ¦    QUERY BLOCK ("$QB_IDX_2")                     ¦              99996 ¦
¦ 2 ¦      MULTIPLE CLUSTER                            ¦ LOCAL/REMOTE 99996 ¦
¦ 3 ¦        SELECT STATEMENT                          ¦              98218 ¦
¦ 4 ¦          QUERY BLOCK ("$QB_IDX_2")               ¦              98218 ¦
¦ 5 ¦            GROUP                                 ¦              98218 ¦
¦ 6 ¦              INDEX ACCESS ("ORDERS" AS _A1)      ¦             500004 ¦
================================================================================
     1  -  TARGET : ORDERS.O_CUSTKEY
```

```
    2  -  SQL : SELECT /*+ INDEX( _A1, "PUBLIC"."ORDERS_CUSTKEY_FK" ) */
                        "_A1"."O_CUSTKEY"
                   FROM "PUBLIC"."ORDERS"@LOCAL AS "_A1"
                  WHERE "_A1"."O_CUSTKEY" > :_V0
               GROUP BY "_A1"."O_CUSTKEY"
               ORDER BY "_A1"."O_CUSTKEY" ASC NULLS LAST
          TARGET DOMAIN : G1(G1N1,G1N2) 98218 rows,
                          G2(G2N1,G2N2) 98174 rows,
                          G3(G3N1,G3N2) 98138 rows
          MERGE GROUPING
            SORT KEY : ORDERS.O_CUSTKEY
            GROUP KEY : ORDERS.O_CUSTKEY
    4  -  TARGET : _A1.O_CUSTKEY
    5  -  GROUP KEY : _A1.O_CUSTKEY
    6  -  HASH SHARD ( # 3 )
          READ INDEX COLUMN : _A1.O_CUSTKEY
            MIN RANGE : _A1.O_CUSTKEY > :_V0
            MAX RANGE : _A1.O_CUSTKEY IS NOT NULL
<<<  end print plan
```

In the execution result above, they are output in an order of group key col.

〈distinct hints〉

Those hints are related to *distinct* processing, so they are valid only when *distinct* clause exists.

〈distinct operation hints〉

USE_DISTINCT_HASH

If USE_DISTINCT_HASH hint is specified, then an optimizer uses a hash instant to process *distinct*.

Generally, a hash instant is used to process *distinct*. However, if rows on the subordinate nodes ascend b y being sorted for the distinct key column, then hash instants are not accumulated. Instead, *distinct* is pro cessed by comparing the distinct key column values of rows.

An optimizer does not accumulate hash instants when performing the cost estimation but guides the sub ordinate node to use an index. However, if the statistics information is incorrect, then the cost of this met hod can be more expensive. Therefore, use USE_DISTINCT_HASH hint in this case.

The following is an example of using USE_DISTINCT_HASH hint. Compare the execution plan before and after using the hint to figure out the process of USE_DISTINCT_HASH hint.

```
EXPLAIN PLAN
   SELECT
          DISTINCT
          c_nationkey
     FROM  customer
    WHERE  c_comment not like '%special%requests%'
      AND  c_nationkey > 5;
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                  ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                  ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                     ¦
¦    2  ¦      GROUP                                                 ¦
¦    3  ¦        INDEX ACCESS ("CUSTOMER", "CUSTOMER_NATIONKEY_FK")    ¦
=============================================================================
     1  -  TARGET : CUSTOMER.C_NATIONKEY
     2  -  GROUP KEY : CUSTOMER.C_NATIONKEY
     3  -  READ INDEX COLUMN : CUSTOMER.C_NATIONKEY
           READ TABLE COLUMN : CUSTOMER.C_COMMENT
             MIN RANGE : CUSTOMER.C_NATIONKEY > 5
             MAX RANGE : CUSTOMER.C_NATIONKEY IS NOT NULL
             LOGICAL TABLE FILTER : CUSTOMER.C_COMMENT NOT LIKE '%special%requests%'
<<<  end print plan
\EXPLAIN PLAN
   SELECT  /*+ USE_DISTINCT_HASH */
          DISTINCT
          c_nationkey
     FROM  customer
    WHERE  c_comment not like '%special%requests%'
      AND  c_nationkey > 5;
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                  ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                  ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                     ¦
¦    2  ¦      GROUP HASH INSTANT                                  ¦
¦    3  ¦        INDEX ACCESS ("CUSTOMER", "CUSTOMER_NATIONKEY_FK")    ¦
```

```
=============================================================================
     1  -  TARGET : CUSTOMER.C_NATIONKEY
     2  -  GROUP KEY : CUSTOMER.C_NATIONKEY
           READ KEY COLUMN : CUSTOMER.C_NATIONKEY
     3  -  READ INDEX COLUMN : CUSTOMER.C_NATIONKEY
           READ TABLE COLUMN : CUSTOMER.C_COMMENT
             MIN RANGE : CUSTOMER.C_NATIONKEY > 5
             MAX RANGE : CUSTOMER.C_NATIONKEY IS NOT NULL
             LOGICAL TABLE FILTER : CUSTOMER.C_COMMENT NOT LIKE '%special%requests%'
<<<  end print plan
```

USE_DISTINCT_HASH(hash__bucket_count)

USE_DISTINCT_HASH(hash__bucket_count) hint is as same as USE_DISTINCT_HASH hint, but in addition to that it can specify the hash bucket count.

If statistics information is incorrect, then hash bucket count of hash instants created for DISTINCT may be too many or too little so it may degrade the performance. In this case, specify the hash bucket count by u sing the hint to prevent the performance from degrading.

The following is an example of using USE_DISTINCT_HASH hint(hash_bucket_count).

```
\EXPLAIN PLAN VERBOSE
   SELECT  /*+ USE_DISTINCT_HASH(19) */
           DISTINCT
           c_nationkey
     FROM  customer
    WHERE  c_comment not like '%special%requests%'
      AND  c_nationkey > 5;
>>>  start print plan
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                                  |  ROWS | Ellipsis |
-----------------------------------------------------------------------------
| 0 |SELECT STATEMENT                                        |    19 | ... |
| 1 |  QUERY BLOCK ("$QB_IDX_2")                             |    19 | ... |
| 2 |    GROUP HASH INSTANT                                  |    19 | ... |
| 3 |      INDEX ACCESS ("CUSTOMER", "CUSTOMER_NATIONKEY_FK")|111562 | ... |
=============================================================================
     1  -  TARGET : CUSTOMER.C_NATIONKEY
     2  -  GROUP KEY : CUSTOMER.C_NATIONKEY
           READ KEY COLUMN : CUSTOMER.C_NATIONKEY
           HASH BUCKET COUNT : 19
```

```
     3  -  READ INDEX COLUMN : CUSTOMER.C_NATIONKEY
           READ TABLE COLUMN : CUSTOMER.C_COMMENT
             MIN RANGE : CUSTOMER.C_NATIONKEY > 5
             MAX RANGE : CUSTOMER.C_NATIONKEY IS NOT NULL
             LOGICAL TABLE FILTER : CUSTOMER.C_COMMENT NOT LIKE '%special%requests%'
<<<  end print plan
```

In the example above, the hash bucket count is specified as many as the number of expected output rows of GROUP HASH INSTANT.

NO_USE_DISTINCT_HASH

If NO_USE_DISTINCT_HASH hint is specified, then an optimizer processes *distinct* without using a hash instant.

In other words, if the intermediate result is sorted for distinct key and ascends from the subordinate plan, then it is used. Otherwise, it processes *distinct* by using the sort instant.

```
\EXPLAIN PLAN
   SELECT  /*+ NO_USE_DISTINCT_HASH */
           DISTINCT
           c_nationkey
     FROM  customer
    WHERE  c_comment not like '%special%requests%'
      AND  c_nationkey > 5;
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                  ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                    ¦
¦    2  ¦      GROUP                                                      ¦
¦    3  ¦        INDEX ACCESS ("CUSTOMER", "CUSTOMER_NATIONKEY_FK")       ¦
=============================================================================
     1  -  TARGET : CUSTOMER.C_NATIONKEY
     2  -  GROUP KEY : CUSTOMER.C_NATIONKEY
     3  -  READ INDEX COLUMN : CUSTOMER.C_NATIONKEY
           READ TABLE COLUMN : CUSTOMER.C_COMMENT
             MIN RANGE : CUSTOMER.C_NATIONKEY > 5
             MAX RANGE : CUSTOMER.C_NATIONKEY IS NOT NULL
             LOGICAL TABLE FILTER : CUSTOMER.C_COMMENT NOT LIKE '%special%requests%'
<<<  end print plan
```

USE_DISTINCT_SORT

If USE_DISTINCT_SORT hint is specified, then an optimizer processes *distinct* by using a sort instant.

```
\EXPLAIN PLAN
   SELECT  /*+ USE_DISTINCT_SORT */
           DISTINCT
           c_nationkey
     FROM  customer
    WHERE  c_comment not like '%special%requests%'
      AND  c_nationkey > 5;
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                 ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                    ¦
¦    2  ¦      GROUP SORT INSTANT                                  ¦
¦    3  ¦        TABLE ACCESS ("CUSTOMER")                                ¦
=============================================================================
     1  -  TARGET : CUSTOMER.C_NATIONKEY
     2  -  GROUP KEY : CUSTOMER.C_NATIONKEY
           READ KEY COLUMN : CUSTOMER.C_NATIONKEY
     3  -  READ COLUMN : CUSTOMER.C_NATIONKEY, CUSTOMER.C_COMMENT
             PHYSICAL FILTER : CUSTOMER.C_NATIONKEY > 5
             LOGICAL FILTER : CUSTOMER.C_COMMENT NOT LIKE '%special%requests%'
<<<  end print plan
```

NO_USE_DISTINCT_SORT

If NO_USE_DISTINCT_SORT hint is specified, then an optimizer processes *distinct* without using a sort inst ant.

```
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                 ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                    ¦
¦    2  ¦      GROUP HASH INSTANT                                  ¦
¦    3  ¦        TABLE ACCESS ("CUSTOMER")                                ¦
```

```
=============================================================================
     1  -  TARGET : CUSTOMER.C_NATIONKEY
     2  -  GROUP KEY : CUSTOMER.C_NATIONKEY
           READ KEY COLUMN : CUSTOMER.C_NATIONKEY
     3  -  READ COLUMN : CUSTOMER.C_NATIONKEY, CUSTOMER.C_COMMENT
             PHYSICAL FILTER : CUSTOMER.C_NATIONKEY > 5
             LOGICAL FILTER : CUSTOMER.C_COMMENT NOT LIKE '%special%requests%'
<<<  end print plan
```

〈distinct driver hints〉

Those hints are available when performing distinct clause in the cluster system. They are ignored in the st andalone system.

LOCAL_DISTINCT

It performs distinct in the current server.

The following is an example of using LOCAL_DISTINCT hint.

```
\EXPLAIN PLAN
SELECT /*+ LOCAL_DISTINCT */
       DISTINCT o_custkey
  FROM orders
 WHERE o_orderdate = date '1995-03-15'
   AND o_orderstatus = 'F'
   AND o_orderpriority = '2-HIGH';
>>>  start print plan
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                              |            ROWS |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                              |              34 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                   |              34 |
|    2  |      GROUP HASH INSTANT                        |              34 |
|    3  |        PLAN BASED CLUSTER                      | LOCAL/REMOTE 34 |
|    4  |          TABLE ACCESS ("ORDERS")               |              13 |
=============================================================================
     1  -  TARGET : ORDERS.O_CUSTKEY
     2  -  GROUP KEY : ORDERS.O_CUSTKEY
           READ KEY COLUMN : ORDERS.O_CUSTKEY
     3  -  SQL : SELECT /*+ FULL( _A1 ) */
                        "_A1"."O_CUSTKEY"
```

```
                  FROM "PUBLIC"."ORDERS"@LOCAL AS "_A1"
                 WHERE "_A1"."O_ORDERSTATUS" = :_V0
                   AND "_A1"."O_ORDERDATE" = :_V1
                   AND "_A1"."O_ORDERPRIORITY" = :_V2
          TARGET DOMAIN : G1(G1N1,G1N2) 13 rows,
                          G2(G2N1,G2N2) 11 rows,
                          G3(G3N1,G3N2) 10 rows
     4  -  HASH SHARD ( # 3 )
          READ COLUMN : ORDERS.O_CUSTKEY, ORDERS.O_ORDERSTATUS, ORDERS.O_ORDERDATE,
ORDERS.O_ORDERPRIORITY
            PHYSICAL FILTER : ORDERS.O_ORDERSTATUS = 'F' AND ORDERS.O_ORDERDATE =
DATE'1995-03-15' AND ORDERS.O_ORDERPRIORITY = '2-HIGH'
<<<  end print plan
```

In the execution plan above, it brings intermediate results satisfying conditions from orders to a local, the n creates GROUP HASH INSTANT for distinct.

REMOTE_DISTINCT

It performs distinct in a server of each group.

The following is an example of using REMOTE_DISTINCT hint.

```
\EXPLAIN PLAN
SELECT /*+ REMOTE_DISTINCT */
       DISTINCT o_orderstatus
  FROM orders
 WHERE o_orderdate = date '1995-03-15';
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                                |          ROWS |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                |             3 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                     |             3 |
|    2  |      SINGLE CLUSTER                              |LOCAL/REMOTE 3 |
|    3  |        SELECT STATEMENT                          |             3 |
|    4  |          QUERY BLOCK ("$QB_IDX_2")               |             3 |
|    5  |            GROUP HASH INSTANT                    |             3 |
|    6  |              TABLE ACCESS ("ORDERS" AS _A1)      |           221 |
================================================================================
     1  -  TARGET : ORDERS.O_ORDERSTATUS
     2  -  SQL : SELECT /*+ USE_DISTINCT_HASH(3)
```

```
                        FULL( _A1 )
                    */
                    DISTINCT "_A1"."O_ORDERSTATUS"
                FROM "PUBLIC"."ORDERS"@LOCAL AS "_A1"
               WHERE "_A1"."O_ORDERDATE" = :_V0
        TARGET DOMAIN : G1(G1N1,G1N2) 3 rows,
                        G2(G2N1,G2N2) 3 rows,
                        G3(G3N1,G3N2) 3 rows
        RE-GROUPING
          GROUP KEY : ORDERS.O_ORDERSTATUS
    4  -  TARGET : _A1.O_ORDERSTATUS
    5  -  GROUP KEY : _A1.O_ORDERSTATUS
          READ KEY COLUMN : _A1.O_ORDERSTATUS
    6  -  HASH SHARD ( # 3 )
          READ COLUMN : _A1.O_ORDERSTATUS, _A1.O_ORDERDATE
            PHYSICAL FILTER : _A1.O_ORDERDATE = :_V0
<<<  end print plan
```

In the execution plan above, it performs remote distinct by transferring SQL including DISTINCT.

〈distinct cluster regrouping hints〉

Those hints are available when performing the distinct clause in the cluster system. It is ignored in the sta ndalone system.

MERGE_DISTINCT

MERGE_DISTINCT hint is available when it satisfies the following conditions.

- A distinct clause exists.
- It can be performed with remote distinct.
- The intermediate result ascends from the subordinate node of the distinct node in a state that the or der for the distinct key column is guaranteed.

Using MERGE_DISTINCT hint guarantees the order even after fetching the intermediate result to the drive r.

The following is an example of using MERGE_DISTINCT hint.

```
\EXPLAIN PLAN
SELECT /*+ MERGE_DISTINCT */
       DISTINCT o_custkey
  FROM orders
 WHERE o_custkey > 0
```

```
ORDER BY o_custkey;
O_CUSTKEY
---------
        1
        2
        4
      ...
99996 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
¦IDX¦  NODE DESCRIPTION                                                        ¦
--------------------------------------------------------------------------------
¦ 0 ¦  SELECT STATEMENT                                                        ¦
¦ 1 ¦    QUERY BLOCK ("$QB_IDX_2")                                             ¦
¦ 2 ¦      MULTIPLE CLUSTER                                                    ¦
¦ 3 ¦        SELECT STATEMENT                                                  ¦
¦ 4 ¦          QUERY BLOCK ("$QB_IDX_2")                                       ¦
¦ 5 ¦            GROUP                                                         ¦
¦ 6 ¦              INDEX ACCESS ("ORDERS" AS _A1, "ORDERS_CUSTKEY_FK")         ¦
================================================================================
     1  -  TARGET : ORDERS.O_CUSTKEY
     2  -  SQL : SELECT /*+ INDEX( _A1, "PUBLIC"."ORDERS_CUSTKEY_FK" ) */
                         DISTINCT "_A1"."O_CUSTKEY"
                    FROM "PUBLIC"."ORDERS"@LOCAL AS "_A1"
                   WHERE "_A1"."O_CUSTKEY" > :_V0
                ORDER BY "_A1"."O_CUSTKEY" ASC NULLS LAST
           TARGET DOMAIN : G1(G1N1,G1N2) 98218 rows,
                           G2(G2N1,G2N2) 98174 rows,
                           G3(G3N1,G3N2) 98138 rows
           MERGE GROUPING
             SORT KEY : ORDERS.O_CUSTKEY
             GROUP KEY : ORDERS.O_CUSTKEY
     4  -  TARGET : _A1.O_CUSTKEY
     5  -  GROUP KEY : _A1.O_CUSTKEY
     6  -  HASH SHARD ( # 3 )
           READ INDEX COLUMN : _A1.O_CUSTKEY
             MIN RANGE : _A1.O_CUSTKEY > :_V0
             MAX RANGE : _A1.O_CUSTKEY IS NOT NULL
<<<  end print plan
```

In the execution plan above, *MULTIPLE CLUSTER(IDX:2)* keeps the order for o_custkey. Therefore, SORT n ode for ORDER BY is useless, so it is dropped.

〈order hints〉

Those hints are related to *order by* processing, so they are valid only when *order by* clause exists.

〈order operation hints〉

USE_ORDER_SORT

If USE_ORDER_SORT hint is specified, then an optimizer uses a sort instant to process *order by*.

Generally, a sort instant is used to process *order by*. However, if rows on the subordinate nodes ascend b y being sorted for the sort key column, then *order by* is processed without accumulating sort instants.

An optimizer does not accumulate sort instants when performing the cost estimation but guides the subo rdinate node to use an index. However, if the statistics information is incorrect, then the cost of this meth od can be more expensive. Therefore, use USE_ORDER_SORT hint in this case.

The following is an example of using USE_ORDER_SORT hint. Compare the execution plan before and aft er using the hint to figure out the process of USE_ORDER_SORT hint.

```
\EXPLAIN PLAN
  SELECT n_nationkey,
         n_name
    FROM nation
ORDER BY n_nationkey;
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                 ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                 ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                    ¦
¦    2  ¦      INDEX ACCESS ("NATION", "NATION_PK_INDEX")       ¦
=============================================================================
     1  -  TARGET : NATION.N_NATIONKEY, NATION.N_NAME
     2  -  READ INDEX COLUMN : NATION.N_NATIONKEY
           READ TABLE COLUMN : NATION.N_NAME
<<<  end print plan
\EXPLAIN PLAN
  SELECT /*+ USE_ORDER_SORT */
         n_nationkey,
```

```
        n_name
    FROM nation
ORDER BY n_nationkey;
>>>  start print plan
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
|    2  |      SORT INSTANT                                                 |
|    3  |        TABLE ACCESS ("NATION")                                    |
=============================================================================
     1  -  TARGET : NATION.N_NATIONKEY, NATION.N_NAME
     2  -  SORT KEY : "NATION.N_NATIONKEY ASC NULLS LAST"
           RECORD COLUMN : NATION.N_NAME
           READ KEY COLUMN : NATION.N_NATIONKEY
           READ RECORD COLUMN : NATION.N_NAME
     3  -  READ COLUMN : NATION.N_NATIONKEY, NATION.N_NAME
<<<  end print plan
```

NO_USE_ORDER_SORT

If NO_USE_ORDER_SORT hint is specified, then an optimizer makes the row to be sorted for sort key colu mn and ascend from the subordinate node without accumulating sort instant.

```
\EXPLAIN PLAN
  SELECT /*+ NO_USE_ORDER_SORT */
        n_nationkey,
        n_name
    FROM nation
ORDER BY n_nationkey;
>>>  start print plan
< Execution Plan >
=============================================================================
|  IDX  |  NODE DESCRIPTION                                                 |
-----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                                 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                      |
|    2  |      INDEX ACCESS ("NATION", "NATION_PK_INDEX")                   |
=============================================================================
     1  -  TARGET : NATION.N_NATIONKEY, NATION.N_NAME
```

```
     2  -  READ INDEX COLUMN : NATION.N_NATIONKEY
           READ TABLE COLUMN : NATION.N_NAME
<<<  end print plan
```

SORT INSTANT does not exist in the execution plan above. Instead, it outputs the result which is sorted fo r n_nationkey by performing INDEX ACCESS.

USE_ORDER_LIMIT_SORT

If USE_ORDER_LIMIT_SORT hint is specified, then an optimizer sorts the results by using limit sort method. In this case, it is applicable only when LIMIT clause and ORDER BY clause are used together.

```
\EXPLAIN PLAN
  SELECT /*+ USE_ORDER_LIMIT_SORT */
         n_nationkey,
         n_name
    FROM nation
ORDER BY n_nationkey
   LIMIT 10,10;
>>>  start print plan
< Execution Plan >
=============================================================================
¦  IDX  ¦  NODE DESCRIPTION                                                  ¦
-----------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                                  ¦
¦    1  ¦    QUERY BLOCK ("$QB_IDX_2")                                     ¦
¦    2  ¦      SORT INSTANT                                                ¦
¦    3  ¦        TABLE ACCESS ("NATION")                                   ¦
=============================================================================
     1  -  TARGET : NATION.N_NATIONKEY, NATION.N_NAME
     2  -  LIMIT SORT
           SORT KEY : "NATION.N_NATIONKEY ASC NULLS LAST"
           RECORD COLUMN : NATION.N_NAME
           READ KEY COLUMN : NATION.N_NATIONKEY
           READ RECORD COLUMN : NATION.N_NAME
     3  -  READ COLUMN : NATION.N_NATIONKEY, NATION.N_NAME
<<<  end print plan
```

SORT INSTANT is performed with LIMIT SORT in the execution plan above.

NO_USE_ORDER_LIMIT_SORT

If NO_USE_ORDER_LIMIT_SORT hint is specified, then an optimizer does not apply limit sort method. The refore, it is processed by creating SORT INSTANT, or making the row to be sorted for sort key column and

to ascend from the subordinate node.

```
\EXPLAIN PLAN
  SELECT /*+ NO_USE_ORDER_LIMIT_SORT */
         n_nationkey,
         n_name
    FROM nation
ORDER BY n_nationkey
   LIMIT 10,10;
>>>  start print plan
< Execution Plan >
=========================================================================
|  IDX  |  NODE DESCRIPTION                                             |
-------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                             |
|    1  |    QUERY BLOCK ("$QB_IDX_2")                                  |
|    2  |      INDEX ACCESS ("NATION", "NATION_PK_INDEX")               |
=========================================================================
     1  -  TARGET : NATION.N_NATIONKEY, NATION.N_NAME
     2  -  READ INDEX COLUMN : NATION.N_NATIONKEY
           READ TABLE COLUMN : NATION.N_NAME
<<<  end print plan
```

LIMIT SORT method is not applied in the execution plan above.

<order driver hints>

Those hints are available when performing *order by* clause in the cluster system. They are ignored in the s tandalone system.

LOCAL_ORDER

It performs *order by* in the current server.

The following is an example of using LOCAL_ORDER hint.

```
\EXPLAIN PLAN
  SELECT /*+ LOCAL_ORDER */
         o_orderdate, o_orderstatus
    FROM orders
   WHERE o_orderdate >= date '1993-07-01'
     AND o_orderdate < date '1993-07-01' + interval '1' month
ORDER BY o_orderdate;
>>>  start print plan
```

```
< Execution Plan >
============================================================================
|  IDX  |  NODE DESCRIPTION                          |              ROWS |
----------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                          |             19319 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")               |             19319 |
|    2  |      SORT INSTANT                          |             19319 |
|    3  |        PLAN BASED CLUSTER                  | LOCAL/REMOTE 19319 |
|    4  |          TABLE ACCESS ("ORDERS")           |              6453 |
============================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     2  -  SORT KEY : "ORDERS.O_ORDERDATE ASC NULLS LAST"
           RECORD COLUMN : ORDERS.O_ORDERSTATUS
           READ KEY COLUMN : ORDERS.O_ORDERDATE
           READ RECORD COLUMN : ORDERS.O_ORDERSTATUS
     3  -  SQL : SELECT /*+ FULL( _A1 ) */
                         "_A1"."O_ORDERSTATUS", "_A1"."O_ORDERDATE"
                    FROM "PUBLIC"."ORDERS"@LOCAL AS "_A1"
                   WHERE "_A1"."O_ORDERDATE" < :_V0
                     AND "_A1"."O_ORDERDATE" >= :_V1
           TARGET DOMAIN : G1(G1N1,G1N2) 6453 rows,
                           G2(G2N1,G2N2) 6450 rows,
                           G3(G3N1,G3N2) 6416 rows
     4  -  HASH SHARD ( # 3 )
           READ COLUMN : ORDERS.O_ORDERSTATUS, ORDERS.O_ORDERDATE
             PHYSICAL FILTER : ORDERS.O_ORDERDATE < DATE'1993-07-01' + CAST( '1' AS
INTERVAL(MONTH) ) AND ORDERS.O_ORDERDATE >= DATE'1993-07-01'
```

REMOTE_ORDER

It performs *order by* in a server of each group.

The following is an example of using REMOTE_ORDER hint.

```
\EXPLAIN PLAN
  SELECT /*+ REMOTE_ORDER */
         o_orderdate, o_orderstatus
    FROM orders
   WHERE o_orderdate >= date '1993-07-01'
     AND o_orderdate < date '1993-07-01' + interval '1' month
ORDER BY o_orderdate;
>>>  start print plan
```

```
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                          |               ROWS |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                          |              19319 |
|    1  |    QUERY BLOCK ("$QB_IDX_2")               |              19319 |
|    2  |      MULTIPLE CLUSTER                      | LOCAL/REMOTE 19319 |
|    3  |        SELECT STATEMENT                    |               6453 |
|    4  |          QUERY BLOCK ("$QB_IDX_2")         |               6453 |
|    5  |            SORT INSTANT                    |               6453 |
|    6  |              TABLE ACCESS ("ORDERS" AS _A1) |               6453 |
================================================================================
     1  -  TARGET : ORDERS.O_ORDERDATE, ORDERS.O_ORDERSTATUS
     2  -  SQL : SELECT /*+ USE_ORDER_SORT
                            FULL( _A1 )
                        */
                        "_A1"."O_ORDERDATE", "_A1"."O_ORDERSTATUS"
                   FROM "PUBLIC"."ORDERS"@LOCAL AS "_A1"
                  WHERE "_A1"."O_ORDERDATE" < :_V0
                    AND "_A1"."O_ORDERDATE" >= :_V1
                ORDER BY "_A1"."O_ORDERDATE" ASC NULLS LAST
           TARGET DOMAIN : G1(G1N1,G1N2) 6453 rows,
                           G2(G2N1,G2N2) 6450 rows,
                           G3(G3N1,G3N2) 6416 rows
           MERGE SORTING
             SORT KEY : ORDERS.O_ORDERDATE
     4  -  TARGET : _A1.O_ORDERDATE, _A1.O_ORDERSTATUS
     5  -  SORT KEY : "_A1.O_ORDERDATE ASC NULLS LAST"
           RECORD COLUMN : _A1.O_ORDERSTATUS
           READ KEY COLUMN : _A1.O_ORDERDATE
           READ RECORD COLUMN : _A1.O_ORDERSTATUS
     6  -  HASH SHARD ( # 3 )
           READ COLUMN : _A1.O_ORDERSTATUS, _A1.O_ORDERDATE
             PHYSICAL FILTER : _A1.O_ORDERDATE < :_V0 AND _A1.O_ORDERDATE >= :_V1
<<<  end print plan
```

〈aggregation hints〉

Those hints are available when performing single-row aggregation.

〈aggregation driver hints〉

Those hints are available when performing single-row aggregation in the cluster system. It is ignored in th e standalone system.

LOCAL_AGGR

It performs aggregation in the current server.

The following is an example of using LOCAL_AGGR hint.

```
\EXPLAIN PLAN
SELECT /*+ LOCAL_AGGR */
       sum(ps_supplycost * ps_availqty) * 0.0001
  FROM partsupp,
       supplier,
       nation
 WHERE ps_suppkey = s_suppkey
   AND s_nationkey = n_nationkey
   AND n_name = 'GERMANY';
>>>  start print plan
< Execution Plan >
================================================================================
¦IDX¦  NODE DESCRIPTION                              ¦               ROWS ¦
--------------------------------------------------------------------------------
¦ 0 ¦  SELECT STATEMENT                              ¦                  1 ¦
¦ 1 ¦    QUERY BLOCK ("$QB_IDX_2")                   ¦                  1 ¦
¦ 2 ¦      AGGREGATION BY HASH                       ¦                  1 ¦
¦ 3 ¦        PLAN BASED CLUSTER                      ¦ LOCAL/REMOTE 31680 ¦
¦ 4 ¦          NESTED JOIN (INNER JOIN)              ¦              10508 ¦
¦ 5 ¦            NESTED JOIN (INNER JOIN)            ¦                396 ¦
¦ 6 ¦              TABLE ACCESS ("NATION")           ¦                  1 ¦
¦ 7 ¦              INDEX ACCESS ("SUPPLIER")         ¦                396 ¦
¦ 8 ¦            INDEX ACCESS ("PARTSUPP")           ¦              10508 ¦
================================================================================
     1  -  TARGET : SUM( PARTSUPP.PS_SUPPLYCOST * PARTSUPP.PS_AVAILQTY ) * 0.0001
     2  -  AGGREGATION : SUM( PARTSUPP.PS_SUPPLYCOST * PARTSUPP.PS_AVAILQTY )
     3  -  SQL : SELECT /*+ KEEP_JOINED_TABLE
                            USE_NL_IN( _A1 )
                            USE_NL_IN( _A2 )
                            FULL( _A3 )
                            INDEX( _A2, "PUBLIC"."SUPPLIER_NATIONKEY_FK" )
                            INDEX( _A1, "PUBLIC"."PARTSUPP_SUPPKEY_FK" )
```

```
                    */
                    "_A1"."PS_SUPPLYCOST", "_A1"."PS_AVAILQTY"
                FROM ( ( "PUBLIC"."NATION"@LOCAL AS "_A3"
                         INNER JOIN
                         "PUBLIC"."SUPPLIER"@LOCAL AS "_A2" ON true
                       ) ALIAS "_A4"
                       INNER JOIN
                       "PUBLIC"."PARTSUPP"@LOCAL AS "_A1" ON true
                    ) ALIAS "_A5"
               WHERE "_A3"."N_NAME" = :_V0
                 AND "_A2"."S_NATIONKEY" = "_A3"."N_NATIONKEY"
                 AND "_A1"."PS_SUPPKEY" = "_A2"."S_SUPPKEY"
        TARGET DOMAIN : G1(G1N1,G1N2) 10508 rows,
                        G2(G2N1,G2N2) 10567 rows,
                        G3(G3N1,G3N2) 10605 rows
    4  -  JOINED COLUMN : PARTSUPP.PS_SUPPLYCOST, PARTSUPP.PS_AVAILQTY
    5  -  JOINED COLUMN : SUPPLIER.S_SUPPKEY
    6  -  CLONED
          READ COLUMN : NATION.N_NATIONKEY, NATION.N_NAME
            PHYSICAL FILTER : NATION.N_NAME = 'GERMANY'
    7  -  CLONED
          READ INDEX COLUMN : SUPPLIER.S_NATIONKEY
          READ TABLE COLUMN : SUPPLIER.S_SUPPKEY
            MIN RANGE : SUPPLIER.S_NATIONKEY = {NATION.N_NATIONKEY}
            MAX RANGE : SUPPLIER.S_NATIONKEY = {NATION.N_NATIONKEY}
    8  -  HASH SHARD ( # 3 )
          READ INDEX COLUMN : PARTSUPP.PS_SUPPKEY
          READ TABLE COLUMN : PARTSUPP.PS_AVAILQTY, PARTSUPP.PS_SUPPLYCOST
            MIN RANGE : PARTSUPP.PS_SUPPKEY = {SUPPLIER.S_SUPPKEY}
            MAX RANGE : PARTSUPP.PS_SUPPKEY = {SUPPLIER.S_SUPPKEY}
<<<  end print plan
```

In the execution plan above, it brings all join results to a local, then performs aggregation.

REMOTE_AGGR

It performs *aggregation* in a server of each group.

The following is an example of using REMOTE_AGGR hint.

```
\EXPLAIN PLAN
SELECT /*+ REMOTE_AGGR */
       sum(ps_supplycost * ps_availqty) * 0.0001
```

```
  FROM partsupp,
       supplier,
       nation
 WHERE ps_suppkey = s_suppkey
   AND s_nationkey = n_nationkey
   AND n_name = 'GERMANY';
>>>  start print plan
< Execution Plan >
================================================================================
| IDX |  NODE DESCRIPTION                                |           ROWS |
--------------------------------------------------------------------------------
|  0 |  SELECT STATEMENT                                |              1 |
|  1 |    QUERY BLOCK ("$QB_IDX_2")                     |              1 |
|  2 |      SINGLE CLUSTER                              | LOCAL/REMOTE 1 |
|  3 |        SELECT STATEMENT                          |              1 |
|  4 |          QUERY BLOCK ("$QB_IDX_2")               |              1 |
|  5 |            AGGREGATION BY HASH                   |              1 |
|  6 |              NESTED JOIN (INNER JOIN)            |          10508 |
|  7 |                NESTED JOIN (INNER JOIN)          |            396 |
|  8 |                  TABLE ACCESS ("NATION" AS _A3)  |              1 |
|  9 |                  INDEX ACCESS ("SUPPLIER" AS _A2)|            396 |
| 10 |                INDEX ACCESS ("PARTSUPP" AS _A1)  |          10508 |
================================================================================
     1  -  TARGET : SUM( PARTSUPP.PS_SUPPLYCOST * PARTSUPP.PS_AVAILQTY ) * 0.0001
     2  -  SQL : SELECT /*+ KEEP_JOINED_TABLE
                            USE_NL_IN( _A1 )
                            USE_NL_IN( _A2 )
                            FULL( _A3 )
                            INDEX( _A2, "PUBLIC"."SUPPLIER_NATIONKEY_FK" )
                            INDEX( _A1, "PUBLIC"."PARTSUPP_SUPPKEY_FK" )
                         */
                         SUM( "_A1"."PS_SUPPLYCOST" * "_A1"."PS_AVAILQTY" )
                    FROM ( ( "PUBLIC"."NATION"@LOCAL AS "_A3"
                             INNER JOIN
                             "PUBLIC"."SUPPLIER"@LOCAL AS "_A2" ON true
                           ) ALIAS "_A4"
                           INNER JOIN
                           "PUBLIC"."PARTSUPP"@LOCAL AS "_A1" ON true
                         ) ALIAS "_A5"
                   WHERE "_A3"."N_NAME" = :_V0
                     AND "_A2"."S_NATIONKEY" = "_A3"."N_NATIONKEY"
```

```
                AND "_A1"."PS_SUPPKEY" = "_A2"."S_SUPPKEY"
        TARGET DOMAIN : G1(G1N1,G1N2) 1 rows,
                        G2(G2N1,G2N2) 1 rows,
                        G3(G3N1,G3N2) 1 rows
        RE-AGGREGATION
          AGGREGATION : SUM( SUM( PARTSUPP.PS_SUPPLYCOST * PARTSUPP.PS_AVAILQTY ) )
 4  -  TARGET : SUM( _A1.PS_SUPPLYCOST * _A1.PS_AVAILQTY )
 5  -  AGGREGATION : SUM( _A1.PS_SUPPLYCOST * _A1.PS_AVAILQTY )
 6  -  JOINED COLUMN : _A1.PS_SUPPLYCOST, _A1.PS_AVAILQTY
          CONSTANT FILTER : TRUE
 7  -  JOINED COLUMN : _A2.S_SUPPKEY
          CONSTANT FILTER : TRUE
 8  -  CLONED
        READ COLUMN : _A3.N_NATIONKEY, _A3.N_NAME
          PHYSICAL FILTER : _A3.N_NAME = :_V0
 9  -  CLONED
        READ INDEX COLUMN : _A2.S_NATIONKEY
        READ TABLE COLUMN : _A2.S_SUPPKEY
          MIN RANGE : _A2.S_NATIONKEY = {_A3.N_NATIONKEY}
          MAX RANGE : _A2.S_NATIONKEY = {_A3.N_NATIONKEY}
10  -  HASH SHARD ( # 3 )
        READ INDEX COLUMN : _A1.PS_SUPPKEY
        READ TABLE COLUMN : _A1.PS_AVAILQTY, _A1.PS_SUPPLYCOST
          MIN RANGE : _A1.PS_SUPPKEY = {_A2.S_SUPPKEY}
          MAX RANGE : _A1.PS_SUPPKEY = {_A2.S_SUPPKEY}
<<<  end print plan
```

In the execution plan above, it performs aggregation in each server, then performs re-aggregation by brin g the result to a local.

15.7 SQL Trace Log

Overview

SQL trace log records the information about performing a user query which can is analyzable. SQL trace log is created in $GOLDILOCKS_DATA/trc directory being separated as independent files such as process ID and a session ID. SQL trace log consists of various information such as a user query, SQL execution plan, and SQL execution time per each SQL process phase, then output.

Output

TRACE_LOG_ID value should be set by using ALTER SESSION statement or ALTER SYSTEM statement to output SQL trace log . For more information about the set value, refer to **TRACE_LOG_ID** of server property.

Trace log can record both successful SQL query and failed SQL query, and it can be set by combining flag values of TRACE_LOG_ID.

SQL query which failed on execution phase of **SQL Processing** is called as a failed SQL query. Therefore, SQL queries failed on a parser, a validator, a rewriter, an enumerator, a code planner, a data planner phases are not recorded on a trace log.

The following is an example of outputting SQL trace log by using TRACE_LOG_ID property.

- It outputs both successful SQL statement and failed SQL statement.

```
gSQL> ALTER SESSION SET TRACE_LOG_ID = 110000;
Session altered.
```

- It outputs both successful SQL statement and the bind value.

```
gSQL> ALTER SYSTEM SET TRACE_LOG_ID = 100010;
System altered.
```

When set TRACE_LOG_ID, which is a property to output SQL trace log, to ALTER SESSION, then it is applied only to the corresponding session and operated in it. If it is set to ALTER SYSTEM, then it is applied to all sessions of all processes connected to the server then operated in them. Therefore, use ALTER SESSIO

N to see the SQL trace log of the current session, and use ALTER SYSTEM to see SQL trace log of other pr ocesses and other sessions.

> ⚠ If the number of processes and sessions connected to the server when it is set to ALTER SYSTE M, then SQL trace log files are created a s many, so be cautious to use it.

SQL trace log file is created under trc directory, and the file name is created as follows.

```
opt_p[process ID]_s[session ID].trc
```

It specifies opt in front of a file name, then identifier p together with process ID, then session ID together with identifier s. The delimiter is_, and extension is trc. If the data created in the same session of the sam e process is bigger than the maximum size of SQL trace log file, then the file name is changed by adding t he current time at the end of the existing file name. Then, it creates a new file with the current file name and keep recording.

The following is an example of SQL trace log file name.

```
opt_p17104_s12.trc
```

Output Format

SQL trace log are divided into <SQL query string>, <Execution plan>, <Execution type>, <Bind param value > and <Time info>.

SQL Query String

It outputs the query input by a user including the current time, whether it is successful and the query proc essing time, and the output format is as follows.

```
[current time] [whether it is successful][query processing time] SQL statement
```

The date and time in us unit is input in [current time], and if it is successful S is input in [*whether it is succ essful*], if it is failed, then F is output. The query processing time is output in us unit, and SQL statement is input by a user.

The query processing time is measure in 10 ms when **TRACE_LOG_TIME_DETAIL** property is not set to *O N*. However, if this property is set to *ON*, then the query processing performance may be degraded.

Execution Plan

It outputs the execution plan of SQL statement. The form is similar to **Execution Plan**, and the total time c olumn is additionally output on the execution plan node table. The total time output on the statement is the time of processing the entire query, and the total time output on other nodes is the processing time o n each node. In this case, the total time is output in 10 ms unit.

Use **TRACE_LOG_TIME_DETAIL** property to output the detailed time. However, if this property is set to *O N*, then the then the query processing performance may be degraded.

Execution Type

When it directly performs the query by executing SQL statement, then it outputs *DIRECT EXECUTE*. When it performs the query by using prepare, it outputs *PREPARE EXECUTE*.

Bind Param Value

When a bind param value is used in SQL statement, then it outputs the information about the bind para m value. When a bind param value is not used in SQL statement, it outputs *No Bind Param*.

Time Info

It outputs the execution time per each phase of SQL processing. Time info is output being divided into m odule, time, rate and call. Module outputs the phase name such as parse and validate, and time output t he actual execution time, and rate outputs the ratio of execution time of each phase to the total executio n time. Call outputs how may times each phase is called.

Module consists of 7 phases such as parse, validate, code opt, optimizer, data opt, execute, fetch and tot al phase. A query is parsed on parse phase, and the parsed query is validated on phase. it is preprocessed to perform SQL optimizer on code opt phase, and SQL optimizer is actually executed on optimizer phase. It is prepared to execute the SQL execution plan on data opt phase, then SQL execution plan is executed on execute phase. Query results are collected like as SELECT statement and the result is returned on fetch phase.

When a plan cache is used, validate, code opt, optimizer may not be called. Time is output in 10 ms unit, so the execution time shorter than 10 ms is output as 0. Also, rate is the ratio of execution time of each p hase to the total execution time, so the total is 100%, and the ratio of dividing the execution time of eac h phase to total is output. In this case, of the execution time of each phase is 0, then it is output as 0%.

Use **TRACE_LOG_TIME_DETAIL** property to output the detailed time. However, if this property is set to *O N*, then the then the query processing performance may be degraded.

Examples

The following is SQL statement which does not have a bind param value.

```
SELECT O_TOTALPRICE, O_ORDERDATE, L_QUANTITY
  FROM ORDERS, LINEITEM
 WHERE O_ORDERKEY = L_ORDERKEY
   AND O_ORDERDATE >= DATE '1996-01-01'
   AND L_SHIPMODE = 'AIR';
```

The following SQL trace log is output when executing the SQL statement above after setting TRACE_LOG _ID to 101111.

```
[2017-05-25 12:27:59.199657] [S][0.000000] SELECT O_TOTALPRICE, O_ORDERDATE, L_QUANTITY
  FROM ORDERS, LINEITEM
 WHERE O_ORDERKEY = L_ORDERKEY
   AND O_ORDERDATE >= DATE '1996-01-01'
   AND L_SHIPMODE = 'AIR'
< Execution Plan >
================================================================================
¦ IDX ¦  NODE DESCRIPTION                              ¦ ROWS ¦ Total Time ¦
--------------------------------------------------------------------------------
¦  0  ¦  SELECT STATEMENT                              ¦      ¦ 0:00:00.00 ¦
¦  1  ¦    NESTED LOOP JOIN (INNER JOIN)              ¦    1 ¦ 0:00:00.00 ¦
¦  2  ¦      TABLE ACCESS ("LINEITEM")                ¦    2 ¦ 0:00:00.00 ¦
¦  3  ¦      INDEX ACCESS ("ORDERS, ORDERS_PK_INDEX") ¦    1 ¦ 0:00:00.00 ¦
================================================================================
     1  -  JOINED COLUMNS : ORDERS.O_TOTALPRICE, ORDERS.O_ORDERDATE, LINEITEM.L_QUANTITY
     2  -  READ COLUMNS : L_ORDERKEY, L_QUANTITY, L_SHIPMODE
             PHYSICAL FILTER : L_SHIPMODE = 'AIR'
     3  -  READ INDEX COLUMNS : O_ORDERKEY
           READ TABLE COLUMNS : O_TOTALPRICE, O_ORDERDATE
             MIN RANGE : O_ORDERKEY = {L_ORDERKEY}
             MAX RANGE : O_ORDERKEY = {L_ORDERKEY}
             PHYSICAL TABLE FILTER : O_ORDERDATE >= CAST( '1996-01-01' AS DATE )
< Execution Type >
----------------------
  DIRECT EXECUTE
< Bind Param Value >
--------------------------
```

```
  No Bind Param.
< Time Info >
=========================================
| Module    | Time       | Rate     | Call |
-----------------------------------------
| Parse     | 0:00:00.00 |   0.00 % |    1 |
| Validate  | 0:00:00.00 |   0.00 % |    1 |
| Code Opt  | 0:00:00.00 |   0.00 % |    1 |
| Optimizer | 0:00:00.00 |   0.00 % |    1 |
| Data Opt  | 0:00:00.00 |   0.00 % |    1 |
| Execute   | 0:00:00.00 |   0.00 % |    1 |
| Fetch     | 0:00:00.00 |   0.00 % |    1 |
| Total     | 0:00:00.00 | 100.00 % |      |
=========================================
```

The following is SQL statement which has a bind param value.

```
SELECT L_QUANTITY
  FROM LINEITEM
 WHERE L_SHIPMODE = :V1;
```

The following SQL trace log is output when executing the SQL statement above after setting TRACE_LOG _ID to 101111.

```
[2017-05-25 12:27:59.200204] [S][0.000000] SELECT L_QUANTITY
  FROM LINEITEM
 WHERE L_SHIPMODE = :V1
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                      |       ROWS | Total Time |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                      |            | 0:00:00.00 |
|    1  |    TABLE ACCESS ("LINEITEM")           |          0 | 0:00:00.00 |
================================================================================
     1  -  READ COLUMNS : L_QUANTITY, L_SHIPMODE
             PHYSICAL FILTER : L_SHIPMODE = :V1
< Execution Type >
----------------------
  DIRECT EXECUTE
< Bind Param Value >
--------------------------
   1 - :V1(IN, "AIR")
```

```
< Time Info >
==========================================
| Module    | Time       | Rate     | Call |
------------------------------------------
| Parse     | 0:00:00.00 |   0.00 % |    1 |
| Validate  | 0:00:00.00 |   0.00 % |    1 |
| Code Opt  | 0:00:00.00 |   0.00 % |    1 |
| Optimizer | 0:00:00.00 |   0.00 % |    1 |
| Data Opt  | 0:00:00.00 |   0.00 % |    1 |
| Execute   | 0:00:00.00 |   0.00 % |    1 |
| Fetch     | 0:00:00.00 |   0.00 % |    1 |
| Total     | 0:00:00.00 | 100.00 % |      |
==========================================
```

16.

Built-in Data Type References

16.1 Aliases of Built-in Data Types

- BIGINT
  - It is as same as NUMBER(19,0).
  - Refer to **NUMBER**.
- BINARY
  - Refer to **BINARY**.
- BINARY VARYING
  - Refer to **BINARY VARYING** .
- BINARY LONG VARYING
  - Refer to **BINARY LONG VARYING** .
- BOOLEAN
  - Refer to **BOOLEAN**.
- CHAR
  - It is as same as CHARACTER.
  - Refer to **CHARACTER**.
- CHARACTER
  - Refer to **CHARACTER**.
- CHARACTER VARYING
  - Refer to **CHARACTER VARYING**.
- CHARACTER LONG VARYING
  - Refer to **CHARACTER LONG VARYING**.
- DATE
  - Refer to **DATE**.
- DEC
  - It is as same as NUMERIC.
  - Refer to **NUMERIC**.
- DECIMAL
  - It is as same as NUMERIC.
  - Refer to **NUMERIC**.
- DOUBLE
  - It is as same as FLOAT(53).
  - Refer to **FLOAT**.
- DOUBLE PRECISION
  - It is as same as FLOAT(53).
  - Refer to **FLOAT**.
- FLOAT
  - Refer to **FLOAT**.
- FLOAT4
  - It is as same as FLOAT(24).

    - Refer to **FLOAT**.
- FLOAT8
    - It is as same as FLOAT(53).
    - Refer to **FLOAT**.
- INT
    - It is as same as NUMBER(10,0).
    - Refer to **NUMBER**.
- INT2
    - It is as same as NUMBER(5,0).
    - Refer to **NUMBER**.
- INT4
    - It is as same as NUMBER(10,0).
    - Refer to **NUMBER**.
- INT8
    - It is as same as NUMBER(19,0).
    - Refer to **NUMBER**.
- INTEGER
    - It is as same as NUMBER(10,0).
    - Refer to **NUMBER**.
- INTERVAL
    - Refer to **INTERVAL**.
- LONG BINARY VARYING
    - It is as same as BINARY LONG VARYING.
    - Refer to **BINARY LONG VARYING**.
- LONG CHAR VARYING
    - It is as same as CHARACTER LONG VARYING.
    - Refer to **CHARACTER LONG VARYING**.
- LONG CHARACTER VARYING
    - It is as same as CHARACTER LONG VARYING.
    - Refer to **CHARACTER LONG VARYING**.
- LONG VARCHAR
    - It is as same as CHARACTER LONG VARYING.
    - Refer to **CHARACTER LONG VARYING**.
- NATIVE_BIGINT
    - Refer to **NATIVE_BIGINT**.
- NATIVE_DOUBLE
    - Refer to **NATIVE_DOUBLE**.
- NATIVE_INTEGER
    - Refer to **NATIVE_INTEGER**.
- NATIVE_REAL
    - Refer to **NATIVE_REAL**.

- NATIVE_SMALLINT
  - Refer to **NATIVE_SMALLINT**.
- NUMBER
  - Refer to **NUMBER**.
- NUMERIC
  - Refer to **NUMERIC**.
- ROWID
  - Refer to **ROWID**.
- SMALLINT
  - It is as same as NUMBER(5,0).
  - Refer to **NUMBER**.
- TIME
  - Refer to **TIME**.
- TIMESTAMP
  - Refer to **TIMESTAMP**.
- VARBINARY
  - It is as same as BINARY VARYING.
  - Refer to **BINARY VARYING**.
- VARCHAR
  - It is as same as CHARACTER VARYING.
  - Refer to **CHARACTER VARYING**.
- VARCHAR2
  - It is as same as CHARACTER VARYING.
  - Refer to **CHARACTER VARYING**.

16.2 BINARY

Syntax

```
BINARY [ (length) ]
```

Syntax Rules and Parameters

- length: It is the binary string length.
  - Range: 1 ~ 2000
  - Default value: 1

Description

A fixed-length binary string is stored.
If the binary string length to be stored is shorter than the specified length, X'00 ' is stored in the remaini ng part.

- Storage size: Bytes of the length value

For More Information

Refer to the followings.

- **BINARY VARYING**
- **BINARY LONG VARYING**

16.3 BINARY VARYING

Syntax

```
BINARY VARYING (length)
```

Syntax Rules and Parameters

- length: It is the maximum length of binary string.
  - Range: 1 ~ 4000

Description

The variable-length binary string is stored.

- Storage size: Bytes of the binary string to be stored
- Alias names: VARBINARY

For More Information

Refer to the followings.

- **BINARY**
- **BINARY LONG VARYING**

16.4 BINARY LONG VARYING

Syntax

```
BINARY LONG VARYING
```

Description

The value of the long variable binary string is stored.

- Maximum storage size: 100 megabytes
- Storage size: Bytes of the binary string to be stored
- Alias names: LONG BINARY VARYING, LONG VARBINARY

It can not be used as a column of the key, so there are limitations as follows.

- It can not be used as a key column of an index.
- It can not be used as the expression of ORDER BY clause.
- It can not be used as the expression of GROUP BY clause.
- It can not be used as the expression of DISTINCT clause.
- It can not be used as the expression of UNION, INTERSECT, EXCEPT clauses.

For More Information

Refer to the followings.

- **BINARY**
- **BINARY VARYING**

16.5 BOOLEAN

Syntax

```
BOOLEAN
```

Description

TRUE or FALSE is stored.

- Storage size: 1 byte.

16.6 CHARACTER

Syntax

```
CHARACTER [ (length [ CHARACTERS ¦ OCTETS ¦ CHAR ¦ BYTE ] ) ]
```

Syntax Rules and Parameters

- length: It is the string length.
  - Range: 1 ~ 2000
  - Default value: 1

- [CHARACTERS | OCTETS | CHAR | BYTE]: It is unit of length.
  - CHARACTERS
    - The number of characters
    - CHAR is as same as CHARACTERS.
  - OCTETS
    - The number of bytes
    - BYTE is as same as OCTETS.
  - If omitted, the default is the property value of **CHAR_LENGTH_UNITS** which is set when creating database.

Description

A fixed-length string is stored.
If the length of the string to be stored is shorter than the specified length, white spaces are stored in the r emaining part.

- Storage size: Bytes of the length value
- Alias name: CHAR

For More Information

Refer to the followings.

- CHARACTER VARYING
- CHARACTER LONG VARYING

16.7 CHARACTER VARYING

Syntax

```
CHARACTER VARYING ( length [ CHARACTERS ¦ OCTETS ¦ CHAR ¦ BYTE ] )
```

Syntax Rules and Parameters

- length: It is the string length.
  - Range: 1 ~ 4000

- [CHARACTERS | OCTETS | CHAR | BYTE]: It is unit of length.
  - CHARACTERS
    - The number of characters
    - CHAR is as same as CHARACTERS.
  - OCTETS
    - The number of bytes
    - BYTE is as same as OCTETS.
  - If omitted, the default is the property value of **CHAR_LENGTH_UNITS** which is set when creating database.

Description

The variable-length string is stored.

- Storage size: Bytes of the string to be stored
- Alias names: VARCHAR, VARCHAR2

For More Information

Refer to the followings.

- **CHARACTER**
- **CHARACTER LONG VARYING**

16.8 CHARACTER LONG VARYING

Syntax

```
CHARACTER LONG VARYING
```

Description

The value of the long variable-length string is stored.

- Maximum storage size: 100 megabytes
- Storage size: Bytes of the string to be stored
- Alias names: LONG CHARACTER VARYING, LONG CHAR VARYING, LONG VARCHAR

It can not be used as a column of the key, so there are limitations as follows.

- It can not be used as a key column of an index.
- It can not be used as the expression of ORDER BY clause.
- It can not be used as the expression of GROUP BY clause.
- It can not be used as the expression of DISTINCT clause.
- It can not be used as the expression of UNION, INTERSECT, EXCEPT clauses.

For More Information

Refer to the followings.

- CHARACTER
- CHARACTER VARYING

16.9 DATE

Syntax

```
DATE
```

Description

It is the date type including YEAR, MONTH, DAY, HOUR, MINUTE and SECOND (excluding fractional seconds).

- Value range: Date value between '4714-11-24 BC' and '9999-12-31 AD'
- Storage size: 8 bytes

For More Information

Refer to the followings.

- **Date Literals**
- **TIME**
- **TIMESTAMP**

16.10 FLOAT

Syntax

```
FLOAT[ ( precision ) ]
```

Syntax Rules and Parameters

- precision: It is the binary precision of significant digits.
  - Precision range: 1 ~ 126
  - Default value: 126

Description

The floating point value with a binary precision is stored.

- Exponential range: 1E-130 ~ 1E+125
- Storage size: (number of digits in integer part+ 1) / 2 + (number of digits in fractional part + 1) / 2 + 1(exponent, sign)

It has a binary precision value unlike **NUMBER**, **NUMERIC** types.

- Alias names: REAL = FLOAT(24), DOUBLE = FLOAT(53), DOUBLE PRECISION = FLOAT(53), FLOAT4 = FLOAT(24), FLOAT8 = FLOAT(53).

For More Information

Refer to the followings.

- **NUMBER**
- **NUMERIC**

16.11 INTERVAL

Syntax

```
<interval_type> ::=
      INTERVAL YEAR [ ( leading_precision ) ]
    ¦ INTERVAL MONTH [ ( leading_precision ) ]
    ¦ INTERVAL DAY [ ( leading_precision ) ]
    ¦ INTERVAL HOUR [ ( leading_precision ) ]
    ¦ INTERVAL MINUTE [ ( leading_precision ) ]
    ¦ INTERVAL SECOND [ ( leading_precision  [ , fractional_seconds_precision ] ) ]
    ¦ INTERVAL YEAR [ ( leading_precision ) ] TO MONTH
    ¦ INTERVAL DAY [ ( leading_precision ) ] TO HOUR
    ¦ INTERVAL DAY [ ( leading_precision ) ] TO MINUTE
    ¦ INTERVAL DAY [ ( leading_precision ) ] TO SECOND [ ( fractional_seconds_precision ) ]
    ¦ INTERVAL HOUR [ ( leading_precision ) ] TO MINUTE
    ¦ INTERVAL HOUR [ ( leading_precision ) ] TO SECOND [ ( fractional_seconds_precision ) ]
    ¦ INTERVAL MINUTE [ ( leading_precision ) ] TO SECOND [ ( fractional_seconds_precision ) ]
```

Syntax Rules and Parameters

- INTERVAL YEAR [(leading_precision)]: A period of YEAR is stored.
  - leading_precision
    - The number of digits in YEAR
    - Value range: 2 ~ 6
    - Default value: 2

- INTERVAL MONTH [(leading_precision)]: A period of MONTH is stored.
  - leading_precision
    - The number of digits in MONTH
    - Value range: 2 ~ 6
    - Default value: 2

- INTERVAL YEAR [(leading_precision)] TO MONTH: A period of YEAR and MONTH is stored.
  - leading_precision
    - The number of digits in YEAR
    - Value range: 2 ~ 6

        - Default value: 2

- INTERVAL DAY [(leading_precision)]: A period of DAY is stored.
    - leading_precision
        - The number of digits in DAY
        - Value range: 2 ~ 6
        - Default value: 2

- INTERVAL HOUR [(leading_precision)]: A period of HOUR is stored.
    - leading_precision
        - The number of digits in HOUR
        - Value range: 2 ~ 6
        - Default value: 2

- INTERVAL MINUTE [(leading_precision)]: A period of MINUTE is stored.
    - leading_precision
        - The number of digits in MINUTE
        - Value range: 2 ~ 6
        - Default value: 2

- INTERVAL SECOND [(leading_precision [, fractional_seconds_precision])]: A period of SECOND is stored.
    - leading_precision
        - The number of digits in SECOND
        - Value range: 2 ~ 6
        - Default value: 2
    - fractional_seconds_precision
        - The number of digits in fractional seconds
        - Value range: 0 ~ 6
        - Default value: 6

- INTERVAL DAY [(leading_precision)] TO HOUR: A period of DAY and HOUR is stored.
    - leading_precision
        - The number of digits in DAY
        - Value range: 2 ~ 6
        - Default value: 2

- INTERVAL DAY [(leading_precision)] TO MINUTE: A period of DAY, HOUR, and MINUTE is stored.
    - leading_precision
        - The number of digits in DAY
        - Value range: 2 ~ 6
        - Default value: 2

- INTERVAL DAY [(leading_precision)] TO SECOND [(fractional_seconds_precision)]: A period of DAY, HOUR, MINUTE, and SECOND is stored.
  - leading_precision
    - The number of digits in DAY
    - Value range: 2 ~ 6
    - Default value: 2
  - fractional_seconds_precision
    - The number of digits in fractional seconds
    - Value range: 0 ~ 6
    - Default value: 6

- INTERVAL HOUR [(leading_precision)] TO MINUTE: A period of HOUR and MINUTE is stored.
  - leading_precision
    - The number of digits in HOUR
    - Value range: 2 ~ 6
    - Default value: 2

- INTERVAL HOUR [(leading_precision)] TO SECOND [(fractional_seconds_precision)]: A period of HOUR, MINUTE and SECOND is stored.
  - leading_precision
    - The number of digits in HOUR
    - Value range: 2 ~ 6
    - Default value: 2
  - fractional_seconds_precision
    - The number of digits in fractional seconds
    - Value range: 0 ~ 6
    - Default value: 6

- INTERVAL MINUTE [(leading_precision)] TO SECOND [(fractional_seconds_precision)]: A period of MINUTE and SECOND is stored.
  - leading_precision
    - The number of digits in MINUTE
    - Value range: 2 ~ 6
    - Default value: 2
  - fractional_seconds_precision
    - The number of digits in fractional seconds
    - Value range: 0 ~ 6
    - Default value: 6

Description

INTERVAL types are classified into YEAR TO MONTH family type and DAY TO SECOND family type depen ding on the range of value representation as follows.

- YEAR TO MONTH family type: Its storage size is 8 bytes.
  - INTERVAL YEAR
  - INTERVAL MONTH
  - INTERVAL YEAR TO MONTH

- DAY TO SECOND family type: Its storage size is 16 bytes.
  - INTERVAL DAY
  - INTERVAL HOUR
  - INTERVAL MINUTE
  - INTERVAL SECOND
  - INTERVAL DAY TO HOUR
  - INTERVAL DAY TO MINUTE
  - INTERVAL DAY TO SECOND
  - INTERVAL HOUR TO MINUTE
  - INTERVAL HOUR TO SECOND
  - INTERVAL MINUTE TO SECOND

If the number which is bigger than number of the specified digits is in the field to which the leading_prec ision is specified, then an error is returned.
If the number which is bigger than number of the specified digits is in the field to which the fractional_se conds_precision is specified, it is rounded off.

Table 16-1 Precisions and value range of the second or later field in INTERVAL * TO *

Field	Precision	Value range
MONTH	2	0 ~ 11
HOUR	2	0 ~ 23
MINUTE	2	0 ~ 59
SECOND (interger part)	2	0 ~ 59

For More Information

Refer to **Interval Literals**.

16.12 NATIVE_BIGINT

Syntax

```
NATIVE_BIGINT
```

Description

Signed 8-byte integer is stored.

It is as same as long long data type of C language (8 bytes integer).

- Value range: -9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807
- Storage size: 8 bytes

16.13 NATIVE_DOUBLE

Syntax

```
NATIVE_DOUBLE
```

Description

Double precision floating-point number (8 bytes) is stored.

It is as same as double data type of C language.

- Exponential range: 1E-307 ~ 1E+308
- Storage size: 8 bytes

16.14 NATIVE_INTEGER

Syntax

```
NATIVE_INTEGER
```

Description

Signed 4 byte integer is stored.

It is as same as integer data type of C language (4 bytes).

- Value range: -2,147,483,648 ~ +2,147,483,647
- Storage size: 4 bytes

16.15 NATIVE_REAL

Syntax

```
NATIVE_REAL
```

Description

Single precision floating-point number (4 bytes) is stored.

It is as same as float data type of C language.

- Exponential range: 1E-37 ~ 1E+37
- Storage size: 4 bytes

16.16 NATIVE_SMALLINT

Syntax

```
NATIVE_SMALLINT
```

Description

Signed 2 byte integer is stored.

It is as same as short data type of C language.

- Value range: -32,768 ~ 32,767
- Storage size: 2 bytes

16.17 NUMBER

Syntax

```
NUMBER  [ ( precision [ , scale ] ) ]
```

Syntax Rules and Parameters

- NUMBER: A floating point number without a precision and scale is stored.
  - The range of significant digits: 38
  - Exponential range: 1E-130 ~ 1E+125
  - It is as same as FLOAT (126).

- NUMBER(precision): The integer with significant digits of precision is stored.
  - Precision range: 1 ~ 38
  - The scale value: 0
  - It has a decimal precision value unlike **FLOAT** type.
  - It is as same as NUMBER(precision, 0), NUMERIC(precision, 0).

- NUMBER(precision, scale): A fixed point number with precision and scale is stored.
  - Precision range: 1 ~ 38
  - Scale range: -84 ~ 127
  - It has a decimal precision value unlike **FLOAT** type.
  - It is as same as NUMERIC (precision,scale).
  - Alias names: SMALLINT = NUMBER(5,0), INTEGER = NUMBER(10,0), BIGINT = NUMBER(19,0), I NT2 = NUMBER(5,0), INT4 = NUMBER(10,0), INT8 = NUMBER(19,0)

Description

NUMBER type is similar to NUMERIC type, but if both the precision and scale are omitted, NUMBER type s tores the floating point number whose precision and scale are not specified.

- NUMBER without precision, scale: Floating-point number
- NUMERIC without precision, scale: The fixed point number of NUMERIC(38, 0)
- Storage size: (number of digits in integer part + 1) / 2 + (number of digits in fractional part + 1) / 2 + 1(exponent, sign)

For More Information

Refer to the followings.

- FLOAT
- NUMERIC

16.18 NUMERIC

Syntax

```
NUMERIC  [ ( precision [ , scale ] ) ]
```

Syntax Rules and Parameters

- precision: It is the decimal precision of significant digits.
  - precision range: 1 ~ 38
  - Default value: 38

- scale: It is the decimal point range.
  - scale range: -84 ~ 127
  - Default value: 0

Description

A fixed point number with precision and scale is stored.

If the precision and scale are omitted, it means as follows.

- NUMERIC = NUMERIC(38,0)
- NUMERIC(p) = NUMERIC(p,0)

NUMBER type is similar to NUMERIC type, but if both the precision and scale are omitted, NUMBER type s tores the floating-point number whose precision and scale are not specified.

- NUMBER without precision, scale: Floating point number
- NUMERIC without precision, scale: Fixed point number of NUMERIC(38,0)
- Storage size: (number of digits in integer part + 1) / 2 + (number of digits in fractional part + 1) / 2 + 1(exponent, sign)

For More Information

Refer to the followings.

- FLOAT
- NUMBER

16.19 ROWID

Syntax

```
ROWID
```

Description

A record identifier (ROWID) is stored.
A record identifier (ROWID) is the identification information of each record in database.
When querying the ROWID pseudo column, each record identifier (ROWID) is obtained. This ROWID pseu do column has the ROWID data type information.

ROWID type consists of the followings in a stand-alone system.
- OBJECT_ID
- TABLESPACE_ID
- PAGE_ID
- OFFSET within PAGE

ROWID type consists of the followings in a cluster system.
- GRID_BLOCK_SEQUENCE
- GRID_BLOCK_ID
- MEMBER_ID
- SHARD_ID

ROWID is stored in the base 64 value, which can include A ~ Z, a ~ z, 0 ~ 9, +, /.
Each component information of ROWID is obtained using ROWID-related functions.

- Storage size: 16 bytes

For More Information

Refer to the followings.

- **ROWID Pseudo Column**
- **ROWID-related Functions**

16.20 TIME

Syntax

```
TIME [ ( fractional_seconds_precision ) ] [ WITH TIME ZONE ¦ WITHOUT TIME ZONE ]
```

Syntax Rules and Parameters

- fractional_seconds_precision: It is the number of significant digits in fractional seconds.
  - fractional_seconds_precision range: 0 ~ 6
  - Default value: 6

- [WITH TIME ZONE | WITHOUT TIME ZONE]: It specifies whether to store TIME ZONE value.
  - WITH TIME ZONE: The time which includes time zone
  - WITHOUT TIME ZONE: The time which does not include time zone
  - Default value: WITHOUT TIME ZONE

Description

The time which includes HOUR, MINUTE and SECOND is stored.

- Storage size
  - TIME WITHOUT TIME ZONE: 8 bytes
  - TIME WITH TIME ZONE: 12 bytes

For More Information

Refer to the followings.

- **Time Literals**
- **Time with Time Zone Literals**
- **DATE**
- **TIMESTAMP**

16.21 TIMESTAMP

Syntax

```
TIMESTAMP [ ( fractional_seconds_precision ) ] [ WITH TIME ZONE ¦ WITHOUT TIME ZONE ]
```

Syntax Rules and Parameters

- fractional_seconds_precision: It is the number of significant digits in the fractional seconds.
  - fractional_seconds_precision range: 0 ~ 6
  - Default value: 6

- [WITH TIME ZONE | WITHOUT TIME ZONE]: It specifies whether to store TIME ZONE value.
  - WITH TIME ZONE: The time which includes time zone.
  - WITHOUT TIME ZONE: The time which does not include time zone.
  - Default value: WITHOUT TIME ZONE

Description

The time which includes YEAR, MONTH, DATE, HOUR, MINUTE and SECOND is stored.

- Storage size
  - TIMESTAMP WITHOUT TIME ZONE: 8 bytes
  - TIMESTAMP WITH TIME ZONE: 12 bytes

For More Information

Refer to the followings.

- **Timestamp Literals**
- **Timestamp with Time Zone Literals**
- **DATE**
- **TIME**

17.

Built-in Function References

17.1 * (MULTIPLICATION)

Syntax

```
expr1 * expr2
```

Description

It returns the multiplication result of expr1 and expr2.

The multiplication types and result types are as follows.
For more information, refer to **Type Conversion**.

Table 17-1 Numeric * operation

expr1 (expr2)	expr2 (expr1)	Result type
NATIVE INTEGER family • NATIVE_SMALLINT • NATIVE_INTEGER • NATIVE_BIGINT	NATIVE INTEGER family • NATIVE_SMALLINT • NATIVE_INTEGER • NATIVE_BIGINT	NATIVE_BIGINT
NUMBER	NUMBER	NUMBER
NATIVE DOUBLE family • NATIVE_REAL • NATIVE_DOUBLE	NATIVE DOUBLE family • NATIVE_REAL • NATIVE_DOUBLE	NATIVE_DOUBLE

Table 17-2 INTERVAL * operation

expr1 (expr2)	expr2 (expr1)	Result type
INTERVAL YEAR TO MONTH	Numeric type	INTERVAL YEAR TO MONTH (The result type is the interval type.)
INTERVAL DAY TO SECOND	Numeric type	INTERVAL DAY TO SECOND (The result type is the interval type.)
Refer to **INTERVAL type details which is included in INTERVAL type written in the following table**.		

Table 17-3 INTERVAL type details which is included in INTERVAL type written in the following table

INTERVAL YEAR TO MONTH	INTERVAL DAY TO SECOND
	• INTERVAL DAY • INTERVAL HOUR • INTERVAL MINUTE

INTERVAL YEAR TO MONTH	INTERVAL DAY TO SECOND
• INTERVAL YEAR • INTERVAL MONTH • INTERVAL YEAR TO MONTH	• INTERVAL SECOND • INTERVAL DAY TO HOUR • INTERVAL DAY TO MINUTE • INTERVAL DAY TO SECOND • INTERVAL HOUR TO MINUTE • INTERVAL HOUR TO SECOND • INTERVAL MINUTE TO SECOND

Example

```
gSQL> SELECT INTERVAL'1-2'YEAR TO MONTH * 2 AS RESULT FROM DUAL;
RESULT
----------
+000002-04
1 row selected.
gSQL> SELECT INTERVAL'1 01:02:03.400000'DAY TO SECOND * 2 AS RESULT
      FROM DUAL;
RESULT
-----------------------
+000002 02:04:06.800000
1 row selected.
```

17.2 + (ADDITION)

Syntax

```
expr1 + expr2
```

Description

It returns the addition result of expr1 and expr2.

The addition types and result types are as follows.
For more information, refer to **Type Conversion**.

Table 17-4 Numeric + operation

expr1 (expr2)	expr2 (expr1)	Result type
NATIVE INTEGER family • NATIVE_SMALLINT • NATIVE_INTEGER • NATIVE_BIGINT	NATIVE INTEGER family • NATIVE_SMALLINT • NATIVE_INTEGER • NATIVE_BIGINT	NATIVE_BIGINT
NUMBER	NUMBER	NUMBER
NATIVE DOUBLE family • NATIVE_REAL • NATIVE_DOUBLE	NATIVE DOUBLE family • NATIVE_REAL • NATIVE_DOUBLE	NATIVE_DOUBLE

Table 17-5 (DATETIME/INTERVAL) + operation

expr1 (expr2)	expr2 (expr1)	Result type
DATE	NUMERIC	DATE
DATE	INTERVAL YEAR TO MONTH	DATE
DATE	INTERVAL DAY	DATE
DATE	INTERVAL DAY TO SECOND	TIMESTAMP
TIME	NUMERIC	TIME
TIME	INTERVAL YEAR TO MONTH	TIME
TIME	INTERVAL DAY TO SECOND	TIME
TIME WITH TIME ZONE	NUMERIC	TIME WITH TIME ZONE
TIME WITH TIME ZONE	INTERVAL YEAR TO MONTH	TIME WITH TIME ZONE
TIME WITH TIME ZONE	INTERVAL DAY TO SECOND	TIME WITH TIME ZONE
TIMESTAMP	NUMERIC	TIMESTAMP

expr1 (expr2)	expr2 (expr1)	Result type
TIMESTAMP	INTERVAL YEAR TO MONTH	TIMESTAMP
TIMESTAMP	INTERVAL DAY TO SECOND	TIMESTAMP
TIMESTAMP WITH TIME ZONE	NUMERIC	TIMESTAMP WITH TIME ZONE
TIMESTAMP WITH TIME ZONE	INTERVAL YEAR TO MONTH	TIMESTAMP WITH TIME ZONE
TIMESTAMP WITH TIME ZONE	INTERVAL DAY TO SECOND	TIMESTAMP WITH TIME ZONE
INTERVAL YEAR TO MONTH	INTERVAL YEAR TO MONTH	INTERVAL YEAR TO MONTH (The result type includes all the interval range of expr1 and expr2.)
INTERVAL DAY TO SECOND	INTERVAL DAY TO SECOND	INTERVAL DAY TO SECOND (The result type includes all the interval range of expr1 and expr2.)
Refer to **INTERVAL type details which is included in INTERVAL type written in the following table**.		

Example

```
gSQL> SELECT TO_DATE( '2012-05-05', 'YYYY-MM-DD' ) + 5 AS RESULT FROM DUAL;
RESULT
----------
2012-05-10
1 row selected.
gSQL> SELECT
      TO_DATE( '2012-05-05', 'YYYY-MM-DD' ) + INTERVAL'01-01'YEAR TO MONTH
      AS RESULT
      FROM DUAL;
RESULT
----------
2013-06-05
1 row selected.
gSQL> SELECT
      INTERVAL'01-01'YEAR TO MONTH + INTERVAL'02-10'YEAR TO MONTH
      AS RESULT
      FROM DUAL;
RESULT
----------
+000003-11
1 row selected.
```

17.3 + (POSITIVE)

Syntax

```
+ expr
```

Description

The + sign is displayed in expr.

Example

```
gSQL> SELECT +3 AS RESULT1, +(-3) AS RESULT2 FROM DUAL;
RESULT1 RESULT2
------- -------
      3      -3
1 row selected.
```

17.4 - (NEGATIVE)

Syntax

```
- expr
```

Description

The - sign is displayed in expr.

Example

```
gSQL> SELECT -3 AS RESULT1, -(-3) AS RESULT2 FROM DUAL;
RESULT1 RESULT2
------- -------
     -3       3
1 row selected.
```

17.5 - (SUBTRACTION)

Syntax

```
expr1 - expr2
```

Description

It returns the subtraction result of expr1 and expr2.

The subtraction types and result types are as follows.
For more information, refer to **Type Conversion**.

Table 17-6 Numeric - operation

expr1	expr2	Result type
NATIVE INTEGER family • NATIVE_SMALLINT • NATIVE_INTEGER • NATIVE_BIGINT	NATIVE INTEGER family • NATIVE_SMALLINT • NATIVE_INTEGER • NATIVE_BIGINT	NATIVE_BIGINT
NUMBER	NUMBER	NUMBER
NATIVE DOUBLE family • NATIVE_REAL • NATIVE_DOUBLE	NATIVE DOUBLE family • NATIVE_REAL • NATIVE_DOUBLE	NATIVE_DOUBLE

Table 17-7 (DATETIME/INTERVAL) - operation

expr1	expr2	Result type
DATE	DATE	NUMBER
DATE	Numeric type	DATE
DATE	INTERVAL YEAR TO MONTH	DATE
DATE	INTERVAL DAY	DATE
DATE	INTERVAL DAY TO SECOND	TIMESTAMP
TIME	TIME	INTERVAL DAY TO SECOND
TIME	Numeric type	TIME
TIME	INTERVAL YEAR TO MONTH	TIME
TIME	INTERVAL DAY TO SECOND	TIME
TIME WITH TIME ZONE	Numeric type	TIME WITH TIME ZONE
TIME WITH TIME ZONE	INTERVAL YEAR TO MONTH	TIME WITH TIME ZONE

expr1	expr2	Result type
TIME WITH TIME ZONE	INTERVAL DAY TO SECOND	TIME WITH TIME ZONE
TIMESTAMP	TIMESTAMP	INTERVAL DAY TO SECOND
TIMESTAMP	Numeric type	TIMESTAMP
TIMESTAMP	INTERVAL YEAR TO MONTH	TIMESTAMP
TIMESTAMP	INTERVAL DAY TO SECOND	TIMESTAMP
TIMESTAMP WITH TIME ZONE	TIMESTAMP WITH TIME ZONE	INTERVAL DAY TO SECOND
TIMESTAMP WITH TIME ZONE	Numeric type	TIMESTAMP WITH TIME ZONE
TIMESTAMP WITH TIME ZONE	INTERVAL YEAR TO MONTH	TIMESTAMP WITH TIME ZONE
TIMESTAMP WITH TIME ZONE	INTERVAL DAY TO SECOND	TIMESTAMP WITH TIME ZONE
INTERVAL YEAR TO MONTH	INTERVAL YEAR TO MONTH	INTERVAL YEAR TO MONTH (The result type includes all the interval range of expr1 and expr2.)
INTERVAL DAY TO SECOND	INTERVAL DAY TO SECOND	INTERVAL DAY TO SECOND (The result type includes all the interval range of expr1 and expr2.)
Refer to **INTERVAL type details which is included in INTERVAL type written in the following table**.		

Example

```
gSQL> SELECT
      TO_DATE( '2012-05-05' ) - TO_DATE( '2012-05-01' ) AS RESULT
      FROM DUAL;
RESULT
------
     4
1 row selected.
gSQL> SELECT TO_DATE( '2012-05-05' ) - 3 AS RESULT FROM DUAL;
RESULT
----------
2012-05-02
1 row selected.
gSQL> SELECT
      TO_DATE( '2012-05-05' ) - INTERVAL'01-02'YEAR TO MONTH AS RESULT
      FROM DUAL;
RESULT
----------
2011-03-05
1 row selected.
```

```
gSQL> SELECT
      INTERVAL'05-01'YEAR TO MONTH - INTERVAL'02-01'YEAR TO MONTH
      AS RESULT
      FROM DUAL;
RESULT
----------
+000003-00
1 row selected.
gSQL> SELECT INTERVAL'15 23:59:59.999999'DAY TO SECOND
           - INTERVAL'10 23:59:59.999999'DAY TO SECOND AS RESULT
      FROM DUAL;
RESULT
----------------------
+000005 00:00:00.000000
1 row selected.
```

17.6 / (DIVISION)

Syntax

```
expr1 / expr2
```

Description

It returns the division result of expr1 and expr2.

The division types and result types are as follows.
For more information, refer to **Type Conversion**.

Table 17-8 Numeric / operation

expr1	expr2	Result type
NATIVE INTEGER family • NATIVE_SMALLINT • NATIVE_INTEGER • NATIVE_BIGINT	NATIVE INTEGER family • NATIVE_SMALLINT • NATIVE_INTEGER • NATIVE_BIGINT	NATIVE_DOUBLE
NUMBER	NUMBER	NUMBER
NATIVE_DOUBLE	NATIVE_DOUBLE	NATIVE_DOUBLE

Table 17-9 (DATETIME/INTERVAL) / operation

expr1	expr2	Result type
INTERVAL YEAR TO MONTH	Numeric type	INTERVAL YEAR TO MONTH (The result type is interval type.)
INTERVAL DAY TO SECOND	Numeric type	INTERVAL DAY TO SECOND (The result type is interval type.)
Refer to **INTERVAL type details which is included in INTERVAL type written in the following table**.		

Example

```
gSQL> SELECT INTERVAL'20-10'YEAR TO MONTH / 2 AS RESULT FROM DUAL;
RESULT
----------
```

```
+000010-05
1 row selected.
gSQL> SELECT   INTERVAL'02 02:04:06.800000'DAY TO SECOND / 2 AS RESULT
      FROM DUAL;
RESULT
----------------------
+000001 01:02:03.400000
1 row selected.
```

17.7 || (CONCATENATE)

Syntax

```
str1 || str2
```

Description

CONCATENATE returns the string concatenating str1 and str2.

If either str1 or str2 is NULL, the string except NULL is returned. If both of str1 and str2 are NULL, NULL is returned.

The argument can be a type which can be converted to either character string type or binary string type. For more information, refer to **Type Conversion**.

It is an alias of **CONCAT**, **CONCATENATE**.

The result types are as follows.

Table 17-10 The result types of || (CONCATENATE)

Data type	CHAR	VARCHAR	LONG VARCHAR
CHAR	CHAR	VARCHAR	LONG VARCHAR
VARCHAR	VARCHAR	VARCHAR	LONG VARCHAR
LONG VARCHAR	LONG VARCHAR	LONG VARCHAR	LONG VARCHAR
Data type	**BINARY**	**VARBINARY**	**LONG VARBINARY**
BINARY	BINARY	VARBINARY	LONG VARBINARY
VARBINARY	VARBINARY	VARBINARY	LONG VARBINARY
LONG VARBINARY	LONG VARBINARY	LONG VARBINARY	LONG VARBINARY

Example

```
gSQL> SELECT 'DATA' || 'BASE' AS RESULT1,
             'DATA' || NULL   AS RESULT2,
               NULL || NULL   AS RESULT3
       FROM DUAL;
```

```
RESULT1  RESULT2 RESULT3
-------- ------- -------
DATABASE DATA    null
1 row selected.
```

17.8 ABS

Syntax

```
ABS( num )
```

Description

ABS returns the absolute value of num.

The num argument can be a numeric type or types which can be converted to number.
If num is NULL, then it returns NULL.

Example

```
gSQL> SELECT ABS(-1) AS RESULT1, ABS(1) AS RESULT2 FROM DUAL;
RESULT1 RESULT2
------- -------
      1       1
1 row selected.
```

17.9 ACOS

Syntax

```
ACOS( num )
```

Description

ACOS returns the arc cosine value of num.
The num argument should be in the range of -1 to 1.
If num is NULL, then it returns NULL.
It returns the radians value in the range of 0 and pi.

Example

```
gSQL> SELECT ACOS( 1 ) FROM DUAL;
ACOS( 1 )
---------
        0
1 row selected.
```

17.10 ADDDATE

Syntax

```
ADDDATE( date, INTERVAL expr unit  )
ADDDATE( expr, days )
```

Description

ADDDATE adds the second argument to the first argument, then returns the result.
If any of the input argument value is NULL, the result is also NULL.
The first argument data type can be DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, and the second argument data type can be INTERVAL or numeric.

The result type is as same as **(DATETIME/INTERVAL) + operation**.

Example

```
gSQL> SELECT ADDDATE( TO_DATE( '2012-12-12', 'YYYY-MM-DD' ), 1 ) AS RESULT
        FROM DUAL;
RESULT
----------
2012-12-13
1 row selected.
gSQL> SELECT ADDDATE( TO_DATE( '2012-11-11', 'YYYY-MM-DD' ),
                      INTERVAL'01-01'YEAR TO MONTH ) AS RESULT
      FROM DUAL;
RESULT
----------
2013-12-11
1 row selected.
```

17.11 ADDTIME

Syntax

```
ADDTIME( expr1, expr2 )
```

Description

ADDTIME adds expr2 to expr1, then returns the result.

expr1 data type can be TIME, TIME WITH TIME ZONE, TIMESTAMP, TIMESTAMP WITH TIME ZONE TYPE, and expr2 data type can be INTERVAL DAY TO SECOND TYPE.
If expr1 or expr2 is NULL, the result is NULL.

The result type is as same as **(DATETIME/INTERVAL) + operation**.

Example

```
gSQL> SELECT
      ADDTIME( TO_TIMESTAMP( '2001-05-05 06:00:00',
                             'YYYY-MM-DD HH24:MI:SS' ),
               INTERVAL'0 00:06:06.666666'DAY TO SECOND ) AS RESULT
      FROM DUAL;
RESULT
--------------------------
2001-05-05 06:06:06.666666
1 row selected.
```

17.12 ADD_MONTHS

Syntax

```
ADD_MONTHS( date, number )
```

Description

ADD_MONTHS adds as many month as the number to the date, then returns the result.
After ADD_MONTHS operation, if the date is bigger than the last day of the month, it is adjusted to the l ast day of the month.

The data type of date argument can be DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, and the num ber argument can be a numeric type.
If any of the input argument is NULL, the result is also NULL.

The result type is always DATE regardless of the input argument date type.

Example

```
gSQL> SELECT
      ADD_MONTHS( TO_DATE( '2001-07-31', 'YYYY-MM-DD' ), 1 ) AS RESULT1,
      ADD_MONTHS( TO_DATE( '2001-07-31', 'YYYY-MM-DD' ), 2 ) AS RESULT2
      FROM DUAL;
RESULT1    RESULT2
---------- ----------
2001-08-31 2001-09-30
1 row selected.
```

17.13 ASCII

Syntax

```
ASCII( char )
```

Description

It returns the database character set code of the first character of char in decimal form.
The data type of char can be a character type such as CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING or can be a type which can be converted to a character type, and the return type is NUMBER.
If char is NULL, then it returns NULL.

Example

```
gSQL> SELECT ASCII( 'G' ) AS RESULT FROM DUAL;
RESULT
------
    71
1 row selected.
```

17.14 ASIN

Syntax

```
ASIN( num )
```

Description

ASIN returns the arc sin value of num.

The num argument should be in the range of -1 to 1.
If num is NULL, then it returns NULL.

It returns the radians value in the range of -pi/2 and pi/2.

Example

```
gSQL> SELECT ASIN( 0 ) FROM DUAL;
ASIN( 0 )
---------
        0
1 row selected.
```

17.15 ATAN

Syntax

```
ATAN( num )
```

Description

ATAN returns the arc tangent value of num.

The num value range is not limited. It returns the radians value in the range of -pi/2 and pi/2.
If num is NULL, then it returns NULL.

Example

```
gSQL> SELECT ATAN(0.5) FROM DUAL;
       ATAN(0.5)
----------------
.463647609000806
1 row selected.
```

17.16 ATAN2

Syntax

```
ATAN2( num1, num2 )
```

Description

ATAN2 returns the arc tangent value of num1 and num2.

The num1 argument value range is not limited. It returns the radians value in the range of -pi and pi. Either num1 or num2 is NULL, then it returns NULL.

Example

```
gSQL> SELECT ATAN2(1,2) FROM DUAL;
      ATAN2(1,2)
----------------
.463647609000806
1 row selected.
```

17.17 AVG

Syntax

```
AVG( [ ALL ¦ DISTINCT ] num )
```

Description

It is an aggregate function, and it obtains average value of exprs.

If ALL is explicitly specified, aggregation is executed for all values.
If DISTINCT is explicitly specified, aggregation is executed for the values which exclude duplicate values.
If ALL or DISTINCT is not explicitly specified, it is processed in the same way as when ALL is specified.

Example

```
gSQL> SELECT AVG(c1) FROM t1;
AVG(C1)
-------
      2
1 row selected.
```

17.18 AVG() OVER

Syntax

```
AVG ( expr ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function AVG calculates the average value of expr.
NULL is excluded from the calculation.

Example

```
gSQL> SELECT min_price AS "MIN_PRICE"
             , AVG( min_price ) OVER ( ORDER BY min_price ) AS "AVG"
        FROM product_information
       WHERE supplier_id = 102050;
MIN_PRICE              AVG
--------- ----------------
       73               73
      247              160
      731 350.333333333333
     null 350.333333333333
     null 350.333333333333
5 rows selected.
```

17.19 BITAND

Syntax

```
BITAND( num1, num2 )
```

Description

It returns the AND operation result for the bits of num1 and num2.

The input argument data type can be NATIVE_SMALLINT, NATIVE_INTEGER, NATIVE_BIGINT or a data type which can be converted to NATIVE_BIGINT.
When converting to NATIVE_BIGINT type, the decimal point is truncated.
If any of the input argument value is NULL, the result is also NULL.

The result type is NATIVE_BIGINT.

Example

```
gSQL> SELECT BITAND( 5, 3 ) AS RESULT FROM DUAL;
RESULT
------
     1
1 row selected.
```

17.20 BITNOT

Syntax

```
BITNOT( num )
```

Description

It returns the NOT operation result for the num bit.

The input argument data type can be NATIVE_SMALLINT, NATIVE_INTEGER, NATIVE_BIGINT or a data type which can be converted to NATIVE_BIGINT.
When converting to NATIVE_BIGINT type, the decimal point is truncated.
If the input argument is NULL, the result is also NULL.

The result type is as follows.

- If the input argument is NATIVE_SMALLINT type, its result type is NATIVE_SMALLINT type.
- If the input argument is NATIVE_INTEGER type, its result type is NATIVE_INTEGER type.
- If the input argument is NATIVE_BIGINT type, its result type is NATIVE_BIGINT type.

Example

```
gSQL> SELECT BITNOT( 5 ) AS RESULT FROM DUAL;
RESULT
------
    -6
1 row selected.
```

17.21 BITOR

Syntax

```
BITOR( num1, num2 )
```

Description

It returns the OR operation result for the bits of num1 and num2.

The input argument data type can be NATIVE_SMALLINT, NATIVE_INTEGER, NATIVE_BIGINT types or a d ata type which can be converted to NATIVE_BIGINT type.
When converting to NATIVE_BIGINT type, the decimal point is truncated.
If any of the input argument value is NULL, the result is NULL.

The result type is NATIVE_BIGINT type.

Example

```
gSQL> SELECT BITOR( 5, 3 ) FROM DUAL;
BITOR( 5, 3 )
-------------
            7
1 row selected.
```

17.22 BITXOR

Syntax

```
BITXOR( num1, num2 )
```

Description

It returns the XOR operation result for the bits of num1 and num2.

The input argument data type can be NATIVE_SMALLINT, NATIVE_INTEGER, NATIVE_BIGINT or a data type which can be converted to NATIVE_BIGINT.
When converting to NATIVE_BIGINT type, the decimal point is truncated.
If any of the input argument value is NULL, the result is NULL.

The result type is NATIVE_BIGINT.

Example

```
gSQL> SELECT BITXOR( 5, 3 ) FROM DUAL;
BITXOR( 5, 3 )
--------------
             6
1 row selected.
```

17.23 BIT_LENGTH

Syntax

```
BIT_LENGTH( str )
```

Description

BIT_LENGTH returns the number of bits for str.
If str is NULL, then it returns NULL.

Example

```
gSQL> SELECT BIT_LENGTH( 'LIKE' ) AS RESULT FROM DUAL;
     RESULT
     ---------------
             32
    1 row selected.
```

17.24 BYTE_LENGTH

Syntax

```
BYTE_LENGTH( str )
```

Description

It is an alias of OCTET_LENGTH.
For more information, refer to **OCTET_LENGTH**, **LENGTHB**.

Example

- Multi byte character set (e.g. UTF8): 1 byte character

```
gSQL> SELECT BYTE_LENGTH( 'OCTET_LENGTH' ) AS RESULT_1BYTE_CHARACTERS
        FROM DUAL;
RESULT_1BYTE_CHARACTERS
-----------------------
                     12
1 row selected.
```

- Multi byte character set (e.g. UTF8): 2 byte character

```
gSQL> SELECT BYTE_LENGTH( 'αβ' ) AS RESULT_2BYTE_CHARACTERS FROM DUAL;
RESULT_2BYTE_CHARACTERS
-----------------------
                      4
1 row selected.
```

17.25 CASE2

Syntax

```
CASE2( condition1, result1
      [, condition2, result2
       , ...
       , conditionN, resultN ]
      [, default ] )
```

Description

CASE2 evaluates the condition in the described order.
If the comparison result is FALSE, it continues evaluating until TRUE comes up.
If the comparison result is TRUE, it returns the corresponding result, and does not evaluate any more.
If all the evaluation results are FALSE, it returns the default value. If the default is omitted, it returns NULL.

If multiple types are used in result, then the result type is determined according to **Result Type Combinati on Rule**.

CASE2 can be expressed by using CASE as follows.

- CASE2(condition1, res1, condition2, res2)

```
CASE WHEN condition1 THEN res1
     WHEN condition2 THEN res2
     ELSE NULL
  END
```

- CASE2(condition1, res1, condition2, res2, default)

```
CASE WHEN condition1 THEN res1
     WHEN condition2 THEN res2
     ELSE default
  END
```

Example

```
gSQL> SELECT I1,
        CASE2( I1 = 1, 'ONE', I1 = 2, 'TWO' ) AS CASE2_RESULT1,
        CASE2( I1 = 1, 'ONE', I1 = 2, 'TWO', 'NUMBER' ) AS CASE2_RESULT2
      FROM T1;
I1 CASE2_RESULT1 CASE2_RESULT2
-- ------------- -------------
 1 ONE           ONE
 2 TWO           TWO
 3 null          NUMBER
3 rows selected.
```

17.26 CBRT

Syntax

```
CBRT( num )
```

Description

It returns the cube root of num.
If num is NULL, the result is also NULL.

Example

```
gSQL> SELECT CBRT( 27 ) FROM DUAL;
CBRT( 27 )
----------
         3
1 row selected.
```

17.27 CEIL

Syntax

```
CEIL( num )
CEILING( num )
```

Description

CEIL returns the smallest integer which is equal to or bigger than num.
If num is NULL, then it returns NULL.

Example

```
gSQL> SELECT CEIL( 3.5 ) AS RESULT1, CEIL( -3.5 ) AS RESULT2 FROM DUAL;
RESULT1 RESULT2
------- -------
      4      -3
1 row selected.
```

17.28 CHAR_LENGTH

Syntax

```
CHAR_LENGTH( str )
CHARACTER_LENGTH( str )
```

Description

CHAR_LENGTH returns the number of character for str according to the character set.

The str can be a character type such as CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYIN G, or it can be a data type which can be converted to character type. The return type is NATIVE_BIGINT.

If the data type of str is CHARACTER, the trailing blanks are included in the calculation.
If str is NULL, it returns NULL.

It is an alias of **LENGTH**.

Example

Multi byte character set: (e.g. UTF8)

```
gSQL> SELECT CHAR_LENGTH( 'αβ-SUMMER' ) AS RESULT FROM DUAL;
     RESULT
     ---------------
              9
    1 row selected.
```

17.29 CHR

Syntax

```
CHR( num )
```

Description

It returns a character in the database character set code corresponding to num.

num is a numeric type.
If num is NULL, then it returns NULL.
The return type is VARCHAR.

Example

```
gSQL> SELECT CHR(71) FROM DUAL;
CHR(71)
-------
G
1 row selected.
```

17.30 CLOCK_DATE

Syntax

```
CLOCK_DATE()
```

Description

Whenever the CLOCK_DATE function is called, the current date (DATE type) value is obtained.

The differences among the functions to obtain the current date are as follows.

- TRANSACTION_DATE(): All date values in the transaction are same.
- STATEMENT_DATE(): All date values in an SQL statement are same.
- CLOCK_DATE(): Whenever the function is called, the current date value is obtained.

Example

Each row can have a different date value.

```
gSQL> SELECT CLOCK_DATE() FROM t1;
CLOCK_DATE()
------------
2013-12-12
2013-12-12
2013-12-13
3 rows selected.
```

17.31 CLOCK_LOCALTIME

Syntax

```
CLOCK_LOCALTIME()
```

Description

Whenever the CLOCK_LOCALTIME function is called, the current time value without TIME ZONE (TIME WITHOUT TIME ZONE type) is obtained.

The differences among the functions to obtain the current time are as follows.

- TRANSACTION_LOCALTIME(): All time values in the transaction are same.
- STATEMENT_LOCALTIME(): All time values in an SQL statement are same.
- CLOCK_LOCALTIME(): Whenever the function is called, the current time value is obtained.

Example

Each row can have a different time value.

```
gSQL> SELECT CLOCK_LOCALTIME() FROM t1;
CLOCK_LOCALTIME()
-----------------
14:42:05.470757
14:42:05.470759
14:42:05.470759
3 rows selected.
```

17.32 CLOCK_LOCALTIMESTAMP

Syntax

```
CLOCK_LOCALTIMESTAMP()
```

Description

Whenever the CLOCK_LOCALTIMESTAMP() function is called, the current TIMESTAMP value without TIM E ZONE (TIMESTAMP WITHOUT TIME ZONE type) is obtained.

The differences among the functions to obtain the current TIMESTAMP are as follows.

- TRANSACTION_LOCALTIMESTAMP(): All TIMESTAMP values in the transaction are same.
- STATEMENT_LOCALTIMESTAMP(): All TIMESTAMP values in an SQL statement are same.
- CLOCK_LOCALTIMESTAMP(): Whenever the function is called, the current timestamp value is obtained.

Example

Each row can have a different timestamp value.

```
gSQL> SELECT CLOCK_LOCALTIMESTAMP() FROM t1;
CLOCK_LOCALTIMESTAMP()
--------------------------
2013-12-12 14:46:17.309206
2013-12-12 14:46:17.309209
2013-12-12 14:46:17.309209
```

17.33 CLOCK_TIME

Syntax

```
CLOCK_TIME()
```

Description

Whenever the CLOCK_TIME() function is called, the current time value with TIME ZONE (TIME WITH TIME ZONE type) is obtained.

The differences among the functions to obtain the current time are as follows.

- TRANSACTION_TIME(): All time values in the transaction are same.
- STATEMENT_TIME(): All time values in an SQL statement are same.
- CLOCK_TIME(): Whenever the function is called, the current time value is obtained.

Example

Each row can have a different time value.

```
gSQL> SELECT CLOCK_TIME() FROM t1;
CLOCK_TIME()
----------------------
14:48:21.052324 +09:00
14:48:21.052326 +09:00
14:48:21.052327 +09:00
3 rows selected.
```

17.34 CLOCK_TIMESTAMP

Syntax

```
CLOCK_TIMESTAMP()
```

Description

Whenever CLOCK_TIMESTAMP() function is called, the current TIMESTAMP value with TIME ZONE (TIME STAMP WITH TIME ZONE type) is obtained.

The differences among the functions to obtain the current TIMESTAMP are as follows.

- TRANSACTION_TIMESTAMP(): All TIMESTAMP values in the transaction are same.
- STATEMENT_TIMESTAMP(): All TIMESTAMP values in an SQL statement are same.
- CLOCK_TIMESTAMP(): Whenever the function is called, the current TIMESTAMP value is obtained.

Example

Each row can have a different timestamp value.

```
gSQL> SELECT CLOCK_TIMESTAMP() FROM t1;
CLOCK_TIMESTAMP()
-------------------------------
2013-12-12 14:49:45.051709 +09:00
2013-12-12 14:49:45.051714 +09:00
2013-12-12 14:49:45.051714 +09:00
3 rows selected.
```

17.35 COALESCE

Syntax

```
COALESCE( expr1, ..., exprN )
```

Description

It returns the first non null expr in the expr list.
If all expr in the expr list are null, it returns null.
In the expr list, there should be two or more expr.

If multiple types are in the expr list, the result type is determined by the **Result Type Combination Rule**.

- COALESCE can be expressed by using CASE as follows.
    - COALESCE(expr1, expr2)

```
CASE WHEN expr1 IS NOT NULL THEN expr1
       ELSE expr2
  END
```

    - COALESCE(expr1, expr2, ..., exprN)

```
CASE WHEN expr1 IS NOT NULL THEN expr1
       ELSE COALESCE( expr2, ..., exprN )
  END
```

Example

```
gSQL> SELECT COALESCE( NULL, 1, 2 ) FROM DUAL;
COALESCE( NULL, 1, 2 )
----------------------
                     1
1 row selected.
gSQL> SELECT COALESCE( NULL, NULL, NULL ) FROM DUAL;
```

```
COALESCE( NULL, NULL, NULL )
----------------------------
null
1 row selected.
```

17.36 CONCAT

Syntax

```
CONCAT( str1, str2, ... )
```

Description

It is an alias of || (CONCATENATE).
It is an argument of CONCAT function and 2 ~ 254 number of CONCATs can be set.
For more information, refer to **|| (CONCATENATE)**, **CONCATENATE**.

Example

```
gSQL> SELECT CONCAT( 'DATA', 'BASE' ) AS RESULT FROM DUAL;
RESULT
--------
DATABASE
1 row selected.
```

17.37 CONCATENATE

Syntax

```
CONCATENATE( str1, str2, ... )
```

Description

It is an alias of || (CONCATENATE).
It is an argument of CONCATENATE function and 2 ~ 254 number of CONCATENATEs can be set.
For more information, refer to **CONCAT**, **|| (CONCATENATE)**.

Example

```
gSQL> SELECT CONCATENATE( 'DATA', 'BASE' ) AS RESULT FROM DUAL;
RESULT
--------
DATABASE
1 row selected.
```

17.38 CORR() OVER

Syntax

```
CORR( expr1, expr2 ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function CORR calculates the coefficient of correlation for the pair of exprs.

If expr1 or expr2 is NULL, then it is excluded from the calculation.
If the number of rows for the pair of exprs is one or less, then it returns NULL as a result.

Example

```
gSQL> SELECT employee_id, TO_CHAR( hire_date, 'YYYY' ) AS hire_date, salary,
             CORR( TO_CHAR( hire_date, 'YYYY' ), salary ) OVER ( ORDER BY employee_id ) AS
corr
      FROM employees
      WHERE department_id = 60;
EMPLOYEE_ID HIRE_DATE SALARY              CORR
----------- --------- ------ -----------------
        103 1990        9000              null
        104 1991        6000                -1
        105 1997        4800 -.805837379342809
        106 1998        4800 -.840210805972693
        107 1999        4200 -.875185734200534
5 rows selected.
```

17.39 COS

Syntax

```
COS(num)
```

Description

It returns the COSINE value of num.
If the num argument is NULL, the result is also NULL.

Example

```
gSQL> SELECT COS( 0 ) FROM DUAL;
COS( 0 )
--------
       1
1 row selected.
```

17.40 COT

Syntax

```
COT(num)
```

Description

It returns the COTANGENT value of num.
If the num argument is NULL, the result is also NULL.

Example

```
gSQL> SELECT COT( 1 ) FROM DUAL;
        COT( 1 )
----------------
.642092615934331
1 row selected.
```

17.41 COUNT

Syntax

```
COUNT( [ ALL ¦ DISTINCT ] expr )
```

Description

It is an aggregate function. It returns the number of row whose expr is not NULL.

If ALL is explicitly specified, aggregation is executed for all values.
If DISTINCT is explicitly specified, aggregation is executed for the values which exclude duplicate values.
If ALL or DISTINCT is not explicitly specified, it is processed in the same way as when ALL is specified.

Example

```
gSQL> SELECT COUNT(c1) FROM t1;
COUNT(C1)
---------
        3
1 row selected.
```

17.42 COUNT() OVER

Syntax

```
COUNT ( expr ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function COUNT counts the number of rows.
NULL is excluded from the calculation.

Example

```
gSQL> SELECT min_price AS "MIN_PRICE"
             , COUNT( min_price ) OVER ( ORDER BY min_price ) AS "COUNT"
        FROM product_information
       WHERE supplier_id = 102050;
MIN_PRICE COUNT
--------- -----
       73     1
      247     2
      731     3
     null     3
     null     3
5 rows selected.
```

17.43 COUNT(*)

Syntax

```
COUNT(*)
```

Description

It is an aggregate function, and the number of rows is obtained.
It has nothing to do with whether it is NULL or not because an expression is not explicitly specified.

Example

```
gSQL> SELECT COUNT(*) FROM t1;
COUNT(*)
--------
       4
1 row selected.
```

17.44 COUNT(*) OVER

Syntax

```
COUNT(*) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function COUNT(*) counts the number of rows.
It does not separately specify an expression, so it is irrelevant whether the value is NULL or not.

Example

```
gSQL> SELECT min_price AS "MIN_PRICE"
             , COUNT(*) OVER ( ORDER BY min_price ) AS "COUNT(*)"
        FROM product_information
       WHERE supplier_id = 102050;
MIN_PRICE COUNT(*)
--------- --------
       73        1
      247        2
      731        3
     null        5
     null        5
5 rows selected.
```

17.45 COVAR_POP() OVER

Syntax

```
COVAR_POP( expr1, expr2 ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function COVAR_POP calculates the population covariance for the pair of exprs.

If expr1 or expr2 is NULL, then it is excluded from the calculation.
If the number of rows for the pair of exprs is one or less, then it returns 0 as a result.

Example

```
gSQL> SELECT employee_id, TO_CHAR( hire_date, 'YYYY' ) AS hire_date, salary,
             COVAR_POP( TO_CHAR( hire_date, 'YYYY' ), salary ) OVER ( ORDER BY employee_id )
AS covar_pop
      FROM employees
      WHERE department_id = 60;
EMPLOYEE_ID HIRE_DATE SALARY COVAR_POP
----------- --------- ------ ---------
        103 1990        9000         0
        104 1991        6000      -750
        105 1997        4800     -4400
        106 1998        4800     -5100
        107 1999        4200     -5640
5 rows selected.
```

17.46 COVAR_SAMP() OVER

Syntax

```
COVAR_SAMP( expr1, expr2 ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function COVAR_SAMP calculates the sample covariance for the pair of exprs.

If expr1 or expr2 is NULL, then it is excluded from the calculation.
If the number of rows for the pair of exprs is one or less, then it returns NULL as a result.

Example

```
gSQL> SELECT employee_id, TO_CHAR( hire_date, 'YYYY' ) AS hire_date, salary,
             COVAR_SAMP( TO_CHAR( hire_date, 'YYYY' ), salary ) OVER ( ORDER BY employee_id )
AS covar_samp
      FROM employees
      WHERE department_id = 60;
EMPLOYEE_ID HIRE_DATE SALARY COVAR_SAMP
----------- --------- ------ ----------
        103 1990        9000       null
        104 1991        6000      -1500
        105 1997        4800      -6600
        106 1998        4800      -6800
        107 1999        4200      -7050
5 rows selected.
```

17.47 CUME_DIST() OVER

Syntax

```
CUME_DIST( ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function CUME_DIST calculates the cumulative distribution according to the relative position of t he current row's value.

The result of CUME_DIST function is the number between 0 and 1.
If the row values are same, then it returns the same result which is the biggest cumulative distribution val ue.

window frame is not available.

Example

```
gSQL> SELECT department_id, salary,
             CUME_DIST() OVER ( ORDER BY salary ) AS cume_dist
      FROM employees
      WHERE department_id = 60;
DEPARTMENT_ID SALARY CUME_DIST
------------- ------ ---------
           60   4200        .2
           60   4800        .6
           60   4800        .6
           60   6000        .8
           60   9000         1
5 rows selected.
```

17.48 CURRENT_CATALOG

Syntax

```
CURRENT_CATALOG [()]
```

Description

The catalog name (database name) is obtained.

Example

```
gSQL> SELECT CURRENT_CATALOG FROM dual;
CURRENT_CATALOG
---------------
TEST_DB
1 row selected.
```

17.49 CURRENT_DATE

Syntax

```
CURRENT_DATE [()]
STATEMENT_DATE()
```

Description

The current date (DATE type) is obtained.

CURRENT_DATE is an SQL standard function.

The differences among the functions to obtain the current date are as follows.

- TRANSACTION_DATE(): All date values in the transaction are same.
- CURRENT_DATE, STATEMENT_DATE(): All date values in an SQL statement are same.
- CLOCK_DATE(): Whenever the function is called, the current date value is obtained.

Example

```
gSQL> SELECT CURRENT_DATE FROM t1;
CURRENT_DATE
------------
2013-12-12
2013-12-12
2013-12-12
3 rows selected.
```

17.50 CURRENT_SCHEMA

Syntax

```
CURRENT_SCHEMA [()]
```

Description

User's current SCHEMA is obtained.

Example

```
gSQL> SELECT CURRENT_SCHEMA FROM dual;
CURRENT_SCHEMA
--------------
PUBLIC
1 row selected.
```

17.51 CURRENT_TIME

Syntax

```
CURRENT_TIME [()]
STATEMENT_TIME()
```

Description

The current TIME WITH TIME ZONE type value based on the session time is obtained.

CURRENT_TIME is an SQL standard function.

The differences among the functions to obtain the current time are as follows.

- TRANSACTION_TIME(): All time values in the transaction are same.
- CURRENT_TIME, STATEMENT_TIME(): All time values in an SQL statement are same.
- CLOCK_TIME(): Whenever the function is called, the current time value is obtained.

Example

All rows have the same value.

```
gSQL> SELECT CURRENT_TIME FROM t1;
CURRENT_TIME
----------------------
16:27:10.116396 +09:00
16:27:10.116396 +09:00
16:27:10.116396 +09:00
3 rows selected.
```

17.52 CURRENT_TIMESTAMP

Syntax

```
CURRENT_TIMESTAMP [()]
STATEMENT_TIMESTAMP()
```

Description

It obtains the TIMESTAMP WITH TIME ZONE type value based on the session time.

CURRENT_TIMESTAMP is an SQL standard function.

The differences among the functions to obtain the current TIMESTAMP are as follows.
• TRANSACTION_TIMESTAMP(): All TIMESTAMP values in the transaction are same.
• CURRENT_TIMESTAMP, STATEMENT_TIMESTAM(): All TIMESTAMP values in an SQL statement are sa me.
• CLOCK_TIMESTAMP(): Whenever the function is called, the current timestamp value is obtained.

Example

All rows have the same value.

```
gSQL> SELECT CURRENT_TIMESTAMP FROM t1;
CURRENT_TIMESTAMP
-------------------------------
2013-12-12 16:34:55.649632 +09:00
2013-12-12 16:34:55.649632 +09:00
2013-12-12 16:34:55.649632 +09:00
3 rows selected.
```

17.53 CURRENT_USER

Syntax

```
CURRENT_USER [()]
```

Description

It returns the current user.

The user information is managed in three types as follows.

- Logon user: It is a user who performed login, and it is maintained until the connection is closed.
- Session user: It is as same as the first logon user, but it can be changed using the SET SESSION AUTH ORIZATION statement.
- Current user: It is generally as same as the session user, but it is temporarily changed internally in syst em to control access when using the PSM, view.
    - The session user and current user is similar to the difference between the unix system's real user and the effective user.

Example

```
% gsql sys gliese
gSQL> SET SESSION AUTHORIZATION test;
Session set.
gSQL> SELECT
        LOGON_USER() AS result1,
        SESSION_USER() AS result2,
        CURRENT_USER() AS result3
      FROM DUAL;
RESULT1 RESULT2 RESULT3
------- ------- -------
SYS     TEST    TEST
1 row selected.
```

17.54 CURRVAL

Syntax

```
seq_name.CURRVAL
CURRVAL(seq_name)
```

Description

The current value of the sequence object is obtained.

A sequence value should be set with NEXTVAL(seq_name) at least once.

Example

```
gSQL> SELECT seq.CURRVAL FROM dual;
SEQ.CURRVAL
-----------
          1
1 row selected.
```

17.55 DATEADD

Syntax

```
DATEADD( datepart, number, date )
```

Description

It adds number to the specified datepart of date, and returns the result.

If the number is decimal point, it is not rounded off.
The date data type can be DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIME, TIME WITH TIME ZONE.
If number or date is NULL, the result is also NULL.

The result type which is as same as the input date argument type is returned.

Table 17-11 Available string format in datepart

datepart	Description
YEAR	Year
QUARTER	Quarter
MONTH	Month
DAYOFYEAR	Day of year
DAY	Day
WEEK	Week
WEEKDAY	Weekday
HOUR	Hour
MINUTE	Minute
SECOND	Second
MILLISECOND	Millisecond
MICROSECOND	Microsecond

Example

```
gSQL> SELECT
      DATEADD( YEAR, 1, TO_DATE( '2013-05-14', 'YYYY-MM-DD' ) ) AS RESULT
      FROM DUAL;
RESULT
----------
2014-05-14
1 row selected.
gSQL> SELECT
      DATEADD( MONTH, 13, TO_DATE('2013-05-14', 'YYYY-MM-DD') ) AS RESULT
      FROM DUAL;
RESULT
----------
2014-06-14
1 row selected.
gSQL> SELECT
      DATEADD( DAY, 397, TO_DATE('2013-05-14', 'YYYY-MM-DD') ) AS RESULT
      FROM DUAL;
RESULT
----------
2014-06-15
1 row selected.
```

17.56 DATEDIFF

Syntax

```
DATEDIFF( datepart, startdate, enddate )
```

Description

It substracts startdate from enddate, then returns the result to the specified datepart.

If the startdate or enddate is NULL, the result is also NULL.
The data type of startdate and enddate can be DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIME.

The result type is NUMBER.

Table 17-12 Available string format in datepart

datepart	Description
YEAR	Year
QUARTER	Quarter
MONTH	Month
DAYOFYEAR	Day of year
DAY	Day
HOUR	Hour
MINUTE	Minute
SECOND	Second
MILLISECOND	Millisecond
MICROSECOND	Microsecond

Example

```
gSQL> SELECT
      DATEDIFF( YEAR,
                TO_DATE( '2013-05-14', 'YYYY-MM-DD' ),
                TO_DATE( '2014-06-15', 'YYYY-MM-DD' ) ) AS RESULT
      FROM DUAL;
```

```
RESULT
------
     1
1 row selected.
gSQL> SELECT
      DATEDIFF( MONTH,
                TO_DATE( '2013-05-14', 'YYYY-MM-DD' ),
                TO_DATE( '2014-06-15', 'YYYY-MM-DD' ) ) AS RESULT
      FROM DUAL;
RESULT
------
    13
1 row selected.
gSQL> SELECT
      DATEDIFF( DAY,
                TO_DATE( '2013-05-14', 'YYYY-MM-DD' ),
                TO_DATE( '2014-06-15', 'YYYY-MM-DD' ) ) AS RESULT
      FROM DUAL;
RESULT
------
   397
1 row selected.
```

17.57 DATE_ADD

Syntax

```
DATE_ADD( date, INTERVAL expr unit  )
```

Description

It is the same function as **ADDDATE** (date, INTERVAL expr unit).

Example

```
gSQL> SELECT
      DATE_ADD( TO_DATE( '2012-01-02', 'YYYY-MM-DD' ),
                INTERVAL '2-2' YEAR TO MONTH ) AS RESULT
      FROM DUAL;
RESULT
----------
2014-03-02
1 row selected.
```

17.58 DATE_PART

Syntax

```
DATE_PART( field, datetime )
```

Description

The result of DATE_PART is as same as the result of the EXTRACT function. It searches for the specified fi eld from the input datetime type, and returns it.

The field argument should be text literal, and YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, TIMEZONE _HOUR, TIMEZONE_MINUTE can be specified to text literal.
The datetime argument data type can be DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIME, TIME WITH TIME ZONE, INTERVAL.

If field is not in the range of datetime, an error is returned.
For DATE type, field should be YEAR, MONTH, DAY, otherwise an error is returned.
If datatime is NULL, then it returns NULL.

The return type is NUMBER.

For more information, refer to **EXTRACT**.

Example

```
gSQL> SELECT
      DATE_PART( 'DAY', TO_DATE( '2012-01-02', 'YYYY-MM-DD' ) ) AS RESULT
      FROM DUAL;
RESULT
------
     2
1 row selected.
gSQL> SELECT
      DATE_PART( 'YEAR', INTERVAL'9-11'YEAR TO MONTH ) AS RESULT
      FROM DUAL;
```

```
RESULT
------
     9
1 row selected.
```

17.59 DECODE

Syntax

```
DECODE( expr, comparison_expr1, result1
          [, comparison_expr2, result2
           , ...
           , comparison_exprN, resultN ]
          [, default ] )
```

Description

It evaluates expr and comparison_expr in the described order in DECODE statement using equal operation.
If the comparison result is FALSE, it continues evaluating until TRUE comes up.
If the comparison result is TRUE, it returns the corresponding result, and does not evaluate any more.

If expr and comparison_expr are equal, or if both expr and comparison_expr are NULL(null = null), it is evaluated as TRUE, and returns the corresponding result.
If all of the evaluated results are FALSE, it returns default. If the default is omitted, it returns NULL.

- Comparing expr and comparison_expr
  All expr, comparison_expr1, ..., comparison_exprN are converted to the data type of comparison_expr1 (the first comparison_expr), then they are compared.
  If comparison_expr1 (the first comparison_expr) is a character type and a numeric type then it becomes the type including the range of types described in each *expr, comparison_expr1, ..., comparison_exprN*.
  If all types described in *expr, comparison_expr1, ..., comparison_exprN* are CHAR, then VARCHAR type comparison is performed.

- Result type
  The result type becomes the data type of result1 (the first result).
  If the data type of result1 (the first result) is a character type and a numeric type then it becomes the type including the range of types described in result1, ..., resultN each.
  If result1 (the first result) is CHAR or NULL, then the result type is VARCHAR.

DECODE can be expressed by using CASE as follows.

- DECODE(expr, comp_expr1, res1, comp_expr2, res2)

```
CASE WHEN (expr = comp_expr1) OR (expr IS NULL AND comp_expr1 IS NULL ) THEN res1
     WHEN (expr = comp_expr2) OR (expr IS NULL AND comp_expr2 IS NULL ) THEN res2
     ELSE NULL
  END
```

- DECODE(expr, comp_expr1, res1, comp_expr2, res2, default)

```
CASE WHEN (expr = comp_expr1) OR (expr IS NULL AND comp_expr1 IS NULL ) THEN res1
     WHEN (expr = comp_expr2) OR (expr IS NULL AND comp_expr2 IS NULL ) THEN res2
     ELSE default
  END
```

Example

```
gSQL> SELECT I1,
             DECODE( I1, 1, 'ONE',
                         2, 'TWO',
                         NULL, 'NULL VALUE',
                         'DEFAULT VALUE' ) AS DECODE_RESULT
      FROM T1;
  I1 DECODE_RESULT
---- -------------
   1 ONE
   2 TWO
null NULL VALUE
   3 DEFAULT VALUE
4 rows selected.
```

17.60 DEGREES

Syntax

```
DEGREES( radians )
```

Description

It converts a degree radians to a value in degrees, and returns the converted value.
If radians is NULL, then it returns NULL.

Example

```
gSQL> SELECT DEGREES( PI() ) AS RESULT FROM DUAL;
RESULT
------
   180
1 row selected.
```

17.61 DENSE_RANK() OVER

Syntax

```
DENSE_RANK( ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function DENSE_RANK calculates the ranking.

The ranking is a consecutive integer starting from 1, and rows with the same value have the same rank. However, unlike RANK, even when rows with the same value appear, the ranking is not skipped.

window frame is not available.

Example

```
gSQL> SELECT department_id, salary,
             DENSE_RANK() OVER ( ORDER BY salary ) AS d_rank
      FROM employees
      WHERE department_id = 60;
DEPARTMENT_ID SALARY D_RANK
------------- ------ ------
           60   4200      1
           60   4800      2
           60   4800      2
           60   6000      3
           60   9000      4
5 rows selected.
```

17.62 DIGEST

Syntax

```
DIGEST( data, type )
```

Description

It hashes the data to the given type, and returns the result in VARBINARY type.

An implicit conversion may occur when inputting data type based on the following rules.

- Input the BINARY, VARBINARY type data in VARBINARY type.
- Input LONG VARBINARY type data in LONG VARBINARY type.
- Input LONG VARCHAR type data in LONG VARCHAR type.
- Input all other type of data after implicitly converting it to VARCHAR type.

DIGEST function supports the following hash types.

- The result of 'SHA1' is 20 byte varbinary.
- The result of 'SHA224' is 28 byte varbinary.
- The result of 'SHA256' is 32 byte varbinary.
- The result of 'SHA384' is 48 byte varbinary.
- The result of 'SHA512' is 64 byte varbinary.

Use HEX function to view the result in hexadecimal character because the result is returned in VARBINAR Y type. In this case, the length becomes double of the original.

Example

```
gSQL> SELECT HEX( DIGEST( 'my password', 'SHA256' ) ) AS RESULT FROM DUAL;
RESULT
----------------------------------------------------------------
BB14292D91C6D0920A5536BB41F3A50F66351B7B9D94C804DFCE8A96CA1051F2
1 row selected.
```

17.63 DUMP

Syntax

```
DUMP( expr )
```

Description

It returns internal representation information of expr.
Internal representation information is displayed as the data type, byte length and data information.

expr can be any data types.
If expr is NULL, then it returns NULL.
The return type is CHARACTER VARYING.

Example

```
gSQL> SELECT DUMP( 'DUMP' ) AS RESULT FROM DUAL;
RESULT
--------------------------------
Type=CHAR Len=4 : Str=68,85,77,80
1 row selected.
```

17.64 EXP

Syntax

```
EXP( num )
```

Description

It returns squared value of e (base of natural logarithm)'s num.
If num is NULL, then it returns NULL.

Example

```
gSQL> SELECT EXP( 1 ) AS RESULT FROM DUAL;
          RESULT
----------------
2.71828182845905
1 row selected.
```

17.65 EXTRACT

Syntax

```
EXTRACT( <field> FROM datetime )
<field> ::=
        YEAR
      ¦ MONTH
      ¦ DAY
      ¦ HOUR
      ¦ MINUTE
      ¦ SECOND
      ¦ TIMEZONE_HOUR
      ¦ TIMEZONE_MINUTE
```

Description

It searches for the specified field from an input datetime type, and returns it.

The datetime argument data type can be DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIME, TIME WITH TIME ZONE, INTERVAL.

If field is not in the range of datetime, an error is returned.
For DATE type, the field should be YEAR, MONTH, DAY, otherwise an error is returned.
The return type is NUMBER.

Result of EXTRACT is as same as the result of the **DATE_PART** function.

Example

```
gSQL> SELECT
      EXTRACT( SECOND FROM TO_TIMESTAMP( '2012-12-13 01:23:44.5',
                                         'YYYY-MM-DD HH24:MI:SS.FF1' )  )
              AS RESULT
      FROM DUAL;
RESULT
```

```
------
  44.5
1 row selected.
gSQL> SELECT
      EXTRACT( YEAR FROM CAST('2-3' AS INTERVAL YEAR TO MONTH) ) AS RESULT
      FROM DUAL;
RESULT
------
     2
1 row selected.
```

17.66 FACTORIAL

Syntax

```
FACTORIAL( num )
```

Description

It multiplies the successive natural numbers from 1 to num in order, and returns the result.
If num is NULL, then it returns NULL.

Example

```
gSQL> SELECT FACTORIAL( 5 ) AS RESULT FROM DUAL;
RESULT
------
   120
1 row selected.
```

17.67 FIRST() OVER

Syntax

```
aggregation_function KEEP ( DENSE_RANK FIRST ORDER BY <sort specification list> ) OVER <
window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function FIRST sorts the sort specification list written in *order by* within KEEP clause, then return s the aggregation function value of rows whose DENSE_RANK is 1.

aggregation_functions are AVG, COUNT, COUNT(*), SUM, MAX, MIN, STDDEV, VARIANCE.

order by is not available within the window clause.
window frame is not available.

Example

```
gSQL> SELECT department_id, salary,
           DENSE_RANK() OVER ( ORDER BY department_id ) AS "DENSE_RANK",
           MAX( salary ) KEEP ( DENSE_RANK FIRST ORDER BY department_id ) OVER () AS
"MAX_FIRST"
  FROM employees
 WHERE department_id BETWEEN 90 AND 100;

DEPARTMENT_ID SALARY DENSE_RANK MAX_FIRST
------------- ------ ---------- ---------
           90  24000          1     24000
           90  17000          1     24000
           90  17000          1     24000
          100  12000          2     24000
          100   9000          2     24000
          100   8200          2     24000
```

```
          100   7700          2     24000
          100   7800          2     24000
          100   6900          2     24000
9 rows selected.
```

17.68 FIRST_VALUE() OVER

Syntax

```
FIRST_VALUE ( expr ) [ RESPECT NULLS ¦ IGNORE NULLS ] OVER < window name or specification >
FIRST_VALUE ( expr [ RESPECT NULLS ¦ IGNORE NULLS ] ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function FIRST_VALUE returns the first value of expr.

RESPECT NULLS returns the first value of rows including NULL.
IGNORE NULLS returns the first value of row except for NULL.
If it is not specified, the default value is RESPECT NULLS.

order by is not available in window clause.
window frame is not available.

Example

```
gSQL> SELECT min_price AS "MIN_PRICE"
             , FIRST_VALUE( min_price ) OVER ( ORDER BY min_price NULLS FIRST ) AS
"FIRST_VALUE"
        FROM product_information
       WHERE supplier_id = 102050;
MIN_PRICE FIRST_VALUE
--------- -----------
     null        null
     null        null
       73        null
      247        null
      731        null
5 rows selected.
```

The following is an example of specifying IGNORE NULLS in null_treatment.

```
gSQL> SELECT min_price AS "MIN_PRICE"
            , FIRST_VALUE( min_price IGNORE NULLS ) OVER ( ORDER BY min_price NULLS FIRST )
                                                          AS "FIRST_VALUE"
        FROM product_information
       WHERE supplier_id = 102050;
MIN_PRICE FIRST_VALUE
--------- -----------
     null        null
     null        null
       73          73
      247          73
      731          73
5 rows selected.
```

17.69 FLOOR

Syntax

```
FLOOR( num )
```

Description

It returns the biggest integer which is equal to or smaller than num.
If num is NULL, then it returns NULL.

Example

```
gSQL> SELECT FLOOR(42.8) AS RESULT1, FLOOR(-42.8) AS RESULT2 FROM DUAL;
RESULT1 RESULT2
------- -------
     42     -43
1 row selected.
```

17.70 FROM_BASE64

Syntax

```
FROM_BASE64( str )
```

Description

The converted character by base 64 encoding is input to FROM_BASE64, then the decoded binary string i s returned.

The input argument data type can be a character type such as CHARACTER VARYING, CHARACTER LON G VARYING, and the result type is a binary character such as BINARY VARYING, or BINARY LONG VARYI NG.

If str is NULL, then the result value is also NULL.
If str includes characters which are not in the range of base64 character, then it returns an error.
A newline, carriage return, tab, and space of str is ignored when decoding.

For more information, refer to **TO_BASE64**.

Example

```
gSQL> SELECT FROM_BASE64( TO_BASE64( 'abc' ) ),
             FROM_BASE64( TO_BASE64( 'abcd' ) )
        FROM DUAL;
FROM_BASE64( TO_BASE64( 'abc' ) ) FROM_BASE64( TO_BASE64( 'abcd' ) )
--------------------------------- ---------------------------------
616263                            61626364
1 row selected.
```

17.71 FROM_TZ

Syntax

```
FROM_TZ( timestamp, timezone )
```

Description

FROM_TZ function converts the timestamp and the timezone in the specified format to TIMESTAMP WIT H TIME ZONE type, then returns it.

The timestamp argument should be TIMESTAMP type or the type convertible to TIMESTAMP type.
If the timestamp argument is NULL, then the result value is also NULL.

The timezone argument should be CHARACTER type such as CHARACTER and CHARACTER VARYING, a nd the format is 'TZH:TZM'.
If the timezone argument is NULL, then the result is also NULL.

The result type is TIMESTAMP(6) WITH TIME ZONE.

Example

```
gSQL> SELECT
      FROM_TZ( TIMESTAMP'2021-01-01 10:10:20.000000', '+06:00' ) AS RESULT
        FROM DUAL;
RESULT
---------------------------------
2021-01-01 10:10:20.000000 +06:00
1 row selected.
```

17.72 GREATEST

Syntax

```
GREATEST( expr1 [, expr2, ... exprn ] )
```

Description

It returns the largest value among the received expr argument.

If any expr argument is NULL, the result value is NULL.

The result type becomes the data type of expr1 (the first expr).
If the data type of expr1 (the first expr) is a character type and a numeric type then it becomes the type in cluding the range of expr1, ..., exprN each.
If all of expr1, ..., exprN is described in CHAR type, then all exprs are compared in VARCHAR type and the result type is VARCHAR.

Example

```
gSQL> SELECT GREATEST( 100, 0, 200, 150, 1 ) AS RESULT FROM DUAL;
RESULT
------
   200
1 row selected.
```

17.73 HEX

Syntax

```
HEX( str )
```

Description

It returns a str argument in hexadecimal character.
A str argument data type can be a character type such as CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING, a type which can be converted to a character type, or a binary character type such as BINARY, BINARY VARYING, BINARY LONG VARYING.
The result type is a character type such as CHARACTER VARYING or CHARACTER LONG VARYING.

If str is NULL, then the result value is also NULL.

If an argument of HEX function is a numeric type, then it returns an error.
To convert a decimal number to a hexadecimal number, use TO_CHAR() function by using 'X' number format.
e.g. TO_CHAR(255, 'XX')

For more information, refer to **UNHEX**.

Example

```
gSQL> SELECT HEX( 'abc' ) FROM DUAL;
HEX( 'abc' )
------------
616263
1 row selected.
```

17.74 INITCAP

Syntax

```
INITCAP( str )
```

Description

It converts the first letter in each word of string str into uppercase, and converts all other letters into lowe rcase, then it returns the result.

str data type can be a character type such as CHARACTER, CHARACTER VARYING, CHARACTER LONG V ARYING.

Each word in string is classified by white space or characters which are not alphanumeric.
If str is NULL, the result is also NULL.

The return type is as same as str argument datatype.

Example

```
gSQL> SELECT INITCAP( 'hi GLIESE' ) AS RESULT FROM DUAL;
RESULT
---------
Hi Gliese
1 row selected.
```

17.75 INSTR

Syntax

```
INSTR( str, substr [, position [, occurrence ] ] )
```

Description

It search for occurrenceth substr starting from str's position, and returns its location.

The data types of str arguments and substr arguments can be a character type such as CHARACTER, CHA RACTER VARYING, CHARACTER LONG VARYING, or a binary character type such as BINARY, BINARY VA RYING, BINARY LONG VARYING.

The position argument and the occurrence argument can be numeric data type.

If position and occurrence are omitted, the default is 1.
The position and occurrence start from 1, and they are calculated in character unit according to character set (not in byte unit).

The position means the first position to search substr in str, it should not be zero, but an integer value.

- If the position is positive: It compares forwards (toward the right) from the beginning of str until it finds the position of substr.
- If the position is negative: It compares backwards (toward the left) from the end of str it finds the position of substr.
- If the position is 0: The result is 0.

The occurrence means the number of repeating the subtr in the str, and it should be a positive integer.

If any of the input argument is NULL, the result is also NULL.

Example

```
gSQL> SELECT INSTR( 'ABCD ABCD ABCDABCD', 'BC' ) AS RESULT1,
             INSTR( 'ABCD ABCD ABCDABCD', 'BC', 4 ) AS RESULT2
      FROM DUAL;
RESULT1 RESULT2
```

```
------- -------
      2       7
1 row selected.
gSQL> SELECT INSTR( 'ABCD ABCD ABCDABCD', 'BC', 5 , 3 ) AS RESULT1,
             INSTR( 'ABCD ABCD ABCDABCD', 'BC', -5,  3 ) AS RESULT2
      FROM DUAL;
RESULT1 RESULT2
------- -------
     16       2
1 row selected.
```

17.76 LAG() OVER

Syntax

```
LAG ( expr [, offset [, default ] ] ) [ RESPECT NULLS ¦ IGNORE NULLS ] OVER < window name or specification >
LAG ( expr [ RESPECT NULLS ¦ IGNORE NULLS ] [, offset [, default ] ] ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function LAG returns the row value ahead from the current row as far as offset.
If offset is out of window range, then it returns the default value.

If offset and default values are not specified, then it is set to the default value.
The default value of offset is 1 and the the default value of default is NULL.

RESPECT NULLS returns the row value ahead as far as offset including NULL.
IGNORE NULLS returns the row value ahead as far as offset except for NULL.
If it is not specified, the default value is RESPECT NULLS.

window frame is not available.

Example

```
gSQL> SELECT department_id, employee_id, manager_id,
             LAG( manager_id ) OVER ( ORDER BY employee_id ) AS lag
        FROM employees
       WHERE department_id = 90;
DEPARTMENT_ID EMPLOYEE_ID MANAGER_ID  LAG
------------- ----------- ---------- ----
           90         100       null null
           90         101        100 null
           90         102        100  100
```

```
3 rows selected.
```

The following is an example of specifying offset.

```
gSQL> SELECT department_id, employee_id, manager_id,
             LAG( manager_id, 2 ) OVER ( ORDER BY employee_id ) AS lag
        FROM employees
       WHERE department_id = 90;
DEPARTMENT_ID EMPLOYEE_ID MANAGER_ID  LAG
------------- ----------- ---------- ----
           90         100       null null
           90         101        100 null
           90         102        100 null
3 rows selected.
```

The following is an example of specifying offset and default.

```
gSQL> SELECT department_id, employee_id, manager_id,
             LAG( manager_id, 2, 0 ) OVER ( ORDER BY employee_id ) AS lag
        FROM employees
       WHERE department_id = 90;
DEPARTMENT_ID EMPLOYEE_ID MANAGER_ID  LAG
------------- ----------- ---------- ----
           90         100       null    0
           90         101        100    0
           90         102        100 null
3 rows selected.
```

17.77 LAST() OVER

Syntax

```
aggregation_function KEEP ( DENSE_RANK LAST ORDER BY <sort specification list> ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function LAST sorts the sort specification list written in *order by* within KEEP clause, then returns the aggregation function value of rows whose DENSE_RANK is the last.

aggregation_functions are AVG, COUNT, COUNT(*), SUM, MAX, MIN, STDDEV, VARIANCE.

order by is not available within the window clause.
window frame is not available.

Example

```
gSQL> SELECT department_id, salary,
           DENSE_RANK() OVER ( ORDER BY department_id ) AS "DENSE_RANK",
           MAX( salary ) KEEP ( DENSE_RANK LAST ORDER BY department_id ) OVER () AS "MAX_LAST"
  FROM employees
 WHERE department_id BETWEEN 90 AND 100;
DEPARTMENT_ID SALARY DENSE_RANK MAX_LAST
------------- ------ ---------- --------
           90  24000          1    12000
           90  17000          1    12000
           90  17000          1    12000
          100  12000          2    12000
          100   9000          2    12000
          100   8200          2    12000
          100   7700          2    12000
          100   7800          2    12000
```

```
          100   6900          2    12000
9 rows selected.
```

17.78 LAST_DAY

Syntax

```
LAST_DAY( date )
```

Description

It returns the last day of the month which is included in date.

The date argument data type can be DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE.
The return type is always DATE regardless of the date argument data type.
If date is NULL, then it returns NULL.

Example

```
gSQL> SELECT
      LAST_DAY( TO_DATE( '2012-07-10', 'YYYY-MM-DD' ) ) AS RESULT FROM DUAL;
RESULT
----------
2012-07-31
1 row selected.
```

17.79 LAST_IDENTITY_VALUE

Syntax

```
LAST_IDENTITY_VALUE()
```

Description

It is the recent value automatically created for an identity column in the current session, and the result type is NATIVE_BIGINT.

If there is not an automatically created value, then it returns null.

This function is similar to @@IDENTITY of MS-SQL and LAST_INSERT_ID() of MySQL. Be cautious when using it because the last altered table determines the value when performing DML for multiple tables as follows.

```
gSQL> INSERT INTO t1(name) VALUES ( 'leekmo' );
1 row created.
gSQL> SELECT LAST_IDENTITY_VALUE() FROM dual;
LAST_IDENTITY_VALUE()
---------------------
                   12
1 row selected.
gSQL> INSERT INTO t2(name) VALUES ( 'leekmo' );
1 row created.
gSQL> SELECT LAST_IDENTITY_VALUE() FROM dual;
LAST_IDENTITY_VALUE()
---------------------
                    2
1 row selected.
```

To obtain an identity column value created when performing the INSERT, use **INSERT INTO name RETURNING .. INTO** statement as follows.

```
gSQL> CREATE TABLE t1 ( id INTEGER GENERATED BY DEFAULT AS IDENTITY, name VARCHAR(32) );
Table created.
gSQL> \var v1 integer
```

```
gSQL> INSERT INTO t1(name) VALUES ( 'leekmo' ) RETURN id INTO :v1;
V1
--
 1
1 row created.
```

Example

The following is an example of using LAST_IDENTITY_VALUE() function.

```
gSQL> CREATE TABLE t1 ( id   INTEGER GENERATED BY DEFAULT AS IDENTITY,
                        name VARCHAR(32) );
Table created.
gSQL> COMMIT;
Commit complete.
```

- There is not an identity value created in the current session.

```
gSQL> SELECT LAST_IDENTITY_VALUE() FROM dual;
LAST_IDENTITY_VALUE()
---------------------
                null
1 row selected.
```

- Identity value (1) is automatically created.

```
gSQL> INSERT INTO t1(name) VALUES ( 'leekmo' );
1 row created.
```

- Result: 1

```
gSQL> SELECT LAST_IDENTITY_VALUE() FROM dual;
LAST_IDENTITY_VALUE()
---------------------
                   1
1 row selected.
```

- Identity value (2) is automatically created as a default value.

```
gSQL> UPDATE t1 SET id = DEFAULT;
1 row updated.
```

- Result: 2

```
gSQL> SELECT LAST_IDENTITY_VALUE() FROM dual;
LAST_IDENTITY_VALUE()
---------------------
                    2
1 row selected.
```

- The user input value does not automatically create an identity value.

```
INSERT INTO t1 VALUES ( 100, 'jhkim' );
1 row updated.
```

- Result: 2

```
SELECT LAST_IDENTITY_VALUE() FROM dual;
LAST_IDENTITY_VALUE()
---------------------
                    2
1 row selected.
```

17.80 LAST_VALUE() OVER

Syntax

```
LAST_VALUE ( expr ) [ RESPECT NULLS ¦ IGNORE NULLS ] OVER < window name or specification >
LAST_VALUE ( expr [ RESPECT NULLS ¦ IGNORE NULLS ] ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function LAST_VALUE returns the last value of expr.

RESPECT NULLS returns the last value of rows including NULL.
IGNORE NULLS returns the last value of row except for NULL.
If it is not specified, the default value is RESPECT NULLS.

order by is not available in window clause.
window frame is not available.

Example

```
gSQL> SELECT min_price AS "MIN_PRICE"
             , LAST_VALUE( min_price ) OVER ( ORDER BY min_price ) AS "LAST_VALUE"
        FROM product_information
       WHERE supplier_id = 102050;
MIN_PRICE LAST_VALUE
--------- ----------
       73         73
      247        247
      731        731
     null       null
     null       null
5 rows selected.
```

The following is an example of specifying IGNORE NULLS in null_treatment.

```
gSQL> SELECT min_price AS "MIN_PRICE"
             , LAST_VALUE( min_price IGNORE NULLS ) OVER ( ORDER BY min_price ) AS
"LAST_VALUE"
        FROM product_information
       WHERE supplier_id = 102050;
MIN_PRICE LAST_VALUE
--------- ----------
       73         73
      247        247
      731        731
     null        731
     null        731
5 rows selected.
```

17.81 LEAD() OVER

Syntax

```
LEAD ( expr [, offset [, default ] ] ) [ RESPECT NULLS ¦ IGNORE NULLS ] OVER < window name or specification >
LEAD ( expr [ RESPECT NULLS ¦ IGNORE NULLS ] [, offset [, default ] ] ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function LEAD returns the row value behind from the current row as far as offset.
If offset is out of window range, then it returns the default value.

If offset and default values are not specified, then it is set to the default value.
The default value of offset is 1 and the the default value of default is NULL.

RESPECT NULLS returns the row value behind as far as offset including NULL.
IGNORE NULLS returns the row value behind as far as offset except for NULL.
If it is not specified, the default value is RESPECT NULLS.

window frame is not available.

Example

```
gSQL> SELECT department_id, employee_id, manager_id,
             LEAD( manager_id ) OVER ( ORDER BY employee_id DESC ) AS lead
        FROM employees
       WHERE department_id = 90;
DEPARTMENT_ID EMPLOYEE_ID MANAGER_ID LEAD
------------- ----------- ---------- ----
           90         102        100  100
           90         101        100 null
           90         100       null null
```

```
3 rows selected.
```

The following is an example of specifying offset.

```
gSQL> SELECT department_id, employee_id, manager_id,
             LEAD( manager_id, 2 ) OVER ( ORDER BY employee_id DESC ) AS lead
        FROM employees
       WHERE department_id = 90;
DEPARTMENT_ID EMPLOYEE_ID MANAGER_ID LEAD
------------- ----------- ---------- ----
           90         102        100 null
           90         101        100 null
           90         100       null null
3 rows selected.
```

The following is an example of specifying offset and default.

```
gSQL> SELECT department_id, employee_id, manager_id,
             LEAD( manager_id, 2, 0 ) OVER ( ORDER BY employee_id DESC ) AS lead
        FROM employees
       WHERE department_id = 90;
DEPARTMENT_ID EMPLOYEE_ID MANAGER_ID LEAD
------------- ----------- ---------- ----
           90         102        100 null
           90         101        100    0
           90         100       null    0
3 rows selected.
```

17.82 LEAST

Syntax

```
LEAST( expr1 [, expr2, ... exprn ] )
```

Description

It returns the smallest value among received expr arguments.

If any of expr is NULL, the result is NULL.

The result type is determined according to the data type of expr1 (the first expr).
If the data type of expr1 (the first expr) is a character type and a numeric type then it becomes the type in cluding the range of expr1, ..., exprN each.
If all of expr1, ..., exprN is described in CHAR type, then all exprs are compared in VARCHAR type and the result type is VARCHAR.

Example

```
gSQL> SELECT LEAST( 100, 0, 200, 150, 1 ) AS RESULT FROM DUAL;
RESULT
------
     0
1 row selected.
```

17.83 LENGTH

Syntax

```
LENGTH( str )
```

Description

It is an alias of **CHAR_LENGTH**.

Example

Multi byte character set: (e.g.UTF8)

```
gSQL> SELECT LENGTH( 'αβ-SUMMER' ) AS RESULT FROM DUAL;
     RESULT
     ---------------
               9
    1 row selected.
```

17.84 LENGTHB

Syntax

```
LENGTHB( str )
```

Description

It is an alias of **OCTET_LENGTH**.
For more information, refer to **BYTE_LENGTH**.

Example

- Multi byte character set (e.g.UTF8): 1 byte character

```
gSQL> SELECT LENGTHB( 'OCTET_LENGTH' ) AS RESULT_1BYTE_CHARACTERS
        FROM DUAL;
RESULT_1BYTE_CHARACTERS
-----------------------
                     12
1 row selected.
```

- Multi byte character set (e.g.UTF8): 2 byte character

```
gSQL> SELECT LENGTHB( 'αβ' ) AS RESULT_2BYTE_CHARACTERS FROM DUAL;
RESULT_2BYTE_CHARACTERS
-----------------------
                      4
1 row selected.
```

17.85 LISTAGG() OVER

Syntax

```
LISTAGG( str [, delimiter] ) WITHIN GROUP ( ORDER BY <sort specification list> ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function LISTAGG connects str in its order sorted within each group.

Only PARTITION BY clause is available in OVER() clause of LISTAGG.
It divides the query result sets into groups by using OVER() clause.

It sorts the records within the group with WITHIN GROUP (ORDER BY <sort specification list>).

It connects str in the order of the record sorted within each group.
If str is NULL, then it is excluded.

A delimiter is str connection delimiter, and if it is omitted the default value is NULL.

str can be a character string or a binary string.
If str is a character string, then the result type is varchar.
If str is a binary string, then the result type is varbinary.

Example

```
gSQL>
SELECT regionkey,
       name,
       LISTAGG( name ) WITHIN GROUP ( ORDER BY nationkey )
                       OVER ( PARTITION BY regionkey )
       AS "LISTAGG( name ) RESULT",
       LISTAGG( name, ', ' ) WITHIN GROUP ( ORDER BY nationkey )
```

```
                           OVER ( PARTITION BY regionkey )
       AS "LISTAGG( name, ', ' ) RESULT"
  FROM nation;
REGIONKEY NAME    LISTAGG( name ) RESULT    LISTAGG( name, ', ' ) RESULT
--------- ------- ------------------------- ----------------------------
        1 BRAZIL  BRAZILCANADAPERU          BRAZIL, CANADA, PERU
        1 CANADA  BRAZILCANADAPERU          BRAZIL, CANADA, PERU
        1 PERU    BRAZILCANADAPERU          BRAZIL, CANADA, PERU
        1 null    BRAZILCANADAPERU          BRAZIL, CANADA, PERU
        2 null    INDIAJAPANCHINAVIETNAM    INDIA, JAPAN, CHINA, VIETNAM
        2 INDIA   INDIAJAPANCHINAVIETNAM    INDIA, JAPAN, CHINA, VIETNAM
        2 null    INDIAJAPANCHINAVIETNAM    INDIA, JAPAN, CHINA, VIETNAM
        2 null    INDIAJAPANCHINAVIETNAM    INDIA, JAPAN, CHINA, VIETNAM
        2 JAPAN   INDIAJAPANCHINAVIETNAM    INDIA, JAPAN, CHINA, VIETNAM
        2 CHINA   INDIAJAPANCHINAVIETNAM    INDIA, JAPAN, CHINA, VIETNAM
        2 null    INDIAJAPANCHINAVIETNAM    INDIA, JAPAN, CHINA, VIETNAM
        2 VIETNAM INDIAJAPANCHINAVIETNAM    INDIA, JAPAN, CHINA, VIETNAM
        3 EGYPT   EGYPTIRANIRAQ             EGYPT, IRAN, IRAQ
        3 IRAN    EGYPTIRANIRAQ             EGYPT, IRAN, IRAQ
        3 IRAQ    EGYPTIRANIRAQ             EGYPT, IRAN, IRAQ
15 rows selected.
```

17.86 LN

Syntax

```
LN( num )
```

Description

It returns the natural logarithm value of num.
num should be a value which is bigger than 0.
If num is NULL, then it returns NULL.

Example

```
gSQL> SELECT LN( 2.71828182845905 ) AS RESULT FROM DUAL;
RESULT
------
     1
1 row selected.
```

17.87 LNNVL

Syntax

```
LNNVL( expr )
```

Description

Logical Not Null VaLue (LNNVL) function is similar to NOT logical operator, but the difference is that it ret urns TRUE as in the following example when the input value is null.

Example

```
gSQL> SELECT c1, c2, (c1 = c2), NOT(c1 = c2), LNNVL(c1 = c2) FROM t1;
C1   C2 (C1 = C2) NOT(C1 = C2) LNNVL(C1 = C2)
-- ---- --------- ------------ --------------
 1    1 TRUE      FALSE        FALSE
 1    2 FALSE     TRUE         TRUE
 1 null null      null         TRUE
3 rows selected.
```

17.88 LOCALTIME

Syntax

```
LOCALTIME [()]
STATEMENT_LOCALTIME()
```

Description

The current TIME WITHOUT TIME ZONE type value based on the session time is obtained.

LOCALTIME is an SQL standard function.

The differences among the functions to obtain the current time are as follows.

- TRANSACTION_LOCALTIME(): All time values in the transaction are same.
- LOCALTIME, STATEMENT_LOCALTIME(): All time values in an SQL statement are same.
- CLOCK_LOCALTIME(): Whenever the function is called, the current time value is obtained.

Example

All rows have the same value.

```
gSQL> SELECT LOCALTIME FROM t1;
LOCALTIME
---------------
16:17:08.592459
16:17:08.592459
16:17:08.592459
3 rows selected.
```

17.89 LOCALTIMESTAMP

Syntax

```
LOCALTIMESTAMP [()]
STATEMENT_LOCALTIMESTAMP()
```

Description

The current TIMESTAMP WITHOUT TIME ZONE type value based on the session time is obtained.

LOCALTIMESTAMP is an SQL standard function.

The differences among the functions to obtain the current TIMESTAMP are as follows.
- TRANSACTION_LOCALTIMESTAMP(): All TIMESTAMP values in the transaction are same.
- LOCALTIMESTAMP, STATEMENT_LOCALTIMESTAMP(): All TIMESTAMP values in an SQL statement ar e same.
- CLOCK_LOCALTIMESTAMP(): Whenever the function is called, the current timestamp value is obtained.

Example

All rows have the same value.

```
gSQL> SELECT LOCALTIMESTAMP FROM t1;
LOCALTIMESTAMP
--------------------------
2013-12-12 16:21:51.790614
2013-12-12 16:21:51.790614
2013-12-12 16:21:51.790614
3 rows selected.
```

17.90 LOCAL_GROUP_ID

Syntax

```
LOCAL_GROUP_ID()
```

Description

It returns a cluster group ID for a server which processes a query from a user.


It is a valid information in a cluster system.

Example

All rows have the same value.

```
gSQL> SELECT LOCAL_GROUP_ID() FROM DUAL;
LOCAL_GROUP_ID()
----------------
               1
1 row selected.
```

17.91 LOCAL_GROUP_NAME

Syntax

```
LOCAL_GROUP_NAME()
```

Description

It returns a cluster group name for a server which processes a query from a user.


It is a valid information in a cluster system.

Example

All rows have the same value.

```
gSQL> SELECT LOCAL_GROUP_NAME() FROM DUAL;
LOCAL_GROUP_NAME()
------------------
G1
1 row selected.
```

17.92 LOCAL_MEMBER_ID

Syntax

```
LOCAL_MEMBER_ID()
```

Description

It returns a cluster member ID for a server which processes a query from a user.


It is a valid information in a cluster system.

Example

All rows have the same value.

```
gSQL> SELECT LOCAL_MEMBER_ID() FROM DUAL;
LOCAL_MEMBER_ID()
-----------------
                1
1 row selected.
```

17.93 LOCAL_MEMBER_NAME

Syntax

```
LOCAL_MEMBER_NAME()
```

Description

It returns a cluster member name for a server which processes a query from a user.


It is a valid information in a cluster system.

Example

All rows have the same value.

```
gSQL> SELECT LOCAL_MEMBER_NAME() FROM DUAL;
LOCAL_MEMBER_NAME()
-------------------
G1N1
1 row selected.
```

17.94 LOG

Syntax

```
LOG( num2 )
LOG( num1, num2 )
```

Description

It returns the logarithm of num2 in the num1 base.
If num1 is omitted, it returns the logarithm value whose base is 10.

num1 should be a positive number except 1 and 0, and num2 should be a positive number.

If num1 or num2 is NULL, then it returns NULL.

Example

```
gSQL> SELECT LOG( 100 ) AS RESULT1, LOG( 4, 16 ) AS RESULT2 FROM DUAL;
RESULT1 RESULT2
------- -------
      2       2
1 row selected.
```

17.95 LOGON_USER

Syntax

```
LOGON_USER()
```

Description

It returns the logged-in user.

The user information is managed in three types as follows.

- Logon user: It is a user who performed login, and it is maintained until the connection is closed.
- Session user: It is as same as the first logon user, but it can be changed using the SET SESSION AUTH ORIZATION statement.
- Current user: It is generally as same as the session user, but it is temporarily changed internally in syst em to control access when using the PSM, view.
    - The session user and current user is similar to the difference between the unix system's real user and the effective user.

Example

```
% gsql test test
gSQL> SELECT LOGON_USER() AS result FROM DUAL;
RESULT
------
TEST
1 row selected.
```

17.96 LOWER

Syntax

```
LOWER( str )
```

Description

It returns lowercases of str.

The str argument data type can be a character type such as CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING.
If str is NULL, the result is also NULL.

The return type is the same datatype as the str argument.

Example

```
gSQL> SELECT LOWER( 'SPRING' ) AS RESULT FROM DUAL;
RESULT
------
spring
1 row selected.
```

17.97 LPAD

Syntax

```
LPAD( str, length, [, fill] )
```

Description

It adds character string fill to the left side of str until the string length becomes length, then returns the re sult.

The str argument data type can be character type such as CHARACTER, CHARACTER VARYING, CHARAC TER LONG VARYING, and a binary character type such as BINARY, BINARY VARYING, BINARY LONG VAR YING.

The length argument is numeric type.

length means the number of characters, and its maximum range is the maximum precision of the result t ype.
If fill is omitted, a white space is added.
If str is longer than the length, it cuts the str as long as the length, then returns it.
If any of str, length, fill is NULL, the result is also NULL.
If length is 0 or a negative number, the result is NULL.

The following table describes the result types.

Table 17-13 Result type of LPAD

str type	Result type
CHAR or VARCHAR	VARCHAR
LONG VARCHAR	LONG VARCHAR
BINARY or VARBINARY	VARBINARY
LONG VARBINARY	LONG VARBINARY

Example

```
gSQL> SELECT LPAD('AA', 5) AS RESULT1,
             LPAD('AA', 5, 'X' ) AS RESULT2,
             LPAD('AA', 1 ) AS RESULT3
      FROM DUAL;
RESULT1 RESULT2 RESULT3
------- ------- -------
   AA   XXXAA   A
1 row selected.
```

17.98 LTRIM

Syntax

```
LTRIM( trim_source [, trim_character ] )
```

Description

It removes the matching characters by comparing from the left side of trim_character in trim_source until the matching character does not exist. Then it returns the result.

The data type of trim_character argument and trim_source argument can be a character type such as CH ARACTER, CHARACTER VARYING, CHARACTER LONG VARYING, and a binary character type such as BIN ARY, BINARY VARYING, BINARY LONG VARYING.

If any of trim_character, trim_source is NULL, the result is NULL.
If trim_character is omitted, a single blank space (' ') is specified by default.

The following table describes the result types.

Table 17-14 Result type of LTRIM

trim_source, trim_character type	Result type
CHAR or VARCHAR	VARCHAR
LONG VARCHAR	LONG VARCHAR
BINARY or VARBINARY	VARBINARY
LONG VARBINARY	LONG VARBINARY

Example

```
gSQL> SELECT LTRIM( '_____LTRIM', '_' ) AS RESULT FROM DUAL;
RESULT
------
LTRIM
1 row selected.
```

17.99 MAX

Syntax

```
MAX( [ ALL ¦ DISTINCT ] expr )
```

Description

It is an aggregate function and the maximum value among rows' exprs is obtained.

If ALL is explicitly specified, aggregation is executed for all values.
If DISTINCT is explicitly specified, aggregation is executed for the values which exclude duplicate values.
If ALL or DISTINCT is not explicitly specified, it is processed in the same way as when ALL is specified.

MAX function returns the same result without being affected by the ALL and DISTINCT.

Example

```
gSQL> SELECT MAX(c1) FROM t1;
MAX(C1)
-------
      3
1 row selected.
```

17.100 MAX() OVER

Syntax

```
MAX ( expr ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function MAX obtains the maximum value of exprs.
NULL is excluded from the calculation.

Example

```
gSQL> SELECT product_id AS "PRODUCT_ID", min_price AS "MIN_PRICE"
             , MAX( min_price ) OVER ( ORDER BY product_id ) AS "MAX"
        FROM product_information
       WHERE supplier_id = 102050;
PRODUCT_ID MIN_PRICE  MAX
---------- --------- ----
      1769      null null
      1770        73   73
      2378       247  247
      2382       731  731
      3355      null  731
5 rows selected.
```

17.101 MEDIAN() OVER

Syntax

```
MEDIAN ( expr ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function MEDIAN returns the median value of row.
NULL is excluded from the calculation.

order by is not available within the window clause.
window frame is not available.

Example

```
gSQL> SELECT supplier_id, min_price
             , MEDIAN( min_price ) OVER ( PARTITION BY supplier_id ) AS "MEDIAN"
        FROM product_information
       WHERE supplier_id = 102050;
SUPPLIER_ID MIN_PRICE MEDIAN
----------- --------- ------
     102050        73    247
     102050       247    247
     102050       731    247
     102050      null    247
     102050      null    247
5 rows selected.
```

17.102 MIN

Syntax

```
MIN( [ ALL ¦ DISTINCT ] expr )
```

Description

It is an aggregate function and the minimum value among rows' exprs is obtained.

If ALL is explicitly specified, aggregation is executed for all values.
If DISTINCT is explicitly specified, aggregation is executed for the values which exclude duplicate values.
If ALL or DISTINCT is not explicitly specified, it is processed in the same way as when ALL is specified.

MIN function returns the same result without being affected by the ALL and DISTINCT.

Example

```
gSQL> SELECT MIN(c1) FROM t1;
MIN(C1)
-------
      1
1 row selected.
```

17.103 MIN() OVER

Syntax

```
MIN ( expr ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function MIN obtains the minimum value of exprs.
NULL is excluded from the calculation.

Example

```
gSQL> SELECT product_id AS "PRODUCT_ID", min_price AS "MIN_PRICE"
             , MIN( min_price ) OVER ( ORDER BY product_id DESC ) AS "MIN"
        FROM product_information
       WHERE supplier_id = 102050;
PRODUCT_ID MIN_PRICE  MIN
---------- --------- ----
      3355      null null
      2382       731  731
      2378       247  247
      1770        73   73
      1769      null   73
5 rows selected.
```

17.104 MOD

Syntax

```
MOD( num1, num2 )
```

Description

It divides num1 by num2, and returns the remainder.
The num1 argument and num2 argument can be a numeric data type.
If num2 is 0, an error is returned.
If the num1 argument or num2 argument is NULL, then NULL is returned.

Example

```
gSQL> SELECT MOD(5, 4) AS RESULT1, MOD(-5, 4) AS RESULT2 FROM DUAL;
RESULT1 RESULT2
------- -------
      1      -1
1 row selected.
```

17.105 MONTHS_BETWEEN

Syntax

```
MONTHS_BETWEEN( date1, date2 )
```

Description

MONTHS_BETWEEN returns the number of months of which days between date2 and date1 are devided by 31.

If date1 or date2 is NULL, then the result is also NULL.
The date1 argument and date2 argument can be DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE type.

The result type is NUMBER.

If the same date (e.g. 2014-01-15 and 2014-02-15), or the last day of the month (e.g. 2014-08-31 and 2014-09-30) is included both in date1 and date2, then it returns the integer result regardless of the agreement of timestamp section (if it exists).

Example

```
gSQL> SELECT
        MONTHS_BETWEEN('2018-01-18', '2018-01-17')
  FROM DUAL;
MONTHS_BETWEEN('2018-01-18', '2018-01-17')
------------------------------------------
                     3.225806451612903E-2
1 row selected.
gSQL> SELECT
        MONTHS_BETWEEN('2018-02-17', '2018-01-17')
  FROM DUAL;
MONTHS_BETWEEN('2018-02-17', '2018-01-17')
------------------------------------------
```

```
                                        1
1 row selected.
gSQL> SELECT
        MONTHS_BETWEEN('2018-02-28', '2018-01-31')
  FROM DUAL;
MONTHS_BETWEEN('2018-02-28', '2018-01-31')
------------------------------------------
                                        1
1 row selected.
```

17.106 NEXT_DAY

Syntax

```
NEXT_DAY( date, day )
```

Description

It obtains a date of the day (day of week) which comes first after the given date (an argument).

The second day argument can be a string or a number which indicates the day.
- String: SUNDAY ~ SATURDAY or SUN ~ SAT
- Number: 1 (sunday) ~ 7 (saturday)

If any of the input argument is NULL, the result is also NULL.

The return type is always DATE regardless of the input type of the date.
The hour, minute and second of the result value returns the same hour, minute and second of the input a rgument date.

Example

- 2020-08-11 is Tuesday.

```
gSQL> SELECT NEXT_DAY( TO_DATE( '2020-08-11', 'YYYY-MM-DD'),
                       'SUNDAY' ) AS RESULT1
      FROM DUAL;
RESULT1
----------
2020-08-16
1 row selected.
gSQL> SELECT NEXT_DAY( TO_DATE( '2020-08-11', 'YYYY-MM-DD' ),
                       'SUN' ) AS RESULT1
FROM DUAL;
RESULT1
----------
2020-08-16
```

```
1 row selected.
gSQL> SELECT NEXT_DAY( TO_DATE( '2020-08-11', 'YYYY-MM-DD' ),
                       1 ) AS RESULT1
FROM DUAL;
RESULT1
----------
2020-08-16
1 row selected.
gSQL> SELECT TO_CHAR( NEXT_DAY( TO_DATE( '2020-08-11', 'YYYY-MM-DD' ),
                                'SUNDAY' ),
                     'YYYY-MM-DD HH24:MI:SS' ) AS RESULT1
FROM DUAL;
RESULT1
-------------------
2020-08-16 00:00:00
1 row selected.
```

17.107 NEXTVAL

Syntax

```
seq_name.NEXTVAL
NEXTVAL( seq_name )
NEXT VALUE FOR seq_name
```

Description

It obtains the next value of the sequence object.

Example

```
gSQL> CREATE SEQUENCE seq;
Sequence created.
gSQL> COMMIT;
Commit complete.
gSQL> SELECT seq.NEXTVAL FROM dual;
SEQ.NEXTVAL
-----------
          1
1 row selected.
gSQL> SELECT NEXTVAL( seq ) FROM dual;
NEXTVAL( SEQ )
--------------
             2
1 row selected.
gSQL> SELECT NEXT VALUE FOR seq FROM dual;
NEXT VALUE FOR SEQ
------------------
                 3
1 row selected.
```

17.108 NTH_VALUE() OVER

Syntax

```
NTH_VALUE ( expr, n ) [ FROM { FIRST ¦ LAST } ][ { RESPECT ¦ IGNORE } NULLS ] OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function NTH_VALUE returns the expr value of the n-th row.
If the number of window's rows are less than n, then it returns NULL.

The n argument can be a numeric type or types which can be converted to number.

FROM FIRST points to the n-th row from the first row.
FROM LAST points to the n-th row from the last row.
If it is not specified, the default value is FROM FIRST.

RESPECT NULLS returns the value of the n-th row including NULL.
IGNORE NULLS returns the value of the n-th row except for NULL.
If it is not specified, the default value is RESPECT NULLS.

Example

```
gSQL> SELECT product_id, min_price,
             NTH_VALUE( min_price, 2 ) OVER ( ORDER BY product_id ) AS nth_value
      FROM product_information
      WHERE supplier_id = 102050;
PRODUCT_ID MIN_PRICE NTH_VALUE
---------- --------- ---------
      1769      null      null
      1770        73        73
      2378       247        73
      2382       731        73
```

```
      3355      null        73
5 rows selected.
```

The following is an example of when FROM LAST is specified.

```
gSQL> SELECT product_id, min_price,
              NTH_VALUE( min_price, 2 ) FROM LAST OVER ( ORDER BY product_id ) AS nth_value
      FROM product_information
      WHERE supplier_id = 102050;
PRODUCT_ID MIN_PRICE NTH_VALUE
---------- --------- ---------
      1769      null      null
      1770        73      null
      2378       247        73
      2382       731       247
      3355      null       731
5 rows selected.
```

The following is an example of when IGNORE NULLS is specified.

```
gSQL> SELECT product_id, min_price,
              NTH_VALUE( min_price, 2 ) IGNORE NULLS OVER ( ORDER BY product_id ) AS nth_value
      FROM product_information
      WHERE supplier_id = 102050;
PRODUCT_ID MIN_PRICE NTH_VALUE
---------- --------- ---------
      1769      null      null
      1770        73      null
      2378       247       247
      2382       731       247
      3355      null       247
5 rows selected.
```

17.109 NTILE() OVER

Syntax

```
NTILE( expr ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function NTILE returns the bucket number corresponding to each row.

The bucket number is a consecutive integer starting from 1, and the number of buckets are as same as th e number of expr.
If the number of buckets are bigger than that of rows, a bucket is given to each row and other buckets re mains empty.

expr should be a positive constant. If expr is not a fixed number, then expr should be the target of *windo w partition by*.

window frame is not available.

Example

```
gSQL> SELECT department_id, salary,
             NTILE(3) OVER ( ORDER BY salary ) AS ntile
      FROM employees
      WHERE department_id = 60;
DEPARTMENT_ID SALARY NTILE
------------- ------ -----
           60   4200     1
           60   4800     1
           60   4800     2
           60   6000     2
           60   9000     3
5 rows selected.
```

The following is an example of when the number of buckets (the number of expr) is bigger than the num ber of rows.

```
gSQL> SELECT department_id, salary,
             NTILE(6) OVER ( ORDER BY salary ) AS ntile
      FROM employees
      WHERE department_id = 60;
DEPARTMENT_ID SALARY NTILE
------------- ------ -----
           60   4200     1
           60   4800     2
           60   4800     3
           60   6000     4
           60   9000     5
5 rows selected.
```

17.110 NULLIF

Syntax

```
NULLIF( expr1, expr2 )
```

Description

If expr1 is equal to expr2, it returns NULL. If it is not equal it returns expr1 which is the first argument.

If the data types of expr1 and expr2 are different, the result type is determined by **Result Type Combinati on Rule**.

NULLIF can be expressed by using CASE as follows.

- NULLIF(expr1, expr2)

```
CASE WHEN expr1 = expr2 THEN NULL
       ELSE expr1
  END
```

Example

```
gSQL> SELECT NULLIF( 'SUN', 'SUN' ) AS RESULT1,
             NULLIF( 'SUN', 'MOON' ) AS RESULT2
       FROM DUAL;
RESULT1 RESULT2
------- -------
null    SUN
1 row selected.
```

17.111 NUMTODSINTERVAL

Syntax

```
NUMTODSINTERVAL( num, interval_indicator )
```

Description

It converts the number in interval_indicator unit to interval day to second type, then returns it.

The argument *number* is a numeric type.

The argument interval_indicator is the same character type as that of CHAR and VARCHAR, and it should be one of 'DAY', 'HOUR', 'MINUTE', 'SECOND' which is case insensitive.

If any argument is NULL, then NULL is returned as a result.

interval day(6) to second(6) type is returned as a result, and a user can not arbitrarily modify the precision. If the leading precision of the converted result exceeds the default precision, then an error is returned. If t he fraction precision exceeds the default precision, then the rounded value is returned as a result.

Example

- It converts *1 Day* to interval day to second type.

```
gSQL> SELECT NUMTODSINTERVAL(1, 'DAY') FROM DUAL;
NUMTODSINTERVAL(1, 'DAY')
-------------------------
+000001 00:00:00.000000
1 row selected.
```

- It converts *36 Hour* to interval day to second type.

```
gSQL> SELECT NUMTODSINTERVAL(36, 'HOUR') FROM DUAL;
NUMTODSINTERVAL(36, 'HOUR')
---------------------------
+000001 12:00:00.000000
1 row selected.
```

- It converts *1530 Minute* to interval day to second type.

```
gSQL> SELECT NUMTODSINTERVAL(1530, 'MINUTE') FROM DUAL;
NUMTODSINTERVAL(1530, 'MINUTE')
------------------------------
+000001 01:30:00.000000
1 row selected.
```

- It converts *90100.1234567 Second* to interval day to second type.

```
gSQL> SELECT NUMTODSINTERVAL(90100.1234567, 'SECOND') FROM DUAL;
NUMTODSINTERVAL(90100.1234567, 'SECOND')
---------------------------------------
+000001 01:01:40.123457
1 row selected.
```

17.112 NUMTOYMINTERVAL

Syntax

```
NUMTOYMINTERVAL( num, interval_indicator )
```

Description

It converts the number in interval_indicator unit to interval year to month type, then returns it.

The argument *number* is a numeric type.

The argument interval_indicator is the same character type as that of CHAR and VARCHAR, and it should be one of 'YEAR', 'MONTH' which is case insensitive.

If any argument is NULL, then NULL is returned as a result.

interval year(6) to month type is returned as a result, and a user can not arbitrarily modify the precision. If the leading precision of the converted result exceeds the default precision, then an error is returned.

Example

- It converts *1 Year* to interval year to month type.

```
gSQL> SELECT NUMTOYMINTERVAL(1, 'YEAR') FROM DUAL;
NUMTOYMINTERVAL(1, 'YEAR')
--------------------------
+000001-00
1 row selected.
```

- It converts *13.5 Month* to interval year to month type.

```
gSQL> SELECT NUMTOYMINTERVAL(13.5, 'MONTH') FROM DUAL;
NUMTOYMINTERVAL(13.5, 'MONTH')
------------------------------
+000001-02
1 row selected.
```

17.113 NVL

Syntax

```
NVL( expr1, expr2 )
```

Description

If expr1 is not NULL, then it returns expr1. If expr1 is NULL, it returns expr2.

The result type is determined according to the data type of expr1.
If NULL is described in expr1, then the result type is determined according to the data type of expr2.
If the data type of expr1 is a character type and a numeric type then it becomes the type including the ra nge of expr1 and expr2 each.
If the data type of both expr1 and expr2 is CHAR type, then the result type is VARCHAR.

Example

```
gSQL> SELECT I1, NVL( I1, 0 ) FROM T1;
  I1 NVL( I1, 0 )
---- ------------
   1            1
null            0
2 rows selected.
```

17.114 NVL2

Syntax

```
NVL2( expr1, expr2, expr3 )
```

Description

If expr1 is not null, then it returns expr2. If expr1 is NULL, it returns expr3.

The result type is determined according to the data type of expr2.
If NULL is described in expr2, then the result type is determined according to the data type of expr3.
If the data type of expr2 is a character type and a numeric type then it becomes the type including the ra nge of expr2 and expr3 each.
If the data type of both expr2 and expr3 is CHAR type, then the result type is VARCHAR.

Example

```
gSQL> SELECT I1, NVL2( I1, I1 * 1000, 0 ) FROM T1;
  I1 NVL2( I1, I1 * 1000, 0 )
---- ------------------------
   1                    1000
null                       0
2 rows selected.
```

17.115 OCTET_LENGTH

Syntax

```
OCTET_LENGTH( str )
BYTE_LENGTH( str )
LENGTHB( str )
```

Description

It returns the number of bytes in str.

The str argument data type can be a character type such as CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING, or a binary character type such as BINARY, BINARY VARYING, BINARY LONGVARYING.

If the str data type is CHARACTER, the white spaces are included in the calculation.
If str is NULL, the result is also NULL.

It is an alias of **BYTE_LENGTH** and **LENGTHB**.

Example

- Multi byte character set (e.g. UTF8): 1 byte character

```
gSQL> SELECT OCTET_LENGTH( 'OCTET_LENGTH' ) AS RESULT_1BYTE_CHARACTERS
        FROM DUAL;
RESULT_1BYTE_CHARACTERS
-----------------------
                     12
1 row selected.
```

- Multi byte character set (e.g. UTF8): 2 byte character

```
gSQL> SELECT OCTET_LENGTH( 'αβ' ) AS RESULT_2BYTE_CHARACTERS FROM DUAL;
RESULT_2BYTE_CHARACTERS
```

```
------------------------
                       4

1 row selected.
```

17.116 OVERLAY

Syntax

```
OVERLAY( str1 PLACING str2 FROM start_position  [ FOR string_length ] )
```

Description

It overlays the characters in the range between str1's start_position and string_lenght with str2.

The data types of str1 argument and str2 argument can be a character type such as CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING, or a binary character type such as BINARY, BINARY VARYING, BINARY LONG VARYING
The start_position argument and string_length argument can be numeric data type.

- OVERLAY function has the following result.
  - When FOR is specified.
    SUBSTRING(str1 FROM 1 FOR (start_position - 1))
    || str2
    || SUBSTRING(str1 FROM (start_position + string_length)
  - When FOR is omitted.
    SUBSTRING(str1 FROM 1 FOR (start_position - 1))
    || str2
    || SUBSTRING(str1 FROM (start_position + CHAR_LENGTH(str2))

For more information, refer to **SUBSTRING**.

The following table describes the result types.

Table 17-15 Result type of OVERLAY

str1, str2 types	Result type
CHAR or VARCHAR	VARCHAR
LONG VARCHAR	LONG VARCHAR
BINARY or VARBINARY	VARBINARY
LONG VARBINARY	LONG VARBINARY

Example

```
gSQL> SELECT
         OVERLAY( 'RESULT_OF_XXX_FUNC' PLACING 'OVERLAY' FROM 11 )
         AS RESULT1,
         OVERLAY( 'RESULT_OF_XXX_FUNC' PLACING 'OVERLAY' FROM 11 FOR 3 )
         AS RESULT2
      FROM DUAL;
RESULT1            RESULT2
------------------ ----------------------
RESULT_OF_OVERLAYC RESULT_OF_OVERLAY_FUNC
1 row selected.
```

17.117 PERCENT_RANK() OVER

Syntax

```
PERCENT_RANK( ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function PERCENT_RANK calculates the ranking ratio of each row for the entire number of rows.

The result of PERCENT_RANK function is the number between 0 and 1, and the rows with the same valu e have the same ratio value.

window frame is not available.

Example

```
gSQL> SELECT department_id, salary,
             PERCENT_RANK() OVER ( ORDER BY salary ) AS p_rank
      FROM employees
      WHERE department_id = 60;
DEPARTMENT_ID SALARY P_RANK
------------- ------ ------
           60   4200      0
           60   4800    .25
           60   4800    .25
           60   6000    .75
           60   9000      1
5 rows selected.
```

17.118 PERCENTILE_CONT() OVER

Syntax

```
PERCENTILE_CONT( expr ) WITHIN GROUP ( ORDER BY <sort specification> ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function PERCENTILE_CONT is an inverse distribution function assuming a consecutive distributi on model.

It calculates the value corresponding to the specified percentile score for *not null values* sorted within the group.
The calculation result may differ from the specific value sorted within the group.

expr should be the percentile score between 0 and 1.

Only PARTITION BY clause is available in OVER() clause.
It divides the query result sets into groups by using PARTITION BY clause.

It sorts the records within the group with WITHIN GROUP (ORDER BY <sort specification>).
It can specify only one <sort specification> in ORDER BY.

NULL is excluded from the sorted values.

The following is a calculation formula.

```
P : Percentile score
N : The number of the records of not null values sorted within the group
RN = ( 1 + ( P * (N-1) ) )
CRN = CEILING( RN )
FRM = FLOOR( RN )
* if ( CRN = FRN = RN )
     sort expression value of RN
* else
     sort expression value of ( CRN - RN ) * FRN + sort expression value of ( RN - FRN ) * CRN
```

MEDIAN window function is a specific case among PERCENTILE_CONT window functions, and its default value is percentile score 0.5.

For more information, refer to the followings.

- MEDIAN() OVER
- PERCENTILE_DISC() OVER

Example

```
gSQL>
SELECT item_no,
       sales,
       PERCENTILE_CONT( 0.5 ) WITHIN GROUP ( ORDER BY sales )
                              OVER ( PARTITION BY item_no )
       AS PERCENTILE_CONT
  FROM store;
ITEM_NO SALES PERCENTILE_CONT
------- ----- ---------------
    100    50             125
    100    90             125
    100    90             125
    100   100             125
    100   120             125
    100   130             125
    100   150             125
    100   150             125
    100   170             125
    100   200             125
    235    50              90
    235    70              90
    235    90              90
    235   130              90
    235   190              90
15 rows selected.
```

17.119 PERCENTILE_DISC() OVER

Syntax

```
PERCENTILE_DISC( expr ) WITHIN GROUP ( ORDER BY <sort specification> ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function PERCENTILE_DISC is an inverse distribution function assuming a discrete distribution m odel.

It calculates the percentile score for *not null values* sorted within the group.
It determines the smallest value among the values same or over the arguments percentile score of calcula ted percentile scores.
It returns the determined percentile score.

expr should be the percentile score between 0 and 1.

Only PARTITION BY clause is available in OVER() clause.
It divides the query result sets into groups by using PARTITION BY clause.

It sorts the records within the group with WITHIN GROUP (ORDER BY <sort specification>).
It can specify only one <sort specification> in ORDER BY.

It calculates CUME_DIST for the sort expression value with the sorted record.
When calculating CUME_DIST, NULL is excluded.

It determines the smallest value among CUME_DIST which is same or over the percentile score specified a s expr.
It returns the determined CUME_DIST value.

The result type is as same as sort expression value type.

For more information, refer to **CUME_DIST() OVER**.

Example

```
gSQL>
SELECT item_no,
       sales,
       CUME_DIST() OVER ( PARTITION BY item_no ORDER BY sales )
       AS CUME_DIST,
       PERCENTILE_DISC( 0.5 ) WITHIN GROUP ( ORDER BY sales )
                              OVER ( PARTITION BY item_no )
       AS PERCENTILE_DISC
  FROM store;
ITEM_NO SALES CUME_DIST PERCENTILE_DISC
------- ----- --------- ---------------
    100    50        .1             120
    100    90        .3             120
    100    90        .3             120
    100   100        .4             120
    100   120        .5             120
    100   130        .6             120
    100   150        .8             120
    100   150        .8             120
    100   170        .9             120
    100   200         1             120
    235    50        .2              90
    235    70        .4              90
    235    90        .6              90
    235   130        .8              90
    235   190         1              90
15 rows selected.
```

17.120 PHYSICAL_LENGTH

Syntax

```
PHYSICAL_LENGTH( expr )
```

Description

PHYSICAL_LENGTH returns the number of internal expression information bytes in expr.

The expr argument can be any data type.

If an input argument is NULL, then the result is 0.

Example

- When an input argument is NULL

```
gSQL> SELECT PHYSICAL_LENGTH( NULL ) AS RESULT FROM DUAL;
RESULT
------
     0
1 row selected.
```

- The following example shows the number of NUMBER type bytes of 1, 123 and 12345.

```
gSQL> SELECT PHYSICAL_LENGTH( 1 ) AS RESULT FROM DUAL;
RESULT
------
     2
1 row selected.
gSQL> SELECT PHYSICAL_LENGTH( 123 ) AS RESULT FROM DUAL;
RESULT
------
     3
1 row selected.
gSQL> SELECT PHYSICAL_LENGTH( 12345 ) AS RESULT FROM DUAL;
```

```
RESULT
------
     4
1 row selected.
```

17.121 PI

Syntax

```
PI()
```

Description

It returns "π" constant.

Example

```
gSQL> SELECT PI() AS RESULT FROM DUAL;
              RESULT
--------------------
3.141592653589793E+0
1 row selected.
```

17.122 POSITION

Syntax

```
POSITION( str1 IN str2 )
```

Description

It searches for the first str1 within str2, then returns its location.

The data type of str1 argument and str2 argument can be a character type such as CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING, or a binary character type BINARY, BINARY VARYING, BINARY LONG VARYING.

If str1 can not be found within str2, the return value is 0.
If str1 is found within str2, the position of str1 is returned, and the return value starts from 1.
The returned position value is calculated in character unit (not in byte unit).
If str1 or str2 is NULL, the return value is also NULL.

Example

```
gSQL> SELECT POSITION( 'CHAR' IN 'LONG CHAR 2000' ) AS RESULT FROM DUAL;
RESULT
------
     6
1 row selected.
```

17.123 POWER

Syntax

```
POWER( num1, num2 )
```

Description

It squares num1 to num2, and returns the result.

The num1 argument and num2 argument can be a numeric data type.
If num1 is a negative number, num2 should be an integer.
If num1 or num2 is NULL, the result is also NULL.

Example

```
gSQL> SELECT POWER( 2, 3 ) AS RESULT FROM DUAL;
RESULT
------
     8
1 row selected.
```

17.124 RADIANS

Syntax

```
RADIANS( degrees )
```

Description

It returns the radians of degrees.
The degrees argument can be a numeric data type.
If the degrees argument is NULL, then NULL is returned.

Example

```
gSQL> SELECT RADIANS( 180 ) AS RESULT FROM DUAL;
          RESULT
----------------
3.14159265358979
1 row selected.
```

17.125 RANDOM

Syntax

```
RANDOM( min, max )
```

Description

It returns a random value in the range above min and below max.
The min argument and max argument can be a numeric data type.
If either min argument or the max argument is NULL, then NULL is returned.

Example

```
gSQL> SELECT RANDOM( 1, 100 ) AS RESULT FROM DUAL;
          RESULT
----------------
34.1870528003201
1 row selected.
```

17.126 RANK() OVER

Syntax

```
RANK( ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function RANK calculates the ranking.

The ranking is an integer starting from 1, and rows with the same value have the same rank.

window frame is not available.

Example

```
gSQL> SELECT department_id, salary,
             RANK() OVER ( ORDER BY salary ) AS rank
      FROM employees
      WHERE department_id = 60;
DEPARTMENT_ID SALARY RANK
------------- ------ ----
           60   4200    1
           60   4800    2
           60   4800    2
           60   6000    4
           60   9000    5
5 rows selected.
```

17.127 RATIO_TO_REPORT() OVER

Syntax

```
RATIO_TO_REPORT ( expr ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function RATIO_TO_REPORT calculates the ratio of each row value for the total sum of expr valu es.
If the row value is NULL, then it returns NULL as a result.

order by is not available in window clause.
window frame is not available.

Example

```
gSQL> SELECT supplier_id, min_price
             , RATIO_TO_REPORT( min_price ) OVER ( PARTITION BY supplier_id ) AS "RATIO"
        FROM product_information
       WHERE supplier_id = 102050;
SUPPLIER_ID MIN_PRICE             RATIO
----------- --------- -----------------
     102050       731  .695528068506185
     102050      null              null
     102050        73 .0694576593720266
     102050       247  .235014272121789
     102050      null              null
5 rows selected.
```

17.128 REGR_AVGX() OVER

Syntax

```
REGR_AVGX( expr1, expr2 ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function REGR_AVGX is a linear regression function.
It calculates the average value of the independent variable expr2 of (X, Y) set's least-squares-fit linear equation.

The arguments of expr1 and expr2 are numeral types.

expr1 is an dependent variable (y) and expr2 is a independent variable (x).

If expr1 is NULL or expr2 is NULL, then it is excluded from the target.

The following is a calculation formula.

```
AVG( expr2 )
```

The returned type is a numeric type.

The returned value is the result from the calculation or NULL.
If all records are excluded from the target because expr1 or expr2 is NULL, then it returns NULL.

Example

```
gSQL>
SELECT item_no,
       price,
       year,
       REGR_AVGX( price, year ) OVER ( PARTITION BY item_no )
       AS "REGR_AVGX(price,year)"
```

```
  FROM store;
ITEM_NO PRICE YEAR REGR_AVGX(price,year)
------- ----- ---- --------------------
   3758 15000 2000                 2003
   3758 14700 2001                 2003
   3758 15300 2002                 2003
   3758 15200 2003                 2003
   3758 15100 2004                 2003
   3758 15300 2005                 2003
   3758 15350 2006                 2003
7 rows selected.
```

17.129 REGR_AVGY() OVER

Syntax

```
REGR_AVGY( expr1, expr2 ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function REGR_AVGY is a linear regression function.
It calculates the average value of the dependent variable expr1 of (X, Y) set's least-squares-fit linear equation.

The arguments of expr1 and expr2 are numeral types.

expr1 is an dependent variable (y) and expr2 is a independent variable (x).

If expr1 is NULL or expr2 is NULL, then it is excluded from the target.

The following is a calculation formula.

```
AVG( expr1 )
```

The returned type is a numeric type.

The returned value is the result from the calculation or NULL.
If all records are excluded from the target because expr1 or expr2 is NULL, then it returns NULL.

Example

```
gSQL>
SELECT item_no,
       price,
       year,
       round( REGR_AVGY( price, year ) OVER ( PARTITION BY item_no ), 5 )
       AS "REGR_AVGY(price,year)"
```

```
  FROM store;
ITEM_NO PRICE YEAR REGR_AVGY(price,year)
------- ----- ---- --------------------
   3758 15000 2000           15135.71429
   3758 14700 2001           15135.71429
   3758 15300 2002           15135.71429
   3758 15200 2003           15135.71429
   3758 15100 2004           15135.71429
   3758 15300 2005           15135.71429
   3758 15350 2006           15135.71429
7 rows selected.
```

17.130 REGR_COUNT() OVER

Syntax

```
REGR_COUNT( expr1, expr2 ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function REGR_COUNT is a linear regression function.
It calculates the least-squares-fit linear equation of (X, Y) set.

It returns the number of (X, Y) pairs, the execution targets of the linear equation and both of which are not NULL.

The arguments of expr1 and expr2 are numeral types.

expr1 is an dependent variable (y) and expr2 is a independent variable (x).

If expr1 is NULL or expr2 is NULL, then it is excluded from the target.

The returned type is a numeric type.

It returns the number of pairs both of whose expr1 and expr2 are not NULL.
If all records are excluded from the target because expr1 or expr2 is NULL, then it returns 0.

Example

```
gSQL>
SELECT item_no,
       price,
       year,
       REGR_COUNT( price, year ) OVER ( PARTITION BY item_no )
       AS "REGR_COUNT(price,year)"
  FROM store;
ITEM_NO PRICE YEAR REGR_COUNT(price,year)
------- ----- ---- ----------------------
```

```
   3758 15000 2000                    7
   3758 14700 2001                    7
   3758 15300 2002                    7
   3758 15200 2003                    7
   3758 15100 2004                    7
   3758 15300 2005                    7
   3758 15350 2006                    7
7 rows selected.
```

17.131 REGR_INTERCEPT() OVER

Syntax

```
REGR_INTERCEPT( expr1, expr2 ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function REGR_INTERCEPT is a linear regression function.
It calculates the y intercept of (X, Y) set's least-squares-fit linear equation.

The arguments of expr1 and expr2 are numeral types.

expr1 is an dependent variable (y) and expr2 is a independent variable (x).

If expr1 is NULL or expr2 is NULL, then it is excluded from the target.

The following is a calculation formula.

```
AVG( expr1 ) - REGR_SLOPE( expr1, expr2 ) * AVG( expr2 )
```

The returned type is a numeric type.

The returned value is the result from the calculation or NULL.

It returns NULL in the following cases.
- When all records are excluded from the target because expr1 or expr2 is NULL
- When the result of REGR_SLOPE is null

Example

```
gSQL>
SELECT item_no,
       price,
       year,
       REGR_INTERCEPT( price, year ) OVER ( PARTITION BY item_no )
```

```
       AS "REGR_INTERCEPT(price,year)"
  FROM store;
ITEM_NO PRICE YEAR REGR_INTERCEPT(price,year)
------- ----- ---- --------------------------
   3758 15000 2000                  -131512.5
   3758 14700 2001                  -131512.5
   3758 15300 2002                  -131512.5
   3758 15200 2003                  -131512.5
   3758 15100 2004                  -131512.5
   3758 15300 2005                  -131512.5
   3758 15350 2006                  -131512.5
7 rows selected.
```

17.132 REGR_R2() OVER

Syntax

```
REGR_R2( expr1, expr2 ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function REGR_R2 is a linear regression function.
It calculates the coefficient of determination (R-squared or fitness) of (X, Y) set's least-squares-fit linear e quation.

The arguments of expr1 and expr2 are numeral types.

expr1 is an dependent variable (y) and expr2 is a independent variable (x).

If expr1 is NULL or expr2 is NULL, then it is excluded from the target.

The following is a calculation formula.

```
If VAR_POP( expr2 ) = 0, then it is NULL.
If VAR_POP( expr1 ) = 0 and VAR_POP( expr2 ) != 0, then it is 1.
If VAR_POP( expr1 ) > 0 and VAR_POP( expr2 ) != 0, then it is POWER( CORR( expr1, expr2), 2 ).
```

The returned type is a numeric type.

The returned value is the result from the calculation or NULL.

It returns NULL in the following cases.
- When all records are excluded from the target because expr1 or expr2 is NULL
- When the result of VAR_POP (expr2) is 0

Example

```
gSQL>
SELECT item_no,
       price,
       year,
       round( REGR_R2( price, year ) OVER ( PARTITION BY item_no ), 10 )
       AS "REGR_R2(price,year)"
  FROM store
 WHERE item_no = 3758;
ITEM_NO PRICE YEAR REGR_R2(price,year)
------- ----- ---- --------------------
   3758 15000 2000          .4786446469
   3758 14700 2001          .4786446469
   3758 15300 2002          .4786446469
   3758 15200 2003          .4786446469
   3758 15100 2004          .4786446469
   3758 15300 2005          .4786446469
   3758 15350 2006          .4786446469
7 rows selected.
```

17.133 REGR_SLOPE() OVER

Syntax

```
REGR_SLOPE( expr1, expr2 ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function REGR_SLOPE is a linear regression function.
It calculates the slope of (X, Y) set's least-squares-fit linear equation.

The arguments of expr1 and expr2 are numeral types.

expr1 is an dependent variable (y) and expr2 is a independent variable (x).

If expr1 is NULL or expr2 is NULL, then it is excluded from the target.

The following is a calculation formula.

```
COVAR_POP(expr1, expr2) / VAR_POP(expr2)
```

The returned type is a numeric type.

The returned value is the result from the calculation or NULL.

It returns NULL in the following cases.
- When all records are excluded from the target because expr1 or expr2 is NULL
- When the result of VAR_POP is 0

Example

```
gSQL>
SELECT item_no,
       price,
       year,
       round( REGR_SLOPE( price, year ) OVER ( PARTITION BY item_no ), 10 )
```

```
       AS "REGR_SLOPE(price,year)"
  FROM store;
ITEM_NO PRICE YEAR REGR_SLOPE(price,year)
------- ----- ---- ----------------------
   3758 15000 2000          73.2142857143
   3758 14700 2001          73.2142857143
   3758 15300 2002          73.2142857143
   3758 15200 2003          73.2142857143
   3758 15100 2004          73.2142857143
   3758 15300 2005          73.2142857143
   3758 15350 2006          73.2142857143
7 rows selected.
```

17.134 REGR_SXX() OVER

Syntax

```
REGR_SXX( expr1, expr2 ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function REGR_SXX is a linear regression function.
It is an auxiliary function calculating the diagnostic statistics of (X, Y) set's least-squares-fit linear equation.

The arguments of expr1 and expr2 are numeral types.

expr1 is an dependent variable (y) and expr2 is a independent variable (x).

If expr1 is NULL or expr2 is NULL, then it is excluded from the target.

The following is a calculation formula.

```
REGR_COUNT( expr1, expr2 ) * VAR_POP( expr2 )
```

The returned type is a numeric type.

The returned value is the result from the calculation or NULL.
If all records are excluded from the target because expr1 or expr2 is NULL, then it returns NULL.

Example

```
gSQL>
SELECT item_no,
       price,
       year,
       REGR_SXX( price, year ) OVER ( PARTITION BY item_no )
       AS "REGR_SXX(price,year)"
  FROM store;
```

```
ITEM_NO PRICE YEAR REGR_SXX(price,year)
------- ----- ---- --------------------
   3758 15000 2000                   28
   3758 14700 2001                   28
   3758 15300 2002                   28
   3758 15200 2003                   28
   3758 15100 2004                   28
   3758 15300 2005                   28
   3758 15350 2006                   28
7 rows selected.
```

17.135 REGR_SXY() OVER

Syntax

```
REGR_SXY( expr1, expr2 ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function REGR_SXY is a linear regression function.
It is an auxiliary function calculating the diagnostic statistics of (X, Y) set's least-squares-fit linear equation.

The arguments of expr1 and expr2 are numeral types.

expr1 is an dependent variable (y) and expr2 is a independent variable (x).

If expr1 is NULL or expr2 is NULL, then it is excluded from the target.

The following is a calculation formula.

```
REGR_COUNT( expr1, expr2 ) * COVAR_POP( expr1, expr2 )
```

The returned type is a numeric type.

The returned value is the result from the calculation or NULL.
If all records are excluded from the target because expr1 or expr2 is NULL, then it returns NULL.

Example

```
gSQL>
SELECT item_no,
       price,
       year,
       REGR_SXY( price, year ) OVER ( PARTITION BY item_no )
       AS "REGR_SXY(price,year)"
  FROM store;
```

```
ITEM_NO PRICE YEAR REGR_SXY(price,year)
------- ----- ---- --------------------
   3758 15000 2000                 2050
   3758 14700 2001                 2050
   3758 15300 2002                 2050
   3758 15200 2003                 2050
   3758 15100 2004                 2050
   3758 15300 2005                 2050
   3758 15350 2006                 2050
7 rows selected.
```

17.136 REGR_SYY() OVER

Syntax

```
REGR_SYY( expr1, expr2 ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function REGR_SYY is a linear regression function.
It is an auxiliary function calculating the diagnostic statistics of (X, Y) set's least-squares-fit linear equation.

The arguments of expr1 and expr2 are numeral types.

expr1 is an dependent variable (y) and expr2 is a independent variable (x).

If expr1 is NULL or expr2 is NULL, then it is excluded from the target.

The following is a calculation formula.

```
REGR_COUNT( expr1, expr2 ) * VAR_POP( expr1 )
```

The returned type is a numeric type.

The returned value is the result from the calculation or NULL.
If all records are excluded from the target because expr1 or expr2 is NULL, then it returns NULL.

Example

```
gSQL>
SELECT item_no,
       price,
       year,
       round( REGR_SYY( price, year ) OVER ( PARTITION BY item_no ), 4 )
       AS "REGR_SYY(price,year)"
  FROM store;
```

```
ITEM_NO PRICE YEAR REGR_SYY(price,year)
------- ----- ---- --------------------
   3758 15000 2000          313571.4286
   3758 14700 2001          313571.4286
   3758 15300 2002          313571.4286
   3758 15200 2003          313571.4286
   3758 15100 2004          313571.4286
   3758 15300 2005          313571.4286
   3758 15350 2006          313571.4286
7 rows selected.
```

17.137 REPEAT

Syntax

```
REPEAT( str, num )
```

Description

The string repeats str as many times as specified in num, and returns the result.

The str argument can be a character type such as CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING, or a binary character type such as BINARY, BINARY VARYING, BINARY LONG VARYING.

The num argument can be a numeric data type.

If either str or num is NULL, the result is also NULL.
If num is 0 or a negative number, the result is also NULL.

The following table describes the result types.

Table 17-16 Result type of REPEAT

str type	Result type
CHAR or VARCHAR	VARCHAR
LONG VARCHAR	LONG VARCHAR
BINARY or VARBINARY	VARBINARY
LONG VARBINARY	LONG VARBINARY

Example

```
gSQL> SELECT REPEAT( 'ab', 3 ) AS RESULT FROM DUAL;
RESULT
------
ababab
1 row selected.
```

17.138 REPLACE

Syntax

```
REPLACE( str, from, to )
```

Description

It replaces all *from* strings in str string with *to* strings, and returns the result.

The str argument, the from argument, and the to argument can be a character data types such as CHAR ACTER, CHARACTER VARYING, CHARACTER LONG VARYING.

If str is NULL, the result is also NULL.
If from is NULL, the str is returned without replacement.
If to value is omitted or NULL, the str value of which from is removed is returned.

The following table describes the result types.

Table 17-17 Result type of REPLACE

str type	Result type
CHAR or VARCHAR	VARCHAR
LONG VARCHAR	LONG VARCHAR

Example

```
gSQL> SELECT REPLACE( 'HI GLIESE', 'HI', 'HELLO' ) AS RESULT FROM DUAL;
RESULT
------------
HELLO GLIESE
1 row selected.
```

17.139 REVERSE

Syntax

```
REVERSE( str )
```

Description

REVERSE returns characters of str in reverse order.

The str argument can be types which is convertible to a character string type or a binary string type.
A character string type is performed in a character unit, and a binary stringtype can be performed in a byte unit.

If str is NULL, then it returns NULL.

The following table describes the arguments and result types.

Table 17-18 Argument and result type of REVERSE

str	Result type
CHAR	CHAR
VARCHAR	VARCHAR
LONG VARCHAR	LONG VARCHAR
BINARY	BINARY
VARBINARY	VARBINARY
LONG VARBINARY	LONG VARBINARY

Example

```
gSQL> SELECT REVERSE( 'GOLDILOCKS' ) AS RESULT FROM DUAL;
RESULT
----------
SKCOLIDLOG
1 row selected.
gSQL> SELECT REVERSE( '선재소프트 2018' ) AS RESULT FROM DUAL;
```

```
RESULT
---------------
8102 트프소재선
1 row selected.
```

17.140 ROUND(number)

Syntax

```
ROUND( num [, scale ] )
```

Description

It rounds off num based on scale, and returns the result.

The num argument and scale argument can be a numeric data type.

If scale is omitted, the scale becomes 0 and is executed as if it is ROUND(num, 0).
If scale is a positive number, it is rounded off based on the number of right digit of the decimal point. If s cale is a negative number, it is rounded off based on the number of left digit of the decimal point.
If either num argument or the scale argument is NULL, then NULL is returned.

Example

```
gSQL> SELECT ROUND( 152.4282, 2 ) AS RESULT FROM DUAL;
RESULT
------
152.43
1 row selected.
gSQL> SELECT ROUND( 152.4282, -2 ) AS RESULT FROM DUAL;
RESULT
------
   200
1 row selected.
```

17.141 ROUND(date)

Syntax

```
ROUND( date [ , fmt ] )
```

Description

It rounds off the date in the specified fmt unit, and returns the result.

The data type of date argument can be DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE.
The fmt argument can be a character type such as CHARACTER, CHARACTER VARYING.
If either date argument or the fmt argument is NULL, then NULL is returned.
The result type is always DATE regardless of the date argument data type.

If fmt is omitted, the default is DAY.
The following table describes the available format strings.

Table 17-19 Available format sting of fmt

String	Description
CC, SCC	It is represented in four digit year by rounding off from 51 year. (e.g. XX01)
YYYY, YEAR, SYYYY, SYEAR, YYY, YY, Y	It is rounded off from July 1st.
IYYY, IYY, IY, I	It is the year embracing the calendar week defined by ISO 8601 standards, and it is rounded off from July 1st.
Q	It is rounded off from the 16th day in the second month of the quarter.
MONTH, MON, MM, RM	It is rounded off from the 16th day.
WW	A week starts from January 1st of the year, and it is rounded off on wednesday 12 p.m of WEEK.
IW	It is the calendar week defined by ISO 8601 standards (1 ~ 52 week or 1 ~ 53 weeks), and it is rounded off on thursday 12 p.m.
W	A week starts from the 1st day of the month, and it is rounded off on wednesday 12 p.m of WEEK.
DDD, DD, J	It is rounded off at 12 p.m.
DAY, DY, D	It is rounded off on wednesday 12 p.m of WEEK.
HH, HH12, HH24	It is rounded off from 30 minutes.
MI	It is rounded off from 30 seconds.

Example

```
gSQL> SELECT
      ROUND( TO_DATE( '2051-07-16', 'YYYY-MM-DD' ), 'CC' ) AS RESULT
      FROM DUAL;
RESULT
----------
2101-01-01
1 row selected.
gSQL> SELECT
      ROUND( TO_DATE( '2051-07-16', 'YYYY-MM-DD' ), 'YYYY' ) AS RESULT
      FROM DUAL;
RESULT
----------
2052-01-01
1 row selected.
gSQL> SELECT
      ROUND( TO_DATE( '2051-07-16', 'YYYY-MM-DD' ), 'MONTH' ) AS   RESULT
      FROM DUAL;
RESULT
----------
2051-08-01
1 row selected.
gSQL> SELECT
      ROUND( TO_TIMESTAMP( '2001-05-05 15:22:33.999999',
                           'YYYY-MM-DD HH24:MI:SS.FF6' ) ) AS RESULT
      FROM DUAL;
RESULT
----------
2001-05-06
1 row selected.
```

17.142 ROW_NUMBER() OVER

Syntax

```
ROW_NUMBER( ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function ROW_NUMBER assigns the unique number to each row.
The unique number is a consecutive integer starting from 1.

window frame is not available.

Example

```
gSQL> SELECT department_id, salary,
             ROW_NUMBER() OVER ( ORDER BY salary ) AS row_num
      FROM employees
      WHERE department_id = 60;
DEPARTMENT_ID SALARY ROW_NUM
------------- ------ -------
           60   4200       1
           60   4800       2
           60   4800       3
           60   6000       4
           60   9000       5
5 rows selected.
```

17.143 ROWID_GRID_BLOCK_ID

Syntax

```
ROWID_GRID_BLOCK_ID( rowid )
```

Description

It returns the GRID block ID.


It is a valid information in a cluster system.

Example

```
gSQL> SELECT C1, ROWID_GRID_BLOCK_ID( ROWID ) FROM T1;
C1 ROWID_GRID_BLOCK_ID( ROWID )
-- ----------------------------
 1                           52
 2                           52
 3                           52
3 rows selected.
```

17.144 ROWID_GRID_BLOCK_SEQ

Syntax

```
ROWID_GRID_BLOCK_SEQ( rowid )
```

Description

It returns the GRID block sequence.

It is a valid information in a cluster system.

Example

```
gSQL> SELECT C1, ROWID_GRID_BLOCK_SEQ( ROWID ) FROM T1;
C1 ROWID_GRID_BLOCK_SEQ( ROWID )
-- ------------------------------
 1                         747465
 2                         747466
 3                         747467
3 rows selected.
```

17.145 ROWID_MEMBER_ID

Syntax

```
ROWID_MEMBER_ID( rowid )
```

Description

It returns the member ID.

It is a valid information in a cluster system.

Example

```
gSQL> SELECT C1, ROWID_MEMBER_ID( ROWID ) FROM T1;
C1 ROWID_MEMBER_ID( ROWID )
-- ------------------------
 1                        1
 2                        1
 3                        1
3 rows selected.
```

17.146 ROWID_OBJECT_ID

Syntax

```
ROWID_OBJECT_ID( rowid )
```

Description

It returns the object ID.

It is an invalid information in a cluster system.

Example

```
gSQL> SELECT ROWID_OBJECT_ID( t1.ROWID ) FROM t1;
ROWID_OBJECT_ID( T1.ROWID )
---------------------------
                      22012
                      22012
                      22012
                      22012
4 rows selected.
```

17.147 ROWID_PAGE_ID

Syntax

```
ROWID_PAGE_ID( rowid )
```

Description

It returns the page ID.

It is an invalid information in a cluster system.

Example

```
gSQL> SELECT ROWID_PAGE_ID( t1.ROWID ) FROM t1;
ROWID_PAGE_ID( T1.ROWID )
-------------------------
                     8227
                     8227
                     8227
                     8227
4 rows selected.
```

17.148 ROWID_ROW_NUMBER

Syntax

```
ROWID_ROW_NUMBER( rowid )
```

Description

It returns the row number.

It is an invalid information in a cluster system.

Example

```
gSQL> SELECT ROWID_ROW_NUMBER( t1.ROWID ) FROM t1;
ROWID_ROW_NUMBER( T1.ROWID )
----------------------------
                           0
                           1
                           2
                           3
4 rows selected.
```

17.149 ROWID_SHARD_ID

Syntax

```
ROWID_SHARD_ID( rowid )
```

Description

It returns the shard ID.

It is a valid information in a cluster system.

Example

```
gSQL> SELECT C1, ROWID_SHARD_ID( ROWID ) FROM T1;
C1 ROWID_SHARD_ID( ROWID )
-- ------------------------
 1                       0
 2                       1
 3                       2
3 rows selected.
```

17.150 ROWID_TABLESPACE_ID

Syntax

```
ROWID_TABLESPACE_ID( rowid )
```

Description

It returns the tablespace ID.


It is an invalid information in a cluster system.

Example

```
gSQL> SELECT ROWID_TABLESPACE_ID( t1.ROWID ) FROM t1;
ROWID_TABLESPACE_ID( T1.ROWID )
------------------------------
                             2
                             2
                             2
                             2
4 rows selected.
```

17.151 ROWNUM

Syntax

```
ROWNUM
```

Description

It sequentially allocates a number starting from 1 to rows which satisfy the WHERE condition.

It allows using ROWNUM in WHERE clause for the compatibility with Oracle.

However, to restrict the number of the query results, it is recommended to use **offset limit clause** (the SQL standard) as follows.

- (Non standard) Describing the number of the results by using ROWNUM

```
gSQL> SELECT * FROM t1 WHERE ROWNUM <= 3;
C1
--
A
B
C
3 rows selected.
```

- (SQL standard) Describing the number of the results by using FETCH statement

```
gSQL> SELECT * FROM t1 FETCH 3;
C1
--
A
B
C
3 rows selected.
```

To restrict the range of the query results, it is recommended to use OFFSET, FETCH statement as follows.

- (Non standard) Describing the range of the number of the results by using ROWNUM

```
gSQL> SELECT c1
        FROM ( SELECT ROWNUM rn, c1
                 FROM t1 )
        WHERE rn BETWEEN 2 AND 3;
C1
--
B
C
2 rows selected.
```

- (SQL standard) Describing the range of the number of the results by using ROWNUM OFFSET, FETCH statement

```
gSQL> SELECT c1 FROM t1 OFFSET 1 FETCH 2;
C1
--
B
C
2 rows selected.
```

It is not recommended to use ROWNUM in WHERE clause for any other uses than the restriction of the n umber of the results.

The results for the same query may be different according to the execution method as follows when usin g the ambiguous condition (WHERE c1 < ROWNUM + 3).

- Creating the data

```
CREATE TABLE t1 ( c1 INTEGER );
CREATE INDEX t1_idx ON t1(c1);
INSERT INTO t1 VALUES (1);
INSERT INTO t1 VALUES (2);
INSERT INTO t1 VALUES (3);
INSERT INTO t1 VALUES (4);
INSERT INTO t1 VALUES (5);
COMMIT;
```

- In case of Oracle

```
SQL> SELECT ROWNUM, c1 FROM t1 WHERE c1 < ROWNUM + 3;
    ROWNUM       C1
---------- ----------
     1        1
```

```
     2        2
SQL> DROP INDEX t1_idx;
```

- The index has been deleted.

```
SQL> SELECT ROWNUM, c1 FROM t1 WHERE c1 < ROWNUM + 3;
    ROWNUM       C1
---------- ----------
     1        1
     2        2
     3        3
     4        4
     5        5
```

- In case of GOLDILOCKS

```
gSQL> SELECT ROWNUM, c1 FROM t1 WHERE c1 < ROWNUM + 3;
ROWNUM C1
------ --
     1  1
     2  2
     3  3
     4  4
     5  5
5 rows selected.
gSQL> DROP INDEX t1_idx;
Index dropped.
gSQL> SELECT ROWNUM, c1 FROM t1 WHERE c1 < ROWNUM + 3;
ROWNUM C1
------ --
     1  1
     2  2
     3  3
     4  4
     5  5
5 rows selected.
```

Example

```
gSQL> SELECT ROWNUM, c1 FROM t1;
ROWNUM C1
------ --
     1 A
     2 B
     3 C
     4 D
     5 E
5 rows selected.
```

17.152 RPAD

Syntax

```
RPAD( str, length, [, fill] )
```

Description

It adds fill string to the right side of str until the string's length becomes length, and it returns the result.

The str argument can be a character type such as CHARACTER, CHARACTER VARYING, CHARACTER LON GVARYING, or a binary character type such as BINARY, BINARY VARYING, BINARY LONG VARYING.

The length argument can be a numeric type.

length means the number of characters, and its maximum range is the maximum PRECISION of the result type.
If fill is omitted, a white space is added.
If str is longer than length, it cuts the str as long as the length, then returns it.
If any of str, length, fill is NULL, the result is also NULL.
If length is 0 or a negative number, the result is NULL.

The following table describes the result types.

Table 17-20 Result type of RPAD

str type	Result type
CHAR or VARCHAR	VARCHAR
LONG VARCHAR	LONG VARCHAR
BINARY or VARBINARY	VARBINARY
LONG VARBINARY	LONG VARBINARY

Example

```
gSQL> SELECT RPAD('AA', 5) AS RESULT1,
             RPAD('AA', 5, 'X') AS RESULT2,
             RPAD('AA', 1) AS RESULT3
```

```
      FROM DUAL;
RESULT1 RESULT2 RESULT3
------- ------- -------
AA      AAXXX   A
1 row selected.
```

17.153 RTRIM

Syntax

```
RTRIM( trim_source [, trim_character ] )
```

Description

It removes the matching characters by comparing from the right side of trim_character in trim_source unt il the matching character does not exist. Then it returns the result.

The data type of trim_character argument and trim_source argument can be a character type such as CH ARACTER, CHARACTER VARYING, CHARACTER LONG VARYING or a binary data type such as BINARY, BI NARY VARYING, BINARY LONG VARYING.

If any of trim_character, trim_source is NULL, the result is NULL.
If trim_character is omitted, a single blank space (' ') is specified by default.

The following table describes the result types.

Table 17-21 Result type of RTRIM

trim_source type, trim_character type	Result type
CHAR or VARCHAR	VARCHAR
LONG VARCHAR	LONG VARCHAR
BINARY or VARBINARY	VARBINARY
LONG VARBINARY	LONG VARBINARY

Example

```
gSQL> SELECT RTRIM('      rtrim      ') AS RESULT1,
             RTRIM('______rtrim______','_') AS RESULT2
      FROM DUAL;
RESULT1     RESULT2
----------- -----------
      rtrim ______rtrim
```

```
1 row selected.
```

17.154 SESSION_ID

Syntax

```
SESSION_ID()
```

Description

It obtains the current session ID.

Example

```
gSQL> SELECT SESSION_ID() FROM dual;
SESSION_ID()
------------
           4
1 row selected.
```

17.155 SESSION_SERIAL

Syntax

```
SESSION_SERIAL()
```

Description

It obtains the serial number of current session.

Example

```
gSQL> SELECT SESSION_SERIAL() FROM dual;
SESSION_SERIAL()
----------------
              16
1 row selected.
```

17.156 SESSION_USER

Syntax

```
SESSION_USER[()]
```

Description

It returns the session user.

The user information is managed in three types as follows.

- Logon user: It is a user who performed login, and it is maintained until the connection is closed.
- Session user: It is as same as the first logon user, but it can be changed using the SET SESSION AUTH ORIZATION statement.
- Current user: It is generally as same as the session user, but it is temporarily changed internally in syst em to control access when using the PSM, view.
    - The session user and current user is similar to the difference between the unix system's real user and the effective user.

Example

```
% gsql sys gliese
gSQL>  SET SESSION AUTHORIZATION test;
Session set.
gSQL> SELECT LOGON_USER() AS result1, SESSION_USER() AS result2 FROM DUAL;
RESULT1 RESULT2
------- -------
SYS     TEST
1 row selected.
```

17.157 SESSIONTIMEZONE

Syntax

```
SESSIONTIMEZONE()
```

Description

SESSIONTIMEZONE returns the time zone of the current session.
The returned value is '[+|-]TZH:TZM' format character.
The returned type is varchar.

Example

Check the time zone of the current session.

```
gSQL> SELECT SESSIONTIMEZONE() FROM dual;
SESSIONTIMEZONE()
-----------------
+09:00
1 row selected.
```

When modifying the time zone of the current session, then it returns the modified time zone value.

```
gSQL> SET TIME ZONE '-05:00';
Session set.
gSQL> SELECT SESSIONTIMEZONE() FROM dual;
SESSIONTIMEZONE()
-----------------
-05:00
1 row selected.
```

17.158 SHARD_GROUP_ID

Syntax

```
SHARD_GROUP_ID( table_name, shard_key_value [, ... ] )
```

Description

It returns the group ID managing the shard which stores shard_key_value when the shard strategy is defi ned in the table_name.

The table_name (an input argument) should be described by an identifier. If an object corresponding to t he table_name is not a base table, or if the shard strategy is not defined, then an error occurs.

The shard_key_value (an input argument) should be listed in an order of shard key column in the shard st rategy defined in the table_name. If the number of shard_key_value and the number of shard key column is not same, then an error occurs.

The result type is NATIVE_BIGINT.


It is a valid information in a cluster system.

Example

```
gSQL> SELECT T1.C1, SHARD_GROUP_ID( T1, T1.C1 ) FROM T1;
C1 SHARD_GROUP_ID( T1, T1.C1 )
-- --------------------------
A                           1
B                           2
C                           3
3 rows selected.
gSQL> SELECT SHARD_GROUP_ID( T1, 'B' ) FROM DUAL;
SHARD_GROUP_ID( T1, 'B' )
-------------------------
                        2
```

```
1 row selected.
```

17.159 SHARD_GROUP_NAME

Syntax

```
SHARD_GROUP_NAME( table_name, shard_key_value [, ... ] )
```

Description

It returns the group NAME managing the shard which stores shard_key_value when the shard strategy is defined in the table_name.

The table_name (an input argument) should be described by an identifier. If an object corresponding to t he table_name is not a base table, or if the shard strategy is not defined, then an error occurs.

The shard_key_value (an input argument) should be listed in an order of shard key column in the shard st rategy defined in the table_name. If the number of shard_key_value and the number of shard key column is not same, then an error occurs.

The result type is VARCHAR.


It is a valid information in a cluster system.

Example

```
gSQL> SELECT T1.C1, SHARD_GROUP_NAME( T1, T1.C1 ) FROM T1;
C1 SHARD_GROUP_NAME( T1, T1.C1 )
-- ------------------------------
A  G1
B  G2
C  G3
3 rows selected.
gSQL> SELECT SHARD_GROUP_NAME( T1, 'B' ) FROM DUAL;
SHARD_GROUP_NAME( T1, 'B' )
---------------------------
G2
```

```
1 row selected.
```

17.160 SHARD_ID

Syntax

```
SHARD_ID( table_name, shard_key_value [, ... ] )
```

Description

It returns the ID for the shard which stores shard_key_value when the shard strategy is defined in the table_name.

The table_name (an input argument) should be described by an identifier. If an object corresponding to the table_name is not a base table, or if the shard strategy is not defined, then an error occurs.

The shard_key_value (an input argument) should be listed in an order of shard key column in the shard strategy defined in the table_name. If the number of shard_key_value and the number of shard key column is not same, then an error occurs.

The result type is NATIVE_BIGINT.


It is a valid information in a cluster system.

Example

```
gSQL> SELECT T1.C1, SHARD_ID( T1, T1.C1 ) FROM T1;
C1 SHARD_ID( T1, T1.C1 )
-- --------------------
A                     0
B                     1
C                     2
3 rows selected.
gSQL> SELECT SHARD_ID( T1, 'B' ) FROM DUAL;
SHARD_ID( T1, 'B' )
-------------------
                  1
```

```
1 row selected.
```

17.161 SHARD_NAME

Syntax

```
SHARD_NAME( table_name, shard_key_value [, ... ] )
```

Description

It returns the NAME for the shard which stores shard_key_value when the shard strategy is defined in the table_name.

The table_name (an input argument) should be described by an identifier. If an object corresponding to t he table_name is not a base table, or if the shard strategy is not defined, then an error occurs.

The shard_key_value (an input argument) should be listed in an order of shard key column in the shard st rategy defined in the table_name. If the number of shard_key_value and the number of shard key column is not same, then an error occurs.

The result type is VARCHAR.


It is a valid information in a cluster system.

Example

```
gSQL> SELECT T1.C1, SHARD_NAME( T1, T1.C1 ) FROM T1;
C1 SHARD_NAME( T1, T1.C1 )
-- ----------------------
A  S1
B  S2
C  S3
3 rows selected.
gSQL> SELECT SHARD_NAME( T1, 'B' ) FROM DUAL;
SHARD_NAME( T1, 'B' )
---------------------
S2
```

```
1 row selected.
```

17.162 SHIFT_LEFT

Syntax

```
SHIFT_LEFT( num, cnt )
```

Description

It moves num to the left as many as cnt bits, and returns the movement values.

The data type of input num argument and cnt argument can be NATIVE_SMALLINT, NATIVE_INTEGER, NATIVE_BIGINT, or the type which can be converted to NATIVE_BIGINT.
When converting to NATIVE_BIGINT type, the decimal point is truncated.

cnt is masked with 6 bit, and it is processed to a value in the range within 6 bit.

If either num or cnt is NULL, then NULL is returned.

The result type is NATIVE_BIGINT.

Example

```
gSQL> SELECT SHIFT_LEFT(1, 3) FROM DUAL;
SHIFT_LEFT(1, 3)
----------------
               8
1 row selected.
```

17.163 SHIFT_RIGHT

Syntax

```
SHIFT_RIGHT( num, cnt )
```

Description

It moves num to the right as many as cnt bits, and returns the movement values.

The data type of input num argument and cnt argument can be NATIVE_SMALLINT, NATIVE_INTEGER, NATIVE_BIGINT, or the type which can be converted to NATIVE_BIGINT.
When converting to NATIVE_BIGINT type, the decimal point is truncated.

cnt is masked with 6 bit, and it is processed to a value in the range within 6 bit.

If either num or cnt is NULL, then NULL is returned.

The result type is NATIVE_BIGINT.

Example

```
gSQL> SELECT SHIFT_RIGHT(8, 3) FROM DUAL;
SHIFT_RIGHT(8, 3)
-----------------
                1
1 row selected.
```

17.164 SIGN

Syntax

```
SIGN( num )
```

Description

It returns the sign of num.

The num argument can be a numeric data type.

The return value is as follows.

- If num < 0, -1 is returned.
- If num = 0, 0 is returned.
- If num > 0, 1 is returned.

If num is NULL, then NULL is returned.

Example

```
gSQL> SELECT SIGN(-10) AS RESULT1,
             SIGN(0) AS RESULT2,
             SIGN(10) AS RESULT3 FROM DUAL;
RESULT1 RESULT2 RESULT3
------- ------- -------
     -1       0       1
1 row selected.
```

17.165 SIN

Syntax

```
SIN( num )
```

Description

It returns the sine value of num.
If num is NULL, then NULL is returned.

Example

```
gSQL> SELECT SIN( 0 ) AS RESULT FROM DUAL;
RESULT
------
     0
1 row selected.
```

17.166 SPLIT_PART

Syntax

```
SPLIT_PART( string, delimiter, field )
```

Description

It returns a character string of the field by specifying a character as delimiter within a string.

The data type of string argument and delimiter argument can be a character data type such as CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING.

The field argument can be a numeric data type.

If any of string, delimiter, field is NULL, the result is also NULL.
The value of field should be a numeric value above 1, and if it is 0 or a negative number, an error is returned.

The following table describes the result types.

Table 17-22 Result type of SPLIT_PART

string type	Result type
CHAR or VARCHAR	VARCHAR
LONG VARCHAR	LONG VARCHAR

Example

```
gSQL> SELECT SPLIT_PART( 'AB;CD;EF;GH', ';', 3  ) AS RESULT FROM DUAL;
RESULT
------
EF
1 row selected.
```

17.167 SQRT

Syntax

```
SQRT( num )
```

Description

It returns the square root of num.

The num argument can be a numeric type, and it should not be a negative number, but above 0.
If the num argument is NULL, then NULL is returned.

Example

```
gSQL> SELECT SQRT( 9 ) AS RESULT FROM DUAL;
RESULT
------
     3
1 row selected.
```

17.168 STATEMENT_DATE

Syntax

```
STATEMENT_DATE()
CURRENT_DATE [()]
```

Description

The current date(DATE type) value is obtained.

The differences among the functions to obtain the current date are as follows.

- TRANSACTION_DATE(): All date values in the transaction are same.
- STATEMENT_DATE(): All date values in an SQL statement are same.
- CLOCK_DATE(): Whenever the function is called, the current date value is obtained.

Example

```
gSQL> SELECT STATEMENT_DATE() AS result FROM t1;
RESULT
----------
2013-12-12
2013-12-12
2013-12-12
3 rows selected.
```

17.169 STATEMENT_LOCALTIME

Syntax

```
STATEMENT_LOCALTIME()
LOCALTIME [()]
```

Description

The current TIME WITHOUT TIME ZONE type value based on the session time is obtained.

LOCALTIME is an SQL standard function.

The differences among the functions to obtain the current time are as follows.

- TRANSACTION_LOCALTIME(): All time values in the transaction are same.
- TATEMENT_LOCALTIME(): All time values in an SQL statement are same.
- CLOCK_LOCALTIME(): Whenever the function is called, the current time value is obtained.

Example

All rows have the same time value.

```
gSQL> SELECT STATEMENT_LOCALTIME() AS result FROM t1;
RESULT
---------------
16:18:50.775870
16:18:50.775870
16:18:50.775870
3 rows selected.
```

17.170 STATEMENT_LOCALTIMESTAMP

Syntax

```
STATEMENT_LOCALTIMESTAMP()
LOCALTIMESTAMP [()]
```

Description

The current TIMESTAMP WITHOUT TIME ZONE type value based on the session time is obtained.

LOCALTIMESTAMP is an SQL standard function.

The differences among the functions to obtain the current timestamp are as follows.

- TRANSACTION_LOCALTIMESTAMP(): All timestamp values in the transaction are same.
- STATEMENT_LOCALTIMESTAMP(): All timestamp values in an SQL statement are same.
- CLOCK_LOCALTIMESTAMP(): Whenever the function is called, the current timestamp value is obtained.

Example

All rows have the same value.

```
gSQL> SELECT STATEMENT_LOCALTIMESTAMP() FROM t1;
STATEMENT_LOCALTIMESTAMP()
--------------------------
2013-12-12 16:23:39.782187
2013-12-12 16:23:39.782187
2013-12-12 16:23:39.782187
3 rows selected.
```

17.171 STATEMENT_TIME

Syntax

```
STATEMENT_TIME()
CURRENT_TIME [()]
```

Description

The current TIME WITH TIME ZONE type value is obtained.

CURRENT_TIME is an SQL standard function.

The differences among the functions to obtain the current time are as follows.

- TRANSACTION_TIME(): All time values in the transaction are same.
- STATEMENT_TIME(): All time values in an SQL statement are same.
- CLOCK_TIME(): Whenever the function is called, the current time value is obtained.

Example

All rows have the same time value.

```
gSQL> SELECT STATEMENT_TIME() AS result FROM t1;
RESULT
----------------------
16:28:19.268513 +09:00
16:28:19.268513 +09:00
16:28:19.268513 +09:00
3 rows selected.
```

17.172 STATEMENT_TIMESTAMP

Syntax

```
STATEMENT_TIMESTAMP()
CURRENT_TIMESTAMP [()]
```

Description

The current TIMESTAMP WITH TIME ZONE type value is obtained.

CURRENT_TIMESTAMP is an SQL standard function.

The differences among the functions to obtain the current timestamp are as follows.

- TRANSACTION_TIMESTAMP(): All timestamp values in the transaction are same.
- STATEMENT_TIMESTAMP(): All timestamp values in an SQL statement are same.
- CLOCK_TIMESTAMP(): Whenever the function is called, the current timestamp value is obtained.

Example

All rows have the same value.

```
gSQL> SELECT STATEMENT_TIMESTAMP() AS result FROM t1;
RESULT
-------------------------------
2013-12-12 16:36:11.032957 +09:00
2013-12-12 16:36:11.032957 +09:00
2013-12-12 16:36:11.032957 +09:00
3 rows selected.
```

17.173 STATEMENT_VIEW_SCN

Syntax

```
STATEMENT_VIEW_SCN()
```

Description

It obtains VIEW SCN of the current STATEMENT.

Example

```
gSQL> SELECT STATEMENT_VIEW_SCN() FROM dual;
STATEMENT_VIEW_SCN()
--------------------
17697.658.17880
1 row selected.
```

17.174 STATEMENT_VIEW_SCN_DCN

Syntax

```
STATEMENT_VIEW_SCN_DCN()
```

Description

It obtains the Domain Change Number (DCN) value of the current STATEMENT's VIEW SCN.

Example

```
gSQL> SELECT STATEMENT_VIEW_SCN_DCN() FROM dual;
STATEMENT_VIEW_SCN_DCN()
------------------------
                     658
1 row selected.
```

17.175 STATEMENT_VIEW_SCN_GCN

Syntax

```
STATEMENT_VIEW_SCN_GCN()
```

Description

It obtains the Global Change Number (GCN) value of the current STATEMENT's VIEW SCN.

Example

```
gSQL> SELECT STATEMENT_VIEW_SCN_GCN() FROM dual;
STATEMENT_VIEW_SCN_GCN()
------------------------
                   17697
1 row selected.
```

17.176 STATEMENT_VIEW_SCN_LCN

Syntax

```
STATEMENT_VIEW_SCN_LCN()
```

Description

It obtains the Local Change Number (LCN) value of the current STATEMENT's VIEW SCN.

Example

```
gSQL> SELECT STATEMENT_VIEW_SCN_LCN() FROM dual;
STATEMENT_VIEW_SCN_LCN()
------------------------
                   17880
1 row selected.
```

17.177 STDDEV

Syntax

```
STDDEV( [ ALL ¦ DISTINCT ] expr )
```

Description

It is an aggregation function, and it obtains the standard deviation of an expr set.

If ALL is specified, this function is performed for all values. If DISTINCT is specified, this function is perfor med for the values of which the duplicates were deleted from. If it is not specified, it is processed as if AL L is apecified.

If the number of expr sets except for NULL after deleting the duplicates by using DISTINCT is one, then it r eturns 0 like as **VARIANCE**.

The following table describes the arguments and result types.

Table 17-23 Argument and result type of STDDEV

expr	Result type
NATIVE_INTEGER family • NATIVE_SMALLINT • NATIVE_INTEGER • NATIVE_BIGINT	NATIVE_DOUBLE
NUMBER	NUMBER
NATIVE_DOUBLE family • NATIVE_REAL • NATIVE_DOUBLE	NATIVE_DOUBLE

GOLDILOCKS gets the standard deviation as follows.

- If the number of expr sets are 1, then it returns 0.
- If the number of expr sets are bigger than 1, It returns the value of **STDDEV_SAMP(expr)**.

The standard deviation is a positive square root of a variance, and it is obtained calculating the s quare root of the variance. In other words, the STDDEV function is as same as the square root o f **VARIANCE** function.

```
STDDEV( [ ALL ] expr )
 = SQRT( VARIANCE( [ ALL ] expr ) )

STDDEV( DISTINCT expr )
 = SQRT( VARIANCE( DISTINCT expr ) )
```

Example

```
gSQL> SELECT STDDEV(c1) FROM t1;
      STDDEV(C1)
----------------
11.4978258814438
1 row selected.
gSQL> SELECT STDDEV(ALL c1) FROM t1;
  STDDEV(ALL C1)
----------------
11.4978258814438
1 row selected.
gSQL> SELECT STDDEV(DISTINCT c1) FROM t1;
STDDEV(DISTINCT C1)
-------------------
   13.2759180473518
1 row selected.
```

17.178 STDDEV() OVER

Syntax

```
STDDEV ( expr ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function STDDEV calculates the standard deviation of expr.
If the number of expr except for NULL is one, then it returns 0 as a result.

Example

```
gSQL> SELECT min_price AS "MIN_PRICE"
             , STDDEV( min_price ) OVER ( ORDER BY min_price ) AS "STDDEV"
        FROM product_information
       WHERE supplier_id = 102050;
MIN_PRICE           STDDEV
--------- ----------------
       73                0
      247 123.036579926459
      731 340.953564775811
     null 340.953564775811
     null 340.953564775811
5 rows selected.
```

17.179 STDDEV_POP

Syntax

```
STDDEV_POP( expr )
```

Description

It is an aggregation function, and it obtains the population standard deviation of an expr set.
If the number of expr sets except for NULL is one, then it returns 0.

The following table describes the arguments and result types.

Table 17-24 Argument and result type of STDDEV_POP

expr	Result type
NATIVE_INTEGER family • NATIVE_SMALLINT • NATIVE_INTEGER • NATIVE_BIGINT	NATIVE_DOUBLE
NUMBER	NUMBER
NATIVE_DOUBLE family • NATIVE_REAL • NATIVE_DOUBLE	NATIVE_DOUBLE

The population standard deviation is a positive square root of a population variance, and it is ob tained by calculating the square root of the population variance. In other words, the STDDEV_P OP function is as same as the square root of **VAR_POP** function.

STDDEV_POP(expr)
= SQRT(VAR_POP(expr))

Example

```
gSQL> SELECT STDDEV_POP(c1) FROM t1;
 STDDEV_POP(C1)
---------------
10.283968105746
1 row selected.
```

17.180 STDDEV_POP() OVER

Syntax

```
STDDEV_POP ( expr ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function STDDEV_POP calculates the population standard deviation of expr.
If the number of expr except for NULL is one, then it returns 0 as a result.

Example

```
gSQL> SELECT min_price AS "MIN_PRICE"
             , STDDEV_POP( min_price ) OVER ( ORDER BY min_price ) AS "STDDEV_POP"
        FROM product_information
       WHERE supplier_id = 102050;
MIN_PRICE      STDDEV_POP
--------- ---------------
       73               0
      247              87
      731 278.38741989457
     null 278.38741989457
     null 278.38741989457
5 rows selected.
```

17.181 STDDEV_SAMP

Syntax

```
STDDEV_SAMP( expr )
```

Description

It is an aggregation function, and it obtains the sample standard deviation of an expr set. If the number o f expr sets except for NULL is one, then it returns NULL.

The following table describes the arguments and result types.

Table 17-25 Argument and result type of STDDEV_SAMP

expr	Result type
NATIVE_INTEGER family • NATIVE_SMALLINT • NATIVE_INTEGER • NATIVE_BIGINT	NATIVE_DOUBLE
NUMBER	NUMBER
NATIVE_DOUBLE family • NATIVE_REAL • NATIVE_DOUBLE	NATIVE_DOUBLE

The sample standard deviation is a positive square root of a sample variance, and it is obtained by calculating the square root of the sample variance. In other words, the STDDEV_SAMP functi on is as same as the square root of **VAR_SAMP** function.

STDDEV_SAMP(expr)
= SQRT(VAR_SAMP(expr))

Example

```
gSQL> SELECT STDDEV_SAMP(c1) FROM t1;
 STDDEV_SAMP(C1)
----------------
11.4978258814438
1 row selected.
```

17.182 STDDEV_SAMP() OVER

Syntax

```
STDDEV_SAMP ( expr ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function STDDEV_SAMP calculates the sample standard deviation of expr.
If the number of expr except for NULL is one, then it returns NULL as a result.

Example

```
gSQL> SELECT min_price AS "MIN_PRICE"
             , STDDEV_SAMP( min_price ) OVER ( ORDER BY min_price ) AS "STDDEV_SAMP"
        FROM product_information
       WHERE supplier_id = 102050;
MIN_PRICE      STDDEV_SAMP
--------- ----------------
       73             null
      247 123.036579926459
      731 340.953564775811
     null 340.953564775811
     null 340.953564775811
5 rows selected.
```

17.183 STRING_AGG() OVER

Syntax

```
STRING_AGG( str [, delimiter] ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function STRING_AGG connects str according to the function's execution range defined in OVE R clause.

If str is NULL, then it is excluded.

A delimiter is str connection delimiter, and if it is omitted the default value is NULL.

str can be a character string or a binary string.
If str is a character string, then the result type is varchar.
If str is a binary string, then the result type is varbinary.

Example

```
gSQL>
SELECT regionkey,
       name,
       STRING_AGG( name ) OVER ( PARTITION BY regionkey
                                 ORDER BY nationkey
                                 ROWS BETWEEN UNBOUNDED PRECEDING
                                          AND CURRENT ROW )
       AS "STRING_AGG( name ) OVER",
       STRING_AGG( name, ', ' ) OVER ( PARTITION BY regionkey
                                       ORDER BY nationkey
                                       ROWS BETWEEN UNBOUNDED PRECEDING
                                                AND CURRENT ROW )
       AS "STRING_AGG( name, ', ' ) OVER"
  FROM nation;
```

```
REGIONKEY NAME    STRING_AGG( name ) OVER STRING_AGG( name, ', ' ) OVER
--------- ------- ----------------------- -----------------------------
        1 BRAZIL  BRAZIL                  BRAZIL
        1 CANADA  BRAZILCANADA            BRAZIL, CANADA
        1 PERU    BRAZILCANADAPERU        BRAZIL, CANADA, PERU
        1 null    BRAZILCANADAPERU        BRAZIL, CANADA, PERU
        2 null    null                    null
        2 INDIA   INDIA                   INDIA
        2 null    INDIA                   INDIA
        2 null    INDIA                   INDIA
        2 JAPAN   INDIAJAPAN              INDIA, JAPAN
        2 CHINA   INDIAJAPANCHINA         INDIA, JAPAN, CHINA
        2 null    INDIAJAPANCHINA         INDIA, JAPAN, CHINA
        2 VIETNAM INDIAJAPANCHINAVIETNAM  INDIA, JAPAN, CHINA, VIETNAM
        3 EGYPT   EGYPT                   EGYPT
        3 IRAN    EGYPTIRAN               EGYPT, IRAN
        3 IRAQ    EGYPTIRANIRAQ           EGYPT, IRAN, IRAQ
15 rows selected.
```

17.184 SUBSTR

Syntax

```
SUBSTR( str FROM start_position [ FOR string_length ] )
SUBSTR( str, start_position [ , string_length ] )
```

Description

It is an alias of **SUBSTRING**.

Example

- Multi byte character set (e.g. UTF8): 1 byte character

```
gSQL> SELECT
      SUBSTR( 'DATABASE MANAGEMENT SYSTEM', 10, 10 ) AS RESULT
      FROM DUAL;
RESULT
----------
MANAGEMENT
1 row selected.
```

- Multi byte character set (e.g. UTF8): 2 bytes or 3 bytes character

```
gSQL> SELECT SUBSTR( '“αβ≠AB”', 2, 5 ) AS RESULT FROM DUAL;
RESULT
------
αβ≠AB
1 row selected.
```

17.185 SUBSTRB

Syntax

```
SUBSTRB( str, start_position [ , string_length ] )
```

Description

It extracts characters which are within string_length range from start_position, and returns the result for s tr.

This function is as same as **SUBSTRING** function, except that start_position and string_length of the SUBS TR function are calculated in byte units.

Example

- Multi byte character set (e.g. UTF8): 1 byte character

```
gSQL> SELECT
      SUBSTRB( 'DATABASE MANAGEMENT SYSTEM', 10, 10 ) AS RESULT
      FROM DUAL;
RESULT
----------
MANAGEMENT
1 row selected.
```

- Multi byte character set (e.g. UTF8): 2 bytes or 3 bytes character

```
gSQL> SELECT SUBSTRB( '“αβ≠AB”', 4, 11 ) AS RESULT FROM DUAL;
RESULT
------
αβ≠AB
1 row selected.
```

17.186 SUBSTRING

Syntax

```
SUBSTRING( str FROM start_position [ FOR string_length ] )
SUBSTRING( str, start_position [ , string_length ] )
```

Description

It extracts characters which are within string_length range from start_position, and returns the result for s tr.

The str argument data type can be a character data type such as CHARACTER, CHARACTER VARYING, C HARACTER LONG VARYING, or a binary data type such as BINARY, BINARY VARYING, BINARY LONG VA RYING.

The start_position argument and string_length argument can be a numeric data type.

If any of str, start_position, string_length is NULL, the result is NULL.
The start_position and string_length start from 1, and they are calculated in character unit according to c haracter set (not in byte unit).

If start_position is 0, the start_position is assigned to 1.
If start_position is a positive number, it searches for the position forwards (towards right) from the begin ning of str.
If start_ position is a negative number, it searches for the position backwards (towards left) from the end of str.
If string_length is omitted, characters from the start_position to the last character of str, is returned.

If string_length is 0 or a negative number, the result is NULL.
If start_position > (str length), the result is NULL.
If (str length + start_position) < 0, the result is NULL.

It is an alias of **SUBSTR**.
For more information, refer to **SUBSTRB**.

The following table describes the result types.

Table 17-26 Result type of SUBSTRING

str type	Result type
CHAR or VARCHAR	VARCHAR
LONG VARCHAR	LONG VARCHAR
BINARY or VARBINARY	VARBINARY
LONG VARBINARY	LONG VARBINARY

Example

- Multi byte character set (e.g. UTF8): 1 byte character

```
gSQL> SELECT
      SUBSTRING( 'DATABASE MANAGEMENT SYSTEM' FROM 10 FOR 10 ) AS RESULT
      FROM DUAL;
RESULT
----------
MANAGEMENT
1 row selected.
```

- Multi byte character set (e.g. UTF8): 2 bytes or 3 bytes character

```
gSQL> SELECT SUBSTRING( '“αβ≠AB”' FROM 2 FOR 5 ) AS RESULT FROM DUAL;
RESULT
------
αβ≠AB
1 row selected.
```

17.187 SUM

Syntax

```
SUM( [ ALL ¦ DISTINCT ] expr )
```

Description

It is an aggregate function and the sum of expr value is obtained.

If ALL is explicitly specified, aggregation is executed for all values.
If DISTINCT is explicitly specified, aggregation is executed for the values which exclude duplicate values.
If ALL or DISTINCT is not explicitly specified, it is processed in the same way as when ALL is specified.

Example

```
gSQL> SELECT SUM(c1) FROM t1;
SUM(C1)
-------
      6
1 row selected.
```

17.188 SUM() OVER

Syntax

```
SUM ( expr ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function SUM calculates the sum of expr value.
NULL is excluded from the calculation.

Example

```
gSQL> SELECT min_price AS "MIN_PRICE"
             , SUM( min_price ) OVER ( ORDER BY min_price ) AS "SUM"
        FROM product_information
       WHERE supplier_id = 102050;
MIN_PRICE  SUM
--------- ----
       73   73
      247  320
      731 1051
     null 1051
     null 1051
5 rows selected.
```

17.189 SYSDATE

Syntax

```
SYSDATE
```

Description

It obtains the current DATE type value based on the OS time of the database server.

Example

```
gSQL> SELECT SYSDATE FROM t1;
SYSDATE
----------
2013-12-12
2013-12-12
2013-12-12
3 rows selected.
```

17.190 SYS_EXTRACT_UTC

Syntax

```
SYS_EXTRACT_UTC( datetime_with_timezone )
```

Description

It returns the UTC (Coordinated Universal Time—formerly Greenwich Mean Time) value.
If the timezone is not specified, it is calculated as session time zone.

The data type of an input argument can be time, time with time zone, timestamp, timestamp with time z one.
The result type is time or timestamp type.

Example

```
gSQL> SELECT
      SYS_EXTRACT_UTC(
          TO_TIMESTAMP_TZ( '2017-05-25 00:00:00.000000 +09:00',
                           'YYYY-MM-DD HH24:MI:SS.FF6 TZH:TZM' )
          ) AS RESULT
      FROM DUAL;
RESULT
--------------------------
2017-05-24 15:00:00.000000
1 row selected.
```

17.191 SYSTIME

Syntax

```
SYSTIME
```

Description

It obtains the current TIME WITH TIME ZONE type value based on the OS time of the database server.

Example

```
gSQL> SELECT SYSTIME FROM t1;
SYSTIME
----------------------
16:30:46.954941 +09:00
16:30:46.954941 +09:00
16:30:46.954941 +09:00
3 rows selected.
```

17.192 SYSTIMESTAMP

Syntax

```
SYSTIMESTAMP
```

Description

It obtains the current TIMESTAMP WITH TIME ZONE type value based on the OS time of the database ser ver.

Example

```
gSQL> SELECT SYSTIMESTAMP FROM t1;
SYSTIMESTAMP
---------------------------------
2013-12-12 16:37:34.432241 +09:00
2013-12-12 16:37:34.432241 +09:00
2013-12-12 16:37:34.432241 +09:00
3 rows selected.
```

17.193 TAN

Syntax

```
TAN( num )
```

Description

It returns the tangent value of num in radians unit.
If num is NULL, then NULL is returned.

Example

```
gSQL> SELECT TAN( 1 ) AS RESULT FROM DUAL;
         RESULT
---------------
1.5574077246549
1 row selected.
```

17.194 TO_BASE64

Syntax

```
TO_BASE64( str )
```

Description

It converts str by using base64 encoding, and returns the converted character.
str argument can be a character type such as CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING, a type which can be converted to a character type, or a binary character type such as BINARY, BINARY VARYING, BINARY LONG VARYING.
The result type is a character such as CHARACTER VARYING or CHARACTER LONG VARYING.

If str is NULL, the result value is also NULL.

Base64 encoding represents 8 bit binary data in 64 characters consisting of ascii areas.
64 characters consist of A~Z, a~z, 0~9, +, /.

6 bit is represented as a character, and three characters (24 bits) are represented with 4 characters as a unit.
If the encoded characters can not fill 4 characters, then others are filled with '='.
If encoded characters are over 76, then a newline is added and they are divided into multiple lines.

Use FROM_BASE64() function to decode the base64 encoded character.
The newline, carriage return, tab, space are ignored when decoding base64.

For more information, refer to **FROM_BASE64**.

Example

```
gSQL> SELECT TO_BASE64( 'abc' ), TO_BASE64( 'abcd' ) FROM DUAL;
TO_BASE64( 'abc' ) TO_BASE64( 'abcd' )
------------------ -------------------
YWJj               YWJjZA==
1 row selected.
```

17.195 TO_CHAR(datetime)

Syntax

```
TO_CHAR( datetime [, fmt ] )
```

Description

It converts datetime to a string in the specified fmt format, and returns the result.

The datetime argument data type can be DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE, TIME, TIME WITH TIME ZONE, INTERVAL.
The fmt argument data type can be a character data type such as CHARACTER, CHARACTER VARYING.
If any argument is NULL, then NULL is returned.

If fmt is omitted, it follows the default format.
- DATE: Refer to **NLS_DATE_FORMAT**.
- TIMESTAMP: Refer to **NLS_TIMESTAMP_FORMAT**.
- TIMESTAMP WITH TIME ZONE: Refer to **NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT**.
- TIME: Refer to **NLS_TIME_FORMAT**.
- TIME WITH TIME ZONE: Refer to **NLS_TIME_WITH_TIME_ZONE_FORMAT**.

If the data type of the datetime argument is INTERVAL, it is converted to a string then returned regardless of fmt.

For more information about the string which can be specified in fmt, refer to **Datetime Format String**.

The result type is CHARACTER VARYING.

Example

The following is an example of when fmt is omitted, and NLS_DATE_FORMAT = 'YYYY-MM-DD'.

```
gSQL> SELECT
      TO_CHAR( TO_DATE( '2012-03-15','YYYY-MM-DD' ) ) AS RESULT
      FROM DUAL;
```

```
RESULT
----------
2012-03-15
1 row selected.
```

The following is an example of when fmt is specified.

```
gSQL> SELECT
      TO_CHAR( TO_DATE('2012-03-15','YYYY-MM-DD'), 'DD-MON-YY' ) AS RESULT
      FROM DUAL;
RESULT
---------
15-MAR-12
1 row selected.
```

17.196 TO_CHAR(number)

Syntax

```
TO_CHAR( number [, fmt ] )
```

Description

It converts the number to a string in the specified fmt format, and returns the result.

The number argument can be a numeric data type.
The fmt argument data type can be a character data type such as CHARACTER, CHARACTER VARYING.
If fmt is omitted, all significant digits are converted to the string and returned.
For more information about the string which can be specified in fmt, refer to **Number Format String**.
If any input argument is NULL, then NULL is returned.

The result type is CHARACTER VARYING.

Example

```
gSQL> SELECT TO_CHAR( 12500000 ) AS RESULT FROM DUAL;
RESULT
--------
12500000
1 row selected.
gSQL> SELECT TO_CHAR( 12500000, 'S999,999,999' ) AS RESULT FROM DUAL;
RESULT
------------
 +12,500,000
1 row selected.
```

17.197 TO_DATE

Syntax

```
TO_DATE( str [, fmt ] )
```

Description

It converts the str string in the specified fmt format to DATE type, and returns the result.

The str argument and fmt argument data type can be a character data type such as CHARACTER, CHARA CTER VARYING, CHARACTER LONG VARYING.
If fmt is omitted, the default format is NLS_DATE_FORMAT, and in this case str should be the default for mat string.

For more information about the string which can be specified in fmt, refer to **Datetime Format String**.
For more information, refer to **NLS_DATE_FORMAT**.
If either str or fmt is NULL, then NULL is returned.

The result type is DATE.

Example

The following is an example of when fmt is omitted, and NLS_DATE_FORMAT = 'YYYY-MM-DD'.

```
gSQL> SELECT TO_DATE( '2009-07-29' ) AS RESULT FROM DUAL;
RESULT
----------
2009-07-29
1 row selected.
```

The following is an example of when fmt is specified.

```
gSQL> SELECT TO_DATE( '29-JUL-09', 'DD-MON-YY' ) AS RESULT FROM DUAL;
RESULT
----------
```

```
2009-07-29
1 row selected.
```

17.198 TO_NATIVE_BIGINT

Syntax

```
TO_NATIVE_BIGINT( str [, fmt ] )
```

Description

It converts the str string in the specified fmt format to NATIVE_BIGINT type, and returns the result.

The str argument and fmt argument data type can be a character data type such as CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING.

If any of str, fmt is NULL, the result is also NULL.
For more information about the string which can be specified in fmt, refer to **Number Format String**.

The result type is NATIVE_BIGINT.

Example

```
gSQL> SELECT TO_NATIVE_BIGINT( '123.45' ) AS RESULT1,
             TO_NATIVE_BIGINT( '+123.45', 'S999.99' ) AS RESULT2
        FROM DUAL;
RESULT1 RESULT2
------- -------
    123     123
1 row selected.
```

17.199 TO_NATIVE_DOUBLE

Syntax

```
TO_NATIVE_DOUBLE( str [, fmt ] )
```

Description

It converts the str string in the specified fmt format to NATIVE_DOUBLE type, and returns the result.

The data type of str argument and fmt argument can be a character data type such as CHARACTER, CHA RACTER VARYING, CHARACTER LONG VARYING.

If any of str, fmt is NULL, the result is also NULL.
For more information about the string which can be specified in fmt, refer to **Number Format String**.

The result type is NATIVE_DOUBLE.

Example

```
gSQL> SELECT TO_NATIVE_DOUBLE( '123.45' ) AS RESULT1,
             TO_NATIVE_DOUBLE( '+123.45', 'S999.99' ) AS RESULT2
        FROM DUAL;
RESULT1 RESULT2
------- -------
 123.45  123.45
1 row selected.
```

17.200 TO_NATIVE_INTEGER

Syntax

```
TO_NATIVE_INTEGER( str [, fmt ] )
```

Description

It converts the str string in the specified fmt format to NATIVE_INTEGER type, and returns the result.

The data type of str argument and fmt argument can be a character data type such as CHARACTER, CHA RACTER VARYING, CHARACTER LONG VARYING.

If any of str, fmt is NULL, the result is also NULL.
For more information about the string which can be specified in fmt, refer to **Number Format String**.

The result type is NATIVE_INTEGER.

Example

```
gSQL> SELECT TO_NATIVE_INTEGER( '123.45' ) AS RESULT1,
             TO_NATIVE_INTEGER( '+123.45', 'S999.99' ) AS RESULT2
        FROM DUAL;
RESULT1 RESULT2
------- -------
    123     123
1 row selected.
```

17.201 TO_NATIVE_REAL

Syntax

```
TO_NATIVE_REAL( str [, fmt ] )
```

Description

It converts the str string in the specified fmt format to NATIVE_REAL type, and returns the result.

The data type of str argument and fmt argument can be a character data type such as CHARACTER, CHA RACTER VARYING, CHARACTER LONG VARYING.

If any of str, fmt is NULL, the result is also NULL.
For more information about the string which can be specified in fmt, refer to **Number Format String**.

The result type is NATIVE_REAL.

Example

```
gSQL> SELECT TO_NATIVE_REAL( '123.45' ) AS RESULT1,
             TO_NATIVE_REAL( '+123.45', 'S999.99' ) AS RESULT2
        FROM DUAL;
RESULT1 RESULT2
------- -------
 123.45  123.45
```

17.202 TO_NATIVE_SMALLINT

Syntax

```
TO_NATIVE_SMALLINT( str [, fmt ] )
```

Description

It converts the str string in the specified fmt format to NATIVE_SMALLINT type, and returns the result.

The data type of str argument and fmt argument can be a character data type such as CHARACTER, CHA RACTER VARYING, CHARACTER LONG VARYING.

If any of str, fmt is NULL, the result is also NULL.
For more information about the string which can be specified in fmt, refer to **Number Format String**.

The result type is NATIVE_SMALLINT.

Example

```
gSQL> SELECT TO_NATIVE_SMALLINT( '123.45' ) AS RESULT1,
             TO_NATIVE_SMALLINT( '+123.45', 'S999.99' ) AS RESULT2
        FROM DUAL;
RESULT1 RESULT2
------- -------
    123     123
1 row selected.
```

17.203 TO_NUMBER

Syntax

```
TO_NUMBER( str [, fmt] )
```

Description

It converts the str string in the specified fmt format to NUMBER type, and returns the result.

The data type of str argument and fmt argument can be a character data type such as CHARACTER, CHA RACTER VARYING, CHARACTER LONG VARYING.

If any of str, fmt is NULL, the result is also NULL.
For more information about the string which can be specified in fmt, refer to **Number Format String**.

The result type is NUMBER.

Example

```
gSQL> SELECT TO_NUMBER( '123.45' ) AS RESULT1,
             TO_NUMBER( '+123.45', 'S999.99' ) AS RESULT2
        FROM DUAL;
RESULT1 RESULT2
------- -------
 123.45  123.45
1 row selected.
```

17.204 TO_TIME

Syntax

```
TO_TIME( str [, fmt ] )
```

Description

It converts the str string in the specified fmt format to TIME type, and returns the result.

The data type of str argument and fmt argument can be a character data type such as CHARACTER, CHA RACTER VARYING, CHARACTER LONG VARYING.
If fmt is omitted, the default format is NLS_TIME_FORMAT, and in this case str should be the default for mat string.

For more information about the string which can be specified in fmt, refer to **Datetime Format String**.
For more information, refer to **NLS_TIME_FORMAT**.
If either str or fmt is NULL, then NULL is returned.

The result type is TIME.

Example

The following is an example of when fmt is omitted, and NLS_TIME_FORMAT = 'HH24:MI:SS.FF6'.

```
gSQL> SELECT TO_TIME( '11:22:33.999999' ) AS RESULT FROM DUAL;
RESULT
---------------
11:22:33.999999
1 row selected.
```

The following is an example of when fmt is specified.

```
gSQL> SELECT
      TO_TIME( '112233.999999/P.M.', 'HH12MISS.FF6/P.M.' ) AS RESULT
      FROM DUAL;
```

```
RESULT
---------------
23:22:33.999999
1 row selected.
```

17.205 TO_TIME_TZ

Syntax

```
TO_TIME_TZ( str [, fmt ] )
```

Description

It is an alias of **TO_TIME_WITH_TIME_ZONE**.
For more information, refer to **NLS_TIME_WITH_TIME_ZONE_FORMAT**.

Example

The following is an example of when fmt is omitted, and NLS_TIME_WITH_TIME_ZONE_FORMAT = 'HH24:MI:SS.FF6 TZH:TZM'.

```
gSQL> SELECT TO_TIME_TZ( '11:22:33.999999 +09:00' ) AS RESULT FROM DUAL;
RESULT
----------------------
11:22:33.999999 +09:00
1 row selected.
```

The following is an example of when fmt is specified.

```
gSQL> SELECT TO_TIME_TZ( '11:22:33.999999 +09:00 PM',
                         'HH12:MI:SS.FF6 TZH:TZM PM' ) AS RESULT
      FROM DUAL;
RESULT
----------------------
23:22:33.999999 +09:00
1 row selected.
```

17.206 TO_TIME_WITH_TIME_ZONE

Syntax

```
TO_TIME_WITH_TIME_ZONE( str [, fmt ] )
TO_TIME_TZ( str [, fmt ] )
```

Description

It converts the str string in the specified fmt format to TIME WITH TIME ZONE type, and returns the result.

The data type of str argument and fmt argument can be a character data type such as CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING.
If fmt is omitted, the default format is NLS_TIME_WITH_TIME_ZONE_FORMAT, and in this case str should be the default format string.

For more information about the string which can be specified in fmt, refer to **Datetime Format String**
For more information, refer to **NLS_TIME_WITH_TIME_ZONE_FORMAT**.
If either str or fmt is NULL, then NULL is returned.

It is an alias of **TO_TIME_TZ**.

The result type is TIME WITH TIME ZONE.

Example

The following is an example of when fmt is omitted, and NLS_TIME_WITH_TIME_ZONE_FORMAT = 'HH24:MI:SS.FF6 TZH:TZM'.

```
gSQL> SELECT
      TO_TIME_WITH_TIME_ZONE( '11:22:33.999999 +09:00' ) AS RESULT
      FROM DUAL;
RESULT
----------------------
11:22:33.999999 +09:00
1 row selected.
```

The following is an example of when fmt is specified.

```
gSQL> SELECT
      TO_TIME_WITH_TIME_ZONE( '11:22:33.999999 +09:00 PM',
                              'HH12:MI:SS.FF6 TZH:TZM PM' )
      AS RESULT
      FROM DUAL;
RESULT
----------------------
23:22:33.999999 +09:00
1 row selected.
```

17.207 TO_TIMESTAMP

Syntax

```
TO_TIMESTAMP( str [, fmt ] )
```

Description

It converts the str string in the specified fmt format to TIMESTAMP type, and returns the result.

The data type of str argument and fmt argument can be a character data type such as CHARACTER, CHA RACTER VARYING, CHARACTER LONG VARYING.
If fmt is omitted, the default format is NLS_TIMESTAMP_FORMAT, and in this case str should be the defa ult format string.

For more information about the string which can be specified in fmt, refer to **Datetime Format String**.
For more information, refer to **NLS_TIMESTAMP_FORMAT**.
If either str or fmt is NULL, then NULL is returned.

The result type is TIMESTAMP.

Example

The following is an example of when fmt is omitted, and NLS_TIMESTAMP_FORMAT = 'YYYY-MM-DD H H24:MI:SS.FF6'.

```
gSQL> SELECT
      TO_TIMESTAMP( '2009-07-29 11:22:33.999999' ) AS RESULT
      FROM DUAL;
RESULT
--------------------------
2009-07-29 11:22:33.999999
1 row selected.
```

The following is an example of when fmt is specified.

```
gSQL> SELECT
      TO_TIMESTAMP( '090729 112233999999 PM', 'YYMMDD HH12MISSFF6 PM' )
      AS RESULT
      FROM DUAL;
RESULT
--------------------------
2009-07-29 23:22:33.999999
1 row selected.
```

17.208 TO_TIMESTAMP_TZ

Syntax

```
TO_TIMESTAMP_TZ( str [, fmt ] )
```

Description

It is an alias of **TO_TIMESTAMP_WITH_TIME_ZONE**.
For more information, refer to **NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT**.

Example

The following is an example of when fmt is omitted, and NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT = 'YYYY-MM-DD HH24:MI:SS.FF6 TZH:TZM'.

```
gSQL> SELECT
      TO_TIMESTAMP_TZ( '2009-07-29 11:22:33.999999 +09:00' ) AS RESULT
      FROM DUAL;
RESULT
--------------------------------
2009-07-29 11:22:33.999999 +09:00
1 row selected.
```

The following is an example of when fmt is specified.

```
gSQL> SELECT
      TO_TIMESTAMP_TZ( '29-JUL-09 11:22:33.999999 +09:00',
                       'DD-MON-RR HH12:MI:SS.FF6 TZH:TZM' ) AS RESULT
      FROM DUAL;
RESULT
--------------------------------
2009-07-29 11:22:33.999999 +09:00
1 row selected.
```

17.209 TO_TIMESTAMP_WITH_TIME_ZONE

Syntax

```
TO_TIMESTAMP_WITH_TIME_ZONE( str [, fmt ] )
TO_TIMESTAMP_TZ( str [, fmt ] )
```

Description

It converts the str string in the specified fmt format to TIMESTAMP WITH TIME ZONE type, and returns th e result.

The data type of str argument and fmt argument can be a character data type such as CHARACTER, CHA RACTER VARYING, CHARACTER LONG VARYING.
If fmt is omitted, the default format is NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT, and in this case str should be the default format string.

For more information about the string which can be specified in fmt, refer to **Datetime Format String**.
For more information, refer to **NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT**.
If either str or fmt is NULL, then NULL is returned.

It is an alias of **TO_TIMESTAMP_TZ**.

The result type is TIMESTAMP WITH TIME ZONE .

Example

The following is an example of when fmt is omitted, and NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT = 'YYYY-MM-DD HH24:MI:SS.FF6 TZH:TZM'.

```
gSQL> SELECT
      TO_TIMESTAMP_WITH_TIME_ZONE( '2009-07-29 11:22:33.999999 +09:00' )
      AS RESULT
      FROM DUAL;
RESULT
-------------------------------
```

```
2009-07-29 11:22:33.999999 +09:00
1 row selected.
```

The following is an example of when fmt is specified.

```
gSQL> SELECT
      TO_TIMESTAMP_WITH_TIME_ZONE( '29-JUL-09 11:22:33.999999 +09:00',
                                   'DD-MON-RR HH12:MI:SS.FF6 TZH:TZM' )
      AS RESULT
      FROM DUAL;
RESULT
--------------------------------
2009-07-29 11:22:33.999999 +09:00
1 row selected.
```

17.210 TRANSACTION_DATE

Syntax

```
TRANSACTION_DATE()
```

Description

It obtains the current date (DATE type) value based on the session time.

The differences among the functions to obtain the current date are as follows.

- TRANSACTION_DATE(): All date values in the transaction are same.
- STATEMENT_DATE(): All date values in an SQL statement are same.
- CLOCK_DATE(): Whenever the function is called, the current date value is obtained.

Example

All date values are always same within a single transaction.

```
gSQL> SELECT TRANSACTION_DATE() FROM dual;
TRANSACTION_DATE()
------------------
2013-12-12
1 row selected.
gSQL> SELECT TRANSACTION_DATE() FROM dual;
TRANSACTION_DATE()
------------------
2013-12-12
1 row selected.
gSQL> COMMIT;
Commit complete.
gSQL> SELECT TRANSACTION_DATE() FROM dual;
TRANSACTION_DATE()
------------------
2013-12-13
1 row selected.
```

17.211 TRANSACTION_LOCALTIME

Syntax

```
TRANSACTION_LOCALTIME()
```

Description

It obtains the current TIME WITHOUT TIME ZONE type value based on the session time.

The differences among the functions to obtain the current time are as follows.

- TRANSACTION_LOCALTIME(): All time values in the transaction are same.
- STATEMENT_LOCALTIME(): All time values in an SQL statement are same.
- CLOCK_LOCALTIME(): Whenever the function is called, the current time value is obtained.

Example

All time values are always same within a single transaction.

```
gSQL> SELECT TRANSACTION_LOCALTIME() FROM dual;
TRANSACTION_LOCALTIME()
-----------------------
16:43:24.391834
1 row selected.
gSQL> SELECT TRANSACTION_LOCALTIME() FROM dual;
TRANSACTION_LOCALTIME()
-----------------------
16:43:24.391834
1 row selected.
gSQL> COMMIT;
Commit complete.
gSQL> SELECT TRANSACTION_LOCALTIME() FROM dual;
TRANSACTION_LOCALTIME()
-----------------------
16:43:32.651833
1 row selected.
```

17.212 TRANSACTION_LOCALTIMESTAMP

Syntax

```
TRANSACTION_LOCALTIMESTAMP()
```

Description

It obtains the current TIMESTAMP WITHOUT TIME ZONE type value based on the session time.

The differences among the functions to obtain the current timestamp are as follows.

- TRANSACTION_LOCALTIMESTAMP(): All timestamp values in the transaction are same.
- STATEMENT_LOCALTIMESTAMP(): All timestamp values in an SQL statement are same.
- CLOCK_LOCALTIMESTAMP(): Whenever the function is called, the current timestamp value is obtained.

Example

All timestamp values are always same within a single transaction.

```
gSQL> SELECT TRANSACTION_LOCALTIMESTAMP() FROM dual;
TRANSACTION_LOCALTIMESTAMP()
----------------------------
2013-12-12 16:43:32.651833
1 row selected.
gSQL> SELECT TRANSACTION_LOCALTIMESTAMP() FROM dual;
TRANSACTION_LOCALTIMESTAMP()
----------------------------
2013-12-12 16:43:32.651833
1 row selected.
gSQL> COMMIT;
Commit complete.
gSQL> SELECT TRANSACTION_LOCALTIMESTAMP() FROM dual;
TRANSACTION_LOCALTIMESTAMP()
----------------------------
2013-12-12 16:46:07.831834
1 row selected.
```

17.213 TRANSACTION_TIME

Syntax

```
TRANSACTION_TIME()
```

Description

It obtains the current TIME WITH TIME ZONE type value based on the session time.

The differences among the functions to obtain the current time are as follows.

- TRANSACTION_TIME(): All time values in the transaction are same.
- STATEMENT_TIME(): All time values in an SQL statement are same.
- CLOCK_TIME(): Whenever the function is called, the current time value is obtained.

Example

All time values are always same within a single transaction.

```
gSQL> SELECT TRANSACTION_TIME() FROM dual;
TRANSACTION_TIME()
----------------------
16:46:07.831834 +09:00
1 row selected.
gSQL> SELECT TRANSACTION_TIME() FROM dual;
TRANSACTION_TIME()
----------------------
16:46:07.831834 +09:00
1 row selected.
gSQL> COMMIT;
Commit complete.
gSQL> SELECT TRANSACTION_TIME() FROM dual;
TRANSACTION_TIME()
----------------------
16:48:00.691827 +09:00
1 row selected.
```

17.214 TRANSACTION_TIMESTAMP

Syntax

```
TRANSACTION_TIMESTAMP()
```

Description

It obtains the current TIMESTAMP WITH TIME ZONE type value based on the session time.

The differences among the functions to obtain the current timestamp are as follows.

- TRANSACTION_TIMESTAMP(): All timestamp values in the transaction are same.
- STATEMENT_TIMESTAMP(): All timestamp values in an SQL statement are same.
- CLOCK_TIMESTAMP(): Whenever the function is called, the current timestamp value is obtained.

Example

All timestamp values are always same within a single transaction.

```
gSQL> SELECT TRANSACTION_TIMESTAMP() FROM dual;
TRANSACTION_TIMESTAMP()
-------------------------------
2013-12-12 16:48:00.691827 +09:00
1 row selected.
gSQL> SELECT TRANSACTION_TIMESTAMP() FROM dual;
TRANSACTION_TIMESTAMP()
-------------------------------
2013-12-12 16:48:00.691827 +09:00
1 row selected.
gSQL> COMMIT;
Commit complete.
gSQL>  SELECT TRANSACTION_TIMESTAMP() FROM dual;
TRANSACTION_TIMESTAMP()
-------------------------------
2013-12-12 16:49:26.291827 +09:00
1 row selected.
```

17.215 TRANSLATE

Syntax

```
TRANSLATE( string, from, to )
```

Description

It replaces characters. It replaces characters of string which corresponds to the the character of *from* with the character of *to* at the same position as the character of *from*.

The data type of the string argument, the from argument, and the to argument can be a data type such a s CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING.

If any of *string, from, to* is NULL, the result is also NULL.

- When the character of string which are as same as those of *from* exists,
  - If the length of *from* is as same as that of *to*, then it is replaced with the character of *to* at the sa me position as the character of *from*.
  - If the length of *from* is longer than that of *to*, then characters of *from* at the position after that o f *to* length are deleted from the string.
  - If the character of *from* is duplicate, then it is replaced with the character of *to* at the same positi on as the first duplicate character of *from*.
- When the character of string which is as same as those of *from* does not exist, then the string is not r eplaced.

The following table describes the result types.

Table 17-27 Result type of TRANSLATE

string type	Result type
CHAR or VARCHAR	VARCHAR
LONG VARCHAR	LONG VARCHAR

Example

- When the character of string which are as same as those of *from* exists, then it is replaced with the c haracter of *to* at the same position as the character of *from*.
  - A → Z, C → Y, E → X, G → W

```
gSQL> SELECT TRANSLATE('ABCDEFG', 'ACEG', 'ZYXW') AS RESULT
FROM DUAL;
RESULT
-------
ZBYDXFW
1 row selected.
```

- If the length of *from* string is longer than that of *to* string, then characters of *from* at the position aft er that of *to* string length are deleted *from* the string, and replaced.
  - A → Z, C → Y, deleting E, deleting G

```
gSQL> SELECT TRANSLATE('ABCDEFG', 'ACEG', 'ZY') AS RESULT
      FROM DUAL;
RESULT
------
ZBYDF
1 row selected.
```

17.216 TRIM

Syntax

```
TRIM([ [ LEADING ¦ TRAILING ¦ BOTH ]  [trim_character] FROM ] trim_source)
```

Description

It removes the matching characters by comparing trim_character in trim_source from the LEADING, TRAILING, BOTH direction until the matching character does not exist. Then it returns the result.

The trim_character argument and trim_source argument can be a character data type such as CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING or a binary character data type such as BINARY, BINARY VARYING, BINARY LONG VARYING.

If any of trim_character, trim_source is NULL, the result is NULL.

- [LEADING | TRAILING | BOTH]
  - LEADING: Removes trim_character from the beginning of trim_source.
  - TRAILING: Removes trim_character from the back of trim_source.
  - BOTH: Removes trim_character from the both direction (the beginning, the back) of trim_source.
- trim_character should be a single character.
- If trim_character is omitted, single blank space (' ') is specified by default.
- When FROM is specified
  - [LEADING | TRAILING | BOTH], trim_character or [LEADING | TRAILING | BOTH] trim_character should be specified.
    - e.g. TRIM(LEADING FROM ' abc') , TRIM('x' FROM 'xabc') , TRIM(LEADING 'x' FROM 'xabc')
  - If [LEADING | TRAILING | BOTH] is omitted, BOTH is specified by default.
- When FROM is omitted.
  - It is TRIM(trim_source), it is executed in the same way as TRIM(BOTH ' ' FROM trim_source).

The following table describes the result types.

Table 17-28 Result type of TRIM

trim_character, trim_source type	Result type
CHAR or VARCHAR	VARCHAR
LONG VARCHAR	LONG VARCHAR

trim_character, trim_source type	Result type
BINARY or VARBINARY	VARBINARY
LONG VARBINARY	LONG VARBINARY

Example

```
gSQL> SELECT TRIM( LEADING '_' FROM '___TRIM FUNCTION___' ) AS RESULT
      FROM DUAL;
RESULT
----------------
TRIM FUNCTION___
1 row selected.
gSQL> SELECT TRIM( TRAILING '_' FROM '___TRIM FUNCTION___' ) AS RESULT
      FROM DUAL;
RESULT
----------------
___TRIM FUNCTION
1 row selected.
gSQL> SELECT TRIM( BOTH '_' FROM '___TRIM FUNCTION___' ) AS RESULT
      FROM DUAL;
RESULT
-------------
TRIM FUNCTION
1 row selected.
```

17.217 TRUNC(number)

Syntax

```
TRUNC( num [ , scale ] )
```

Description

It truncates the num based on scale, then returns the result.

The num argument and scale argument can be a numeric type.
If either the num argument or the scale argument is NULL, then NULL is returned.

If scale is omitted, the scale becomes 0, and it is executed as same as TRUNC(num, 0).
If scale is a positive number, it is truncated based on the number of right digit of the decimal point.
If scale is a negative number, it is truncated off based on the number of left digit of the decimal point.

Example

```
gSQL> SELECT TRUNC( 142.4282, 2 ) AS RESULT FROM DUAL;
RESULT
------
142.42
1 row selected.
gSQL> SELECT TRUNC( 142.4282, -2 ) AS RESULT FROM DUAL;
RESULT
------
   100
1 row selected.
```

17.218 TRUNC(date)

Syntax

```
TRUNC( date [ , fmt ] )
```

Description

It truncates the date in a specified fmt unit, and returns the result.

The date argument data type can be DATE, TIMESTAMP, TIMESTAMP WITH TIME ZONE.
The fmt argument data type can be a character data type such as CHARACTER, CHARACTER VARYING.
If either date argument or fmt argument is NULL, then NULL is returned.

The result type is always DATE regardless of the input date type.

If fmt is omitted, the default is *DAY*, and the available format string is described in the following table.

Table 17-29 Available format string in fmt

Format string	Description
CC, SCC	Century
YYYY, YEAR, SYYYY, SYEAR, YYY, YY, Y	Year
IYYY, IYY, IY, I	The year embracing the calendar week defined by ISO 8601 standards
Q	Quarter
MONTH, MON, MM, RM	Month
WW	The week whose first week starts from January 1st of the year
IW	The week containing the first thursday of the year designated as the calendar week by ISO 8601 standards (1 ~ 52 week or 1 ~ 53 weeks) becomes the first week.
W	The week whose first week starts from the first day of the month
DDD, DD, J	Day
DAY, DY, D	Day of week
HH, HH12, HH24	Hour
MI	Minute

Example

```
gSQL> SELECT
      TRUNC( TO_DATE( '2051-07-16', 'YYYY-MM-DD' ), 'CC' ) AS RESULT
      FROM DUAL;
RESULT
----------
2001-01-01
1 row selected.
gSQL> SELECT
      TRUNC( TO_DATE( '2051-07-16', 'YYYY-MM-DD' ), 'YYYY' ) AS RESULT
      FROM DUAL;
RESULT
----------
2051-01-01
1 row selected.
gSQL> SELECT
      TRUNC( TO_DATE( '2051-07-16', 'YYYY-MM-DD' ), 'MONTH' ) AS RESULT
      FROM DUAL;
RESULT
----------
2051-07-01
1 row selected.
gSQL> SELECT
      TRUNC( TO_TIMESTAMP( '2001-05-05 11:22:33.999999',
             'YYYY-MM-DD HH24:MI:SS.FF6' ) ) AS RESULT
      FROM DUAL;
RESULT
----------
2001-05-05
1 row selected.
```

17.219 UPPER

Syntax

```
UPPER( str )
```

Description

It returns the uppercase characters of str.

The str argument can be a character data type such as CHARACTER, CHARACTER VARYING, CHARACTER LONG VARYING.
If str is NULL, the result is NULL.

The return type is as same as the str argument type.

Example

```
gSQL> SELECT UPPER( 'spring' ) AS RESULT FROM DUAL;
RESULT
------
SPRING
1 row selected.
```

17.220 UNHEX

Syntax

```
UNHEX( str )
```

Description

str argument is a hexadecimal character and this function represents it as each byte and returns it as a bin ary string.

The input argument can be a character type such as CHARACTER VARYING, CHARACTER LONG VARYIN G. The result type is a binary character type such as BINARY VARYING or BINARY LONG VARYING.

If str is NULL, then the result value is also NULL.
If str includes a character which does not belong to the hexadecimal range, then it returns an error.

For more information, refer to **HEX**.

Example

```
gSQL> SELECT UNHEX( HEX( 'abc' ) ) FROM DUAL;
UNHEX( HEX( 'abc' ) )
---------------------
616263
1 row selected.
```

17.221 UNHEX_TO_CHARSTR

Syntax

```
UNHEX_TO_CHARSTR( str )
```

Description

str argument is a hexadecimal character and this function represents it as each byte and returns it as a ch aracter string.

The input argument can be a character type such as CHARACTER VARYING, CHARACTER LONG VARYIN G. The result type is a character type such as CHARACTER VARYING, CHARACTER LONG VARYING.

If str is NULL, then the result value is also NULL.
If str includes a character which does not belong to the hexadecimal range, then it returns an error.

For more information, refer to **HEX**, **UNHEX**.

Example

```
gSQL> SELECT UNHEX_TO_CHARSTR( '616263' ) FROM DUAL;
UNHEX_TO_CHARSTR( '616263' )
----------------------------
abc
1 row selected.
gSQL> SELECT UNHEX_TO_CHARSTR( HEX( 'abc' ) ) FROM DUAL;
UNHEX_TO_CHARSTR( HEX( 'abc' ) )
--------------------------------
abc
1 row selected.
```

17.222 USER_ID

Syntax

```
USER_ID ()
```

Description

It obtains the current user's number ID.

In cluster system, the value may vary depending on the connected server.
It is recommended to use **CURRENT_USER** function obtaining the current username.

Example

```
% gsql test test
gSQL> SELECT USER_ID() FROM dual;
USER_ID()
---------
        6
1 row selected.
```

17.223 UUID

Syntax

```
UUID()
```

Description

It creates the universal unique identifier, then returns it.
The return type is VARBINARY type, and it internally consists of 16 bytes.

Example

```
gSQL> SELECT HEX( UUID() ) FROM DUAL;
HEX( UUID() )
--------------------------------
E6F0A5C2387511E8B95259E479C2FD50
1 row selected.
```

17.224 VAR_POP

Syntax

```
VAR_POP( expr )
```

Description

It is an aggregation function, and it obtains the population variance of an expr set. If the number of expr sets except for NULL is one, then it returns 0.

The following table describes the arguments and result types.

Table 17-30 Argument and result type of VAR_POP

expr	Result type
NATIVE_INTEGER family • NATIVE_SMALLINT • NATIVE_INTEGER • NATIVE_BIGINT	NATIVE_DOUBLE
NUMBER	NUMBER
NATIVE_DOUBLE family • NATIVE_REAL • NATIVE_DOUBLE	NATIVE_DOUBLE

The population variance is a variance of the population (entire) group, and it is the average of t he square value of deviation. In other words, it is calculated by extracting the population averag e (the entire average) from each value of the data, and squaring each value, then adding them t ogether and dividing them by the number of datas in the population group.
This value is used to figure out how far each value is from the average value.

For more information, refer to **STDDEV_POP**.

Example

```
gSQL> SELECT VAR_POP(c1) FROM t1;
VAR_POP(C1)
-----------
     105.76
1 row selected.
```

17.225 VAR_POP() OVER

Syntax

```
VAR_POP ( expr ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function VAR_POP calculates the population variance of expr.
If the number of expr except for NULL is one, then it returns 0 as a result.

Example

```
gSQL> SELECT product_id, min_price
             , VAR_POP( min_price ) OVER ( ORDER BY product_id ) AS "VAR_POP"
        FROM product_information
       WHERE supplier_id = 102050;
PRODUCT_ID MIN_PRICE          VAR_POP
---------- --------- ----------------
      1769      null             null
      1770        73                0
      2378       247             7569
      2382       731 77499.5555555556
      3355      null 77499.5555555556
5 rows selected.
```

17.226 VAR_SAMP

Syntax

```
VAR_SAMP( expr )
```

Description

It is an aggregation function, and it obtains the sample variance of an expr set. If the number of expr sets except for NULL is one, then it returns NULL.

The following table describes the arguments and result types.

Table 17-31 Argument and result type of VAR_SAMP

expr	Result type
NATIVE_INTEGER family • NATIVE_SMALLINT • NATIVE_INTEGER • NATIVE_BIGINT	NATIVE_DOUBLE
NUMBER	NUMBER
NATIVE_DOUBLE family • NATIVE_REAL • NATIVE_DOUBLE	NATIVE_DOUBLE

Unlike the population variance dealing with the population (entire) group, the sample variance deals with the average and deviation of extracted samples. In other words, it is calculated by ext racting the sample average from each value of the data, and squaring each value, then adding t hem together and dividing them by the number of datas in the population group minus 1.
This value is used to figure out the variance of the population group.

For more information, refer to **STDDEV_SAMP**.

Example

```
gSQL> SELECT VAR_SAMP(c1) FROM t1;
VAR_SAMP(C1)
------------
       132.2
1 row selected.
```

17.227 VAR_SAMP() OVER

Syntax

```
VAR_SAMP ( expr ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function VAR_SAMP calculates the sample variance of expr.
If the number of expr except for NULL is one, then it returns NULL as a result.

Example

```
gSQL> SELECT product_id, min_price
             , VAR_SAMP( min_price ) OVER ( ORDER BY product_id ) AS "VAR_SAMP"
        FROM product_information
       WHERE supplier_id = 102050;
PRODUCT_ID MIN_PRICE         VAR_SAMP
---------- --------- ----------------
      1769      null             null
      1770        73             null
      2378       247            15138
      2382       731 116249.333333333
      3355      null 116249.333333333
5 rows selected.
```

17.228 VARIANCE

Syntax

```
VARIANCE( [ ALL ¦ DISTINCT ] expr )
```

Description

It is an aggregation function, and it obtains the variance of an expr set.

If ALL is specified, this function is performed for all values. If DISTINCT is specified, this function is perfor med for the values of which the duplicates were deleted from. If it is not specified, it is processed as if AL L is apecified.

If the number of expr sets except for NULL after deleting the duplicates by using DISTINCT is one, then it r eturns 0.

The following table describes the arguments and result types.

Table 17-32 Argument and result type of VARIANCE

expr	Result type
NATIVE_INTEGER family • NATIVE_SMALLINT • NATIVE_INTEGER • NATIVE_BIGINT	NATIVE_DOUBLE
NUMBER	NUMBER
NATIVE_DOUBLE family • NATIVE_REAL • NATIVE_DOUBLE	NATIVE_DOUBLE

GOLDILOCKS gets the variance as follows.

- If the number of expr sets are 1, then it returns 0.
- If the number of expr sets are bigger than 1, It returns the value of **STDDEV_SAMP (expr)**.

For more information, refer to **STDDEV**.

Example

```
gSQL> SELECT VARIANCE(c1) FROM t1;
VARIANCE(C1)
------------
       132.2
1 row selected.
gSQL> SELECT VARIANCE(ALL c1) FROM t1;
VARIANCE(ALL C1)
----------------
           132.2
1 row selected.
gSQL> SELECT VARIANCE(DISTINCT c1) FROM t1;
VARIANCE(DISTINCT C1)
---------------------
               176.25
1 row selected.
```

17.229 VARIANCE() OVER

Syntax

```
VARIANCE ( expr ) OVER < window name or specification >
```

For more information about < window name or specification >, refer to **window clause**.

Description

Window function VARIANCE calculates the variance of expr.
If the number of expr except for NULL is one, then it returns 0 as a result.

Example

```
gSQL> SELECT product_id, min_price
             , VARIANCE( min_price ) OVER ( ORDER BY product_id ) AS "VARIANCE"
        FROM product_information
       WHERE supplier_id = 102050;
PRODUCT_ID MIN_PRICE         VARIANCE
---------- --------- ----------------
      1769      null             null
      1770        73                0
      2378       247            15138
      2382       731 116249.333333333
      3355      null 116249.333333333
5 rows selected.
```

17.230 VERSION

Syntax

```
VERSION()
```

Description

It obtains the product's version string.

Example

```
gSQL> SELECT VERSION() FROM dual;
VERSION()
-------------------------------------
Release Name.X.X.X revision(XXXXX)
1 row selected.
```

17.231 WIDTH_BUCKET

Syntax

```
WIDTH_BUCKET( num, min, max, cnt )
```

Description

It creates a section of the same width as cnt within a range between specified min and max, and it return s the section location in which the num is located.

The data type of num argument, min argument, max argument and cnt argument can be a numeric data type.

min, max means the range for the section. If the min value is equal to the max value, an error is returned. cnt means the number of sections. The cnt value should be a positive number. If the cnt value is 0 or a ne gative number, an error is returned.
The section's location is numbered from one.

If any of num, min, max, cnt is NULL, the result is also NULL.

Example

```
gSQL> SELECT WIDTH_BUCKET( 5, 1, 20, 5 ) AS RESULT FROM DUAL;
RESULT
------
     2
1 row selected.
```

18.

SQL References (A~B)

18.1 ALTER AUDIT POLICY

Function

It adds an auditing target to an audit policy object, or drops an auditing target from an audit policy objec t.

Syntax

```
<alter audit policy statement> ::=
    ALTER AUDIT POLICY policy_name
    { <add_audit_option> ¦ <drop_audit_option> }
    ;
<add_audit_option> ::=
    ADD { <privilege_audit_clause> ¦  <action_audit_clause> ¦ <privilege_audit_clause>
<action_audit_clause> }
<drop_audit_option> ::=
    DROP { <privilege_audit_clause> ¦  <action_audit_clause> ¦ <privilege_audit_clause>
<action_audit_clause> }
<privilege_audit_clause> ::=
    PRIVILEGES <database_privilege> [, ...]
<action_audit_clause> ::=
    ACTIONS { <object_action_audit> ¦ <system_action_audit> } [, ...]
<object_action_audit> ::=
      ALL ON [schema_name.]object_name
    ¦ <object_action> ON [schema_name.]object_name
<system_action_audit> ::=
      ALL
    ¦ <system_action>
```

Invocation and Access Rules

AUDIT SYSTEM ON DATABASE privilege is required to perform <alter audit policy statement>.

Syntax Rules and Parameters

policy_name

It is the name of an audit policy object to be altered.

〈add_audit_option〉

It adds an auditing target to an audit policy.

〈drop_audit_option〉

It drops an auditing target from an audit policy.

〈privilege_audit_clause〉

For more information, refer to **CREATE AUDIT POLICY**.

〈action_audit_clause〉

For more information, refer to **CREATE AUDIT POLICY**.

Description

It can alter an audit policy which is already activated, and it does not effect the existing session but it effe cts only the newly created session.

When dropping ALL option as follows, not all actions are dropped, but only the corresponding ALL option is dropped.

```
CREATE AUDIT POLICY p1
       ACTIONS ALL ON u1.t1,
               SELECT ON u1.t1;
ALTER AUDIT POLICY p1 DROP
      ACTIONS ALL ON u1.t1;
```

Examples

The following is an example of adding a new audit option to an audit policy.

```
ALTER AUDIT POLICY policy_dml
      ADD ACTIONS SELECT ON u1.t1;
```

The following is an example of dropping an audit option from an audit policy.

```
ALTER AUDIT POLICY policy_dml
      DROP ACTIONS SELECT ON u1.t1;
```

Compatibility

The SQL standard does not have the audit policy.

For More Information

Refer to the followings.

- Managing audit policy object
  - **CREATE AUDIT POLICY**
  - **DROP AUDIT POLICY**
  - **ALTER AUDIT POLICY**
- Activating/ deactivating audit policy
  - **AUDIT POLICY**
  - **NOAUDIT POLICY**
- Enquiring audit trail: **AUDIT_TRAIL**
- Dropping audit trail: **ALTER DATABASE CLEAR AUDIT TRAIL**

18.2 ALTER CLUSTER GROUP name ADD MEMBER

Function

It adds a cluster member to a cluster group.

Syntax

```
<alter cluster group add member statement> ::=
    ALTER CLUSTER GROUP group_name ADD
        <cluster member definition> [, ...]
    ;
<cluster member definition> ::=
    CLUSTER MEMBER member_name <connection attribute> [<member position>]
<connection attribute> ::=
    HOST 'address' PORT port_no
<member position> ::=
    POSITION DEFAULT
  ¦ POSITION MAX
  ¦ POSITION number
```

Invocation and Access Rules

It can be performed in a cluster system.

ADMINISTRATION ON DATABASE privilege is required to perform <alter cluster group add member statement>.

Syntax Rules and Parameters

group_name

It is the cluster group name.

〈cluster member definition〉

It defines a cluster member to be included in a cluster group.
A cluster group may include maximum 32 cluster members.

member_name

It is the name of a cluster member.
The cluster member name should be as same as the member name which was defined when the databas e of that cluster member was created.
There should not be the same cluster group, nor the same cluster member.
The length of the name should be shorter than 128 bytes.

The start-up phase for the cluster member should be GLOBAL OPEN.

〈connection attribute〉

It defines the connection information for the communication between the cluster members.
〈connection attribute〉 should be as same as the HOST and PORT which were defined when the database of that cluster member was created.
The combination of HOST and PORT should be unique in the cluster system.

- HOST 'address' uses the host name or IPv4 address. If the host name is used, then it uses the first IPv 4 address of the system.
- PORT port_no should be in the range between 1024 ~ 49151.

〈member position〉

It assigns the position number of the cluster member.

- POSITION DEFAULT
  - The system automatically assigns the position number.
- POSITION MAX
  - It assigns the new member position number even when an empty position number exists.
  - It assigns the value bigger than the biggest member position.
- POSITION number
  - It assigns the position number corresponding to the number
  - The position number should be unique in the cluster system.
  - The position number should be an empty position number, and it should be same or smaller than the biggest position number.
- If it is omitted, the default value is POSITION DEFAULT.

The member_position information of the cluster member can be retrieved through DBA_CLUSTER view.

```
SELECT member_name, member_id, member_position FROM dba_cluster;
```

If the following position numbers are being used,

- G1N1: 0
- G1N2: 1
- G2N2: 3
- G3N2: 5

The following values are assigned according to each option.

- POSITION DEFAULT
  - It assigns 2 which is an empty value.
- POSITION MAX
  - It assigns 6 which is a value of a new position number.
- POSITION 3
  - It is duplicated, so it is an error.
- POSITION 4
  - It assigns 4 which is a position number.

Description

<alter cluster group add member statement> statement does not rebalance shards in the tables.
The following statement should be performed to rebalance shards on the added cluster member.

- **ALTER DATABASE REBALANCE**
- **ALTER TABLE name REBALANCE**

Examples

The following is an example of adding two cluster members to a cluster group.

```
gSQL>
ALTER CLUSTER GROUP g1 ADD
    CLUSTER MEMBER g1n3 HOST '192.168.0.13' PORT 10130,
    CLUSTER MEMBER g1n4 HOST '192.168.0.14' PORT 10140
;
```

```
Cluster Group altered.
```

The following is an example of designating the empty member position as a cluster member position.

```
ALTER CLUSTER GROUP g2 ADD
    CLUSTER MEMBER g2n1 HOST '192.168.0.21' PORT 10210 POSITION 4
;
```

Compatibility

The SQL standard does not define the concepts of the cluster.

For More Information

Refer to the followings.

- CREATE CLUSTER GROUP
- DROP CLUSTER GROUP
- ALTER DATABASE REBALANCE
- ALTER TABLE name REBALANCE

18.3 ALTER CLUSTER GROUP name OFFLINE MEMBER

Function

It sets a cluster member of the cluster group to offline.

Syntax

```
<alter cluster group offline member statement> ::=
    ALTER CLUSTER GROUP group_name OFFLINE CLUSTER MEMBER member_name
    ;
```

Invocation and Access Rules

It can be performed in a cluster system.

ADMINISTRATION ON DATABASE privilege is required to perform <alter cluster group offline member statement>.

Syntax Rules and Parameters

group_name

It is the cluster group name.

member_name

It is the name of a cluster member.
The cluster member should be included in the cluster group of group_name.
The cluster member should be inactive.

Description

It sets the inactive cluster member to offline.

<alter cluster group offline member statement> statement does not rebalance shards in the tables.

Examples

If trying to set the cluster member which is not inactive to offline, then the following error occurs.

```
gSQL>
ALTER CLUSTER GROUP g1 OFFLINE CLUSTER MEMBER g1n2;
ERR-42000(16417): active member 'G1N2' cannot be offlined
```

The following is an example of setting a specific cluster member to offline.

```
gSQL>
ALTER CLUSTER GROUP g1 OFFLINE
    CLUSTER MEMBER g1n3
;
Cluster Group altered.
```

Compatibility

The SQL standard does not define the concepts of the cluster.

For More Information

Refer to the followings.

- CREATE CLUSTER GROUP
- DROP CLUSTER GROUP
- ALTER DATABASE REBALANCE
- ALTER TABLE name REBALANCE

18.4 ALTER CLUSTER LOCATION

Function

It alters a cluster location information.

Syntax

```
<alter cluster location statement> ::=
    ALTER CLUSTER LOCATION member_name
    <cluster connection attribute>
    ;
<cluster connection attribute> ::
       HOST 'address' PORT port_no
```

Invocation and Access Rules

It can be performed in a cluster system.

ADMINISTRATION ON DATABASE privilege is required to perform <alter cluster location statement>.

Syntax Rules and Parameters

member_name

It is the name of a cluster member.
The same cluster member name should exist in the registered cluster location information.
The length of the name should be shorter than 128 bytes.

<cluster connection attribute>

It defines the connection information for the communication between the cluster members.
The combination of HOST and PORT should be unique in the cluster system.

- HOST 'address' uses the host name or IPv4 address. If the host name is used, then it uses the first IPv 4 address of the system.
- PORT port_no should be in the range between 1024 ~ 49151.

Description

If the connection information of the cluster location is altered, the cluster member does not need to be dr opped or recreated, but the connection information can be altered by using **ALTER CLUSTER LOCATION**.

Examples

```
gSQL>
ALTER CLUSTER LOCATION g1n2
    HOST '192.168.0.12' PORT 10120
;
Location altered.
```

Compatibility

The SQL standard does not define the concepts of the cluster.

For More Information

Refer to the followings.

- **CREATE CLUSTER LOCATION**
- **DROP CLUSTER LOCATION**

18.5 ALTER DATABASE ADD LOGFILE

Function

It adds log file groups or log file members to the database.

Syntax

```
<alter database add logfile statement> ::=
      <add logfile member statement>
    ¦ <add logfile group statement>
    ;
<add logfile member statement> ::=
    ALTER DATABASE ADD LOGFILE MEMBER <add logfile clause> [, ...] TO
        <group clause>
<add logfile group statement> ::=
    ALTER DATABASE ADD LOGFILE <group clause> ( 'logfile_name' [, ...] )
        <size clause> [ REUSE ]
<group clause> ::=
    GROUP integer
<add logfile clause> ::=
    'logfile_name' [ REUSE ]
<size clause> ::=
    integer [ M ¦ G ]
```

Invocation and Access Rules

ALTER DATABASE ON DATABASE privilege is required to perform <alter database add logfile statement>.

Syntax Rules and Parameters

〈alter database add logfile statement〉

The database should be in MOUNT phase.

〈add logfile member statement〉

The log member is added to an existing log file group.

- 〈add logfile clause〉
  - 'logfile_name' is the file name of logfile member to be added to the log file group.
  - If the file does not exist, a new file is created.
  - The length of the logfile_name should be shorter than 1024 bytes.
- 〈group clause〉
  - It specifies the identifier of the logfile group to be added to the database.
  - Integer should be an identifier of the existing logfile group.
  - If an identifier for the integer does not exist, an error occurs.

〈add logfile group statement〉

It adds a new log file group.

- It is added as the next group of the CURRENT log file group.
- 〈group clause〉
  - It specifies the identifier of the logfile group to be added to the database.
  - Integer should be an identifier of the unexisting logfile group.
  - If an identifier for the integer exists, an error occurs.
- 〈size clause〉
  - The file size can be specified minimum of 20 MB to maximum of 120 GB.
  - The file size should be bigger than the sum of redo log buffer size and pending log buffer size.
- When 'logfile_name' already exists, and the REUSE option is used, if the file size is as same as anothe r group member, then the existing log file is reused.

Description

It is recommended to back up the control file just in case for the file damage because the newly added lo g file groups and log members are stored in the control file.

Examples

The following is an example of adding two log file members to an existing log file group 3.

```
ALTER DATABASE ADD LOGFILE MEMBER 'logfile1.log', 'logfile2.log' TO GROUP 3;
```

The following is an example of adding a new log file group 4 to database. The size of log file group 4 is 1 00 M, and the log file name is 'logfile1.log'.

```
ALTER DATABASE ADD LOGFILE GROUP 4 ( 'logfile1.log' ) SIZE 100M;
```

When adding a log group, a single log file should be used, and multiple log files can be added t o an existing group as a member.

Compatibility

The SQL standard does not define ALTER DATABASE statement.

For More Information

Refer to the followings.

- ALTER DATABASE ADD LOGFILE
- ALTER DATABASE DROP LOGFILE
- ALTER DATABASE RENAME LOGFILE

18.6 ALTER DATABASE ARCHIVELOG

Function

It alters an archive setting of the online log file in the database.

Syntax

```
<alter database archivelog statement> ::=
    ALTER DATABASE { ARCHIVELOG ¦ NOARCHIVELOG }
    ;
```

Invocation and Access Rules

ALTER DATABASE ON DATABASE privilege is required to perform <alter database archivelog statement>.

Syntax Rules and Parameters

<alter database archivelog statement>

- The database should be in MOUNT phase.
- ARCHIVELOG
  - It archives the online log file.
- NOARCHIVELOG
  - It does not archive the online log file.

Description

For the database backup and the media recovery using the backup, the system should be operated in AR CHIVELOG mode.

Example

The following is an example of how to set up a database to archive mode.

```
ALTER DATABASE ARCHIVELOG;
```

Compatibility

The SQL standard does not define ALTER DATABASE statement.

For More Information

Refer to the followings.

- **ALTER DATABASE BACKUP**
- **ALTER TABLESPACE name BACKUP**

18.7 ALTER DATABASE BACKUP

Function

The backup state is set to ACTIVE or INACTIVE to perform a full backup of the database. Then, the increm ental database backup and control file backup are performed.

Syntax

```
<alter database backup statement> ::=
      <database begin backup statement>
    ¦ <database end backup statement>
    ¦ <database incremental backup statement>
    ¦ <database controlfile backup statement>
    ;
<database begin backup statement> ::=
    ALTER DATABASE BEGIN BACKUP [ AT <domain name> ]
    ;
<database end backup statement> ::=
    ALTER DATABASE END BACKUP [ AT <domain name> ]
    ;
<database incremental backup statement> ::=
    ALTER DATABASE BACKUP INCREMENTAL
        <incremental backup option> [ AT <domain name> ]    ;
<incremental backup option> ::=
      LEVEL integer [ CUMULATIVE ¦ DIFFERENTIAL ]
<database controlfile backup statement> ::=
    ALTER DATABASE BACKUP CONTROLFILE TO 'target_name'
        [ AT <domain name> ]    ;
```

Invocation and Access Rules

ALTER DATABASE ON DATABASE privilege is required to perform <alter database backup statement>.

Syntax Rules and Parameters

〈database begin backup clause〉

The database is set to the state which the full backup is available.

- All tablespaces in ONLINE state, which is created and used in the database, are set to the state of whi ch the full backup is available.
- The database should be OPEN state and operated in ARCHIVELOG mode.
- After starting BEGIN BACKUP, the following operations which require writing to the data file can not be performed.
  - SHUTDOWN NORMAL
  - OFFLINE / DROP TABLESPACE
  - ADD / DROP DATAFILE
- It may require media recovery on restart when a full backup is ACTIVE state and the instance is abnor mally terminated.

〈database end backup clause〉

The database is set to the state which the full backup is not available.

- All tablespaces in ONLINE state, which is created and used in the database, are set to the state which the full backup is not available.
- The database should be in OPEN phase and operated in ARCHIVELOG mode.

〈database incremental backup statement〉

- An incremental backup is performed for the database.
- The database should be in OPEN phase and operated in ARCHIVELOG mode.

〈incremental backup option〉

- 'integer' can be specified from 0 to 4.
- LEVEL 0 can not specify CUMULATIVE or DIFFERENTIAL.
- CUMULATIVE | DIFFERENTIAL
  - CUMULATIVE
    - If 'integer' is n, it backs up all pages which are altered after the most recent backups of LEVE L 0 ~ LEVEL n-1.
  - DIFFERENTIAL
    - If 'integer' is n, it backs up all pages which are altered after the most recent backups of LEVE L 0 ~ LEVEL n.

- If it is omitted, DIFFERENTIAL is specified by default.

〈database controlfile backup statement〉

- The control file is backed up.
  - The length of 'target_name' should be shorter than 1024 bytes.
  - If 'target_name' already exists, the operation fails.
- The database should be in OPEN phase and operated in ARCHIVELOG mode.

The maximum length of the 'target_name' managed by GOLDILOCKS is 1024 bytes. However, the maximum lengths of the file name varies depending on the OS, so the actual length of 'target_name' which is available to be created can be shorter than 1024 bytes.

〈domain name〉

It is a name of a member or a group for which the statement is performed.
If it is omitted, it is performed for all groups.

Description

It backs up data files and control files in the database. A full backup of the database begins with BEGIN BACKUP, and copies the datafiles using OS file copy, then ends with END BACKUP. The incremental backup file is created in the path set by the BACKUP_DIR 1 property using a single statement.

Examples

The following is an example to set the entire backup state to ACTIVE.

```
ALTER SYSTEM BEGIN BACkUP;
```

The following is an example to set the entire backup state to INACTIVE.

```
ALTER SYSTEM END BACkUP;
```

The following is an example to create the incremental backup of LEVEL 1 by using DIFFERENTIAL.

```
ALTER DATABASE BACKUP INCREMENTAL LEVEL 1 DIFFERENTIAL;
```

The following is an example to create the 'controlfile.bak' backup file for the control file. If the absolute p ath is not included, a backup file is created in the path set by the LOG_DIR property.

```
ALTER DATABASE BACKUP CONTROLFILE TO 'controlfile.bak';
```

Compatibility

The SQL standard does not define ALTER DATABASE statement.

For More Information

Refer to the followings.

- **ALTER TABLESPACE name BACKUP**
- **ALTER DATABASE RECOVER**

18.8 ALTER DATABASE CLEAR AUDIT TRAIL

Function

It purges audit records which are accumulated when applying an audit policy.

Syntax

```
<clear audit trail statement> ::=
    ALTER DATABASE CLEAR AUDIT TRAIL
;
```

Invocation and Access Rules

AUDIT SYSTEM ON DATABASE privilege is required to perform <clear audit trail statement>.

Description

If an audit policy is activated, an audit trails is getting longer as time goes by.
Tables configuring an audit trail are stored in MEM_AUX_TBS tablespace, and a user should be cautious not to let the audit trail keep increasing.

Storing Audit Trail

A user should purge an audit trail after storing it according to the following procedure to store an audit tr ail when it is necessary.

- When performing it for the first time

```
CREATE TABLE backup_audit_trail AS SELECT * FROM AUDIT_TRAIL;
COMMIT;
```

- When repeatedly performing it

```
INSERT INTO backup_audit_trail SELECT * FROM AUDIT_TRAIL;
COMMIT;
```

- When purging an audit trail

```
ALTER DATABASE CLEAR AUDIT TRAIL;
```

Examples

Purge an audit trail by using the following statement.

```
ALTER DATABASE CLEAR AUDIT TRAIL;
```

Compatibility

The SQL standard does not have the audit policy.

For More Information

Refer to the followings.

- Managing audit policy object
  - **CREATE AUDIT POLICY**
  - **DROP AUDIT POLICY**
  - **ALTER AUDIT POLICY**
- Activating/ deactivating audit policy
  - **AUDIT POLICY**
  - **NOAUDIT POLICY**
- Enquiring audit trail: **AUDIT_TRAIL**
- Dropping audit trail: **ALTER DATABASE CLEAR AUDIT TRAIL**

18.9 ALTER DATABASE CLEAR PASSWORD HISTORY

Function

It deletes the user's password change history which is accumulated due by applying the profile.

Syntax

```
<clear password history statement> ::=
    ALTER DATABASE CLEAR PASSWORD HISTORY
    ;
```

Invocation and Access Rules

ALTER DATABASE ON DATABASE privilege is required to perform <clear password history statement>.

Description

When a profile is applied to a user, the user's password change history is accumulated according to the PASSWORD_REUSE_MAX, PASSWORD_REUSE_TIME policy.

Table 18-1 Managing the change history

PASSWORD_REUSE_MAX	PASSWORD_REUSE_TIME	Managing the change history
value	value	It manages only the change history within the value range, and the change history out of the value range is automatically deleted.
value	UNLIMITED	It accumulates all change history and it does not delete any change history because all change history should be checked.
UNLIMITED	value	It accumulates all change history and it does not delete any change history because all change history should be checked.
UNLIMITED	UNLIMITED	It does not manage the change history because the change history is not checked.

〈Clear password history statement〉 deletes the accumulated user's password change history.

Examples

The following is an example of executing 〈clear password history statement〉 statement.

```
gSQL> ALTER DATABASE CLEAR PASSWORD HISTORY;
Database altered.
gSQL> COMMIT;
Commit complete.
```

Compatibility

The SQL standard does not define ALTER DATABASE statement.

For More Information

Refer to the followings.

- CREATE PROFILE
- CREATE USER

18.10 ALTER DATABASE DATAFILE AUTOEXTEND

Function

It alters the property to automatically extend disk tablespace data file. If the property is ON, then the size to be extended and the maximum size of the data file also can be altered.

Syntax

```
<alter database datafile autoextend statement> ::=
    ALTER DATABASE DATAFILE datafile_name <autoextend clause>
        [ AT <domain name> ]
    ;
<autoextend clause>
    AUTOEXTEND { ON [ <next size clause> ] [ <max size clause> ] ¦ OFF }
<next size clause>
    NEXT <size clause>
<max size clause>
    MAXSIZE { <size clause> ¦ UNLIMITED }
```

Invocation and Access Rules

ALTER DATABASE ON DATABASE privilege is required to perform <alter database datafile autoextend sta tement>

The datafile automatic expand property can alter the property of disk tablespace only.

datafile_name

It specifies the name of the data file to be altered.

<autoextend clause>

It sets the automatic expand property to ON or OFF. If it is set to ON, then it can specify the automatic ex panded size and the maximum size of the data file.

〈next size clause〉

It specifies the size to be extended when the data file in use does not have available space.

〈max size clause〉

It specifies the maximum expanded size of the data file.

Description

Refer to the syntax rules of each statement.

Examples

The following is an example of altering the automatic expand property of the datafile, the automatic exp anded size, and datafile size.

```
gSQL> ALTER DATABASE DATAFILE 'DISK_TBS.dbf' AUTOEXTEND OFF;
Database altered.
gSQL> ALTER DATABASE DATAFILE 'DISK_TBS.dbf' AUTOEXTEND ON;
Database altered.
gSQL> ALTER DATABASE DATAFILE 'DISK_TBS.dbf' AUTOEXTEND ON NEXT 20M;
Database altered.
gSQL> ALTER DATABASE DATAFILE 'DISK_TBS.dbf' AUTOEXTEND ON MAXSIZE 1G;
Database altered.
gSQL> ALTER DATABASE DATAFILE 'DISK_TBS.dbf' AUTOEXTEND ON 20M MAXSIZE 1G;
Database altered.
```

Compatibility

The SQL standard does not define the concepts of the datafile.

For More Information

Refer to **CREATE DISK DATA TABLESPACE**.

18.11 ALTER DATABASE DELETE BACKUP

Function

It deletes the backup file and the backup information of incremental backup. It can delete all incremental backup of the database or no longer usable obsolete backup.

Syntax

```
<alter database delete backup statement> ::=
    ALTER DATABASE DELETE <delete backup list option>
        BACKUP LIST [ <including backup file option> ]
    ;
<delete backup list option> ::=
      OBSOLETE
    | ALL
<including backup file option> ::=
    INCLUDING BACKUP FILES
```

Invocation and Access Rules

ALTER DATABASE ON DATABASE privilege is required to perform <alter database delete backup statement>.

Syntax Rules and Parameters

<alter database delete backup statement>

The database should be in MOUNT or OPEN phase.

〈delete backup list option〉

It selects the backups to be deleted among the existing incremental backups.

- OBSOLETE: It selects backups of database or tablespaces to be deleted, which was backed up before the most recent database LEVEL 0 backup.
- ALL: It selects all incremental backups to be deleted.

〈including backup file option〉

- If it is omitted, it deletes only the backup information from the control file.
- It also deletes not only backup information but also the backup files.

Description

Deletion of the OBSOLETE incremental backup deletes the incremental backup of which is before the most recent LEVEL 0 database backup. When non-LEVEL 0 incremental backup is performed, it is not deleted even if it includes the previously performed incremental backup. It is because it can be used when performing the incomplete recovery by using the incremental backups.

Be cautious of deleting the backup file together when an incremental backup is deleted. It cannot be recovered even by using the control file which has incremental backup information.

Example

The following is an example to delete the backup information and backup files of all existing incremental backups.

```
ALTER DATABASE DELETE ALL BACKUP LIST INCLUING BACKUP FILES;
```

Compatibility

The SQL standard does not define ALTER DATABASE statement.

For More Information

Refer to the followings.

- **ALTER TABLESPACE name BACKUP**
- **ALTER DATABASE RECOVER**

18.12 ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS

Function

It drops the entire inactive cluster member.

Syntax

```
<alter database drop inactive members statement> ::=
    ALTER DATABASE DROP [ FORCE ¦ NO FORCE ] INACTIVE CLUSTER MEMBERS
    ;
```

Invocation and Access Rules

It can be performed in a cluster system.

ADMINISTRATION ON DATABASE privilege is required to perform <alter database drop inactive cluster members statement>.

Syntax Rules and Parameters

[FORCE | NO FORCE]

- FORCE
  - It drops an inactive cluster member even when there is a possibility of data loss.
- NO FORCE
  - It can not drop an inactive cluster member if there is a possibility of data loss.
- The default value is NO FORCE.

Description

It drops the entire inactive cluster member.

The inactive state of a cluster member means that it is not connected to the cluster system, and it occurs i n the following cases.

- An error occurs on a cluster member in an operating cluster system.
- Trying to start-up the cluster system without driving the cluster member.

However, if the table shard is lost while dropping the cluster member, then an inactive cluster member ca n not be dropped.

Also, if there is a possibility of data loss when dropping the cluster member, then an inactive cluster mem ber can not be dropped. The data is not lost when it is guaranteed that the data in replica of the table or the shard which belongs the inactive cluster member to be dropped is not latest comparing to that in the members of that cluster group. Therefore, an inactive cluster member can be dropped when at least one online member exists in the same cluster group in case for the sharded table, and in the entire cluster in c ase for the cloned table.

However, if an online cluster member does not exist in the cluster group and the service is not available d ue to an inactive cluster member, then the inactive cluster member can be dropped by using FORCE optio n despite of the possibility of the data loss.

It is recommended to use <alter database drop inactive members statement> when an inactive cluster me mber can not be included in the cluster system any more.

Examples

The following is an example of executing <alter database drop inactive members statement>.

```
gSQL> ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS;
Database altered.
```

Compatibility

The SQL standard does not define the concepts of the cluster.

For More Information

Refer to **ALTER SYSTEM JOIN DATABASE**.

18.13 ALTER DATABASE DROP LOGFILE

Function

It drops a log file group or a member which exists in the database.

Syntax

```
<alter database drop logfile statement> ::=
      <drop logfile group statement>
    | <drop logfile member statement>
    ;
<drop logfile group statement> ::=
    ALTER DATABASE DROP LOGFILE <group clause>
<group clause> ::=
    GROUP integer
<drop logfile member statement> ::=
    ALTER DATABASE DROP LOGFILE MEMBER <logfile_list>
<logfile_list> ::=
      'logfile_name'
    | <logfile_list> , 'logfile_name'
```

Invocation and Access Rules

ALTER DATABASE ON DATABASE privilege is required to perform <alter database drop logfile statement>.

Syntax Rules and Parameters

<alter database drop logfile statement>

The database should be in MOUNT phase.
An error occurs when the log file to be deleted is in CURRENT or ACTIVE stage.
At least four log file groups should be remained after dropping.

〈drop logfile group statement〉

It drops the existing log file group.

- 〈group clause〉
  - It specifies the log file group to be dropped.
  - An integer should be an identifier of the existing log file.
  - An error occurs if the integer does not exist.

〈drop logfile member statement〉

It drops the existing log file members.

- 〈logfile_list〉
  - It is the list of the log file members to be dropped.
  - 'logfile_name' should be an existing name.
  - An error occurs if 'logfile_name' does not exist.

Description

For more information, refer to the rules for each syntax.

Examples

The following is an example of dropping the existing log file GROUP 3.

```
ALTER DATABASE DROP LOGFILE GROUP 3;
```

The following is an example of dropping logfile1.log and logfile2.log from the existing logfile GROUP 3.

```
ALTER DATABASE DROP LOGFILE MEMBER 'logfile1.log', 'logfile2.log';
```

Compatibility

The SQL standard does not define ALTER DATABASE statement.

For More Information

Refer to the respective syntax rules, and the followings.

- ALTER DATABASE ADD LOGFILE
- ALTER DATABASE RENAME LOGFILE

18.14 ALTER DATABASE DROP OFFLINE SEGMENTS

Function

It drops segments of offline shards for all tables.

Syntax

```
〈alter database drop offline segments statement〉 ::=
    ALTER DATABASE DROP OFFLINE SEGMENTS
    ;
```

Invocation and Access Rules

It can be performed in the cluster system.

ALTER DATABASE ON DATABASE privilege is required to perform 〈alter database drop offline segments statement〉.

Description

It drops segments of offline shards for all tables and it can be performed even when an inactive cluster member exists.

The inactive state of the cluster member means that it is not connected to the cluster system, and it occurs in the following situations.

- When that cluster member fails in the operating cluster system
- When it tries to start-up the cluster system without running that cluster member

〈alter database drop offline segments statement〉 performs **〈alter table drop offline segments statement〉** per each table, and it is equivalent to the sum of following queries.

```
ALTER TABLE t1 DROP OFFLINE SEGMENTS;
COMMIT;
ALTER TABLE t2 DROP OFFLINE SEGMENTS;
COMMIT;
ALTER TABLE t3 DROP OFFLINE SEGMENTS;
COMMIT;
...
ALTER TABLE tn DROP OFFLINE SEGMENTS;
COMMIT;
```

<alter database drop offline segments statement> is not terminated even when an error occurs in a specif ic table, but proceeds in the next table, and succeeds with the following warning.

```
gSQL> ALTER DATABASE DROP OFFLINE SEGMENTS;
ERR-42000(16553): of the total '5' tables, '1' tables failed to drop offline segments
Database altered.
```

The error message above means that one of the five tables failed.

If <alter database drop offline segments statement> is performed again after taking an appropriate action for the error, then it is operated only for the failed table.

For more information about the error, refer to the system trace log (system.trc) of the member which exe cuted the statement.

Example

The following is an example of performing <alter database drop offline segments statement> statement.

```
gSQL> ALTER DATABASE DROP OFFLINE SEGMENTS;
Database altered.
```

Compatibility

The SQL standard does not define the concepts of the cluster.

For More Information

Refer to **ALTER TABLE name DROP OFFLINE SEGMENTS**.

18.15 ALTER DATABASE MOVE SHARD

Function

It rebalances shard of all tables in a specific cluster group to another cluster group.

Syntax

```
<alter database move shard statement> ::=
    ALTER DATABASE MOVE SHARD FROM CLUSTER GROUP src_cluster_group
        TO CLUSTER GROUP dest_cluster_group
       [ ONLINE ¦ OFFLINE ]
       [ <shard divisor> ]
       [ <parallel clause> ]
    ;
<shard divisor> ::=
    SHARD DIVISOR integer
<parallel clause> ::=
    NOPARALLEL
  ¦ PARALLEL [ integer ]
```

Invocation and Access Rules

It can be performed in a cluster system.

ALTER DATABASE ON DATABASE privilege is required to perform <alter database move shard statement>.

Syntax Rules and Parameters

src_cluster_group

It is a cluster group to which the table shard is moved.

dest_cluster_group

It is a target cluster group to which the table shard is moved.

[ONLINE | OFFLINE]

It determines whether to allow DML when rebalancing table shard.

- ONLINE
  - It allows INSERT, UPDATE, and DELETE.
- OFFLINE
  - It does not allow INSERT, UPDATE, DELETE.
- When it is omitted, the default value is ONLINE.

〈shard divisor〉

It specifies the number of shard's partitions.

- It divides the shard as many as the number of partitions, then rebalances them in the remote server.
- The minimum value of an integer is 0 and the maximum value is 1000.
- If it is omitted, then it follows REBALANCE_SHARD_DIVISOR property.
- If the integer is smaller than the parallel integer, then it is revised to the same value as the parallel int eger.

〈parallel clause〉

It specifies the number of threads to use when rebalancing the table.

- NOPARALLEL
  - It does not rebalance tables in parallel.
- PARALLEL [integer]
  - It rebalances tables in parallel.
  - The minimum value of an integer is 0 and the maximum value is 64.
  - If the integer is omitted, then it is 0.
  - If the integer is 0, then the system determines the optimal value.

Description

When adding a cluster member and a cluster group using the following statements, the table shard is not rebalanced.

- CREATE CLUSTER GROUP
- ALTER CLUSTER GROUP name ADD MEMBER

Perform <alter database rebalance statement> to rebalance the shard of the entire table which was not rebalanced when adding a cluster group and a cluster member.

<alter database move shard statement> is performed as the following concepts for tables which did not rebalance the shard.

```
ALTER TABLE t1 MOVE SHARD FROM CLUSTER GROUP src_group TO CLUSTER GROUP dest_group;
COMMIT;
ALTER TABLE t2 MOVE SHARD FROM CLUSTER GROUP src_group TO CLUSTER GROUP dest_group;
COMMIT;
ALTER TABLE t3 MOVE SHARD FROM CLUSTER GROUP src_group TO CLUSTER GROUP dest_group;
COMMIT;
...
...
ALTER TABLE t_n MOVE SHARD FROM CLUSTER GROUP src_group TO CLUSTER GROUP dest_group;
COMMIT;
```

It is performed as above for all tables except for a CLONED table and a CLUSTER WIDE table. The <alter database move shard statement> is proceeded even when the rebalancing the shard of a specific table fails. It does not rollback the table which succeeded in rebalancing the shard.

Therefore, when performing <alter database move shard statement> again after appropriately processed an error, then it rebalances only the shard for the table requiring the rebalancing. In this case, the table which succeeded in rebalancing the shard is not included in a target of the rebalancing.

Examples

The following is an example of performing <alter database move shard statement>.

```
gSQL> ALTER DATABASE MOVE SHARD FROM CLUSTER GROUP G1 TO CLUSTER GROUP G2;
Database altered.
```

Compatibility

The SQL standard does not define the concepts of the cluster.

For More Information

Refer to the followings.

- ALTER TABLE name MOVE SHARD
- CREATE CLUSTER GROUP
- ALTER CLUSTER GROUP name ADD MEMBER

18.16 ALTER DATABASE OFFLINE INACTIVE CLUSTER MEMBERS

Function

It sets the entire inactive cluster member to offline. In other words, it sets the shard map for the cluster member to offline.

Syntax

```
<alter database offline inactive cluster members statement> ::=
    ALTER DATABASE OFFLINE INACTIVE CLUSTER MEMBERS
    ;
```

Invocation and Access Rules

It can be performed in a cluster system.

ADMINISTRATION ON DATABASE privilege is required to perform <alter database offline inactive cluster members statement>.

Syntax Rules and Parameters

It sets the entire inactive cluster member to offline.
The inactive state of a cluster member means that it is not connected to the cluster system, and it occurs in the following cases.

- An error occurs on a cluster member in an operating cluster system.
- Trying to start-up the cluster system without driving the cluster member.

Description

It is recommended to use <alter database offline inactive members statement> when an inactive cluster member can not be included in the cluster system any more.

If an inactive cluster member can participate in a cluster system, then perform **ALTER SYSTEM JOIN DATABASE** to include it in a cluster system.

The cluster member which is set to offline can be shifted to online again by using the following statements after the join.

- **ALTER DATABASE REBALANCE**
- **ALTER TABLE name REBALANCE**

Examples

```
gSQL> ALTER DATABASE OFFLINE INACTIVE CLUSTER MEMBERS;
```

Compatibility

The SQL standard does not define the concepts of the cluster.

For More Information

Refer to the followings.

- **ALTER SYSTEM JOIN DATABASE**
- **ALTER DATABASE REBALANCE**
- **ALTER TABLE name REBALANCE**

18.17 ALTER DATABASE REBALANCE

Function

It rebalances shard of all tables.

Syntax

```
<alter database rebalance statement> ::=
    ALTER DATABASE REBALANCE
       [ ONLINE ¦ OFFLINE ]
       [ <shard divisor> ]
       [ <parallel clause> ]
    ;
<shard divisor> ::=
    SHARD DIVISOR integer
<parallel clause> ::=
    NOPARALLEL
  ¦ PARALLEL [ integer ]
```

Invocation and Access Rules

It can be performed in a cluster system.

ALTER DATABASE ON DATABASE privilege is required to perform <alter database rebalance statement>.

Syntax Rules and Parameters

[ONLINE | OFFLINE]

It determines whether to allow DML when rebalancing table shard.

- ONLINE
  - It allows INSERT, UPDATE, and DELETE.

- OFFLINE
  - It does not allow INSERT, UPDATE, DELETE.
- When it is omitted, the default value is ONLINE.

〈shard divisor〉

It specifies the number of shard's partitions.

- It divides the shard as many as the number of partitions, then rebalances them in the remote server.
- The minimum value of an integer is 0 and the maximum value is 1000.
- If it is omitted, then it follows REBALANCE_SHARD_DIVISOR property.
- If the integer is smaller than the parallel integer, then it is revised to the same value as the parallel int eger.

〈parallel clause〉

It specifies the number of threads to use when rebalancing the table.

- NOPARALLEL
  - It does not rebalance tables in parallel.
- PARALLEL [integer]
  - It rebalances tables in parallel.
  - The minimum value of an integer is 0 and the maximum value is 64.
  - If the integer is omitted, then it is 0.
  - If the integer is 0, then the system determines the optimal value.

Description

When adding a cluster member and a cluster group using the following statements, the table shard is not rebalanced.

- CREATE CLUSTER GROUP
- ALTER CLUSTER GROUP name ADD MEMBER

Perform 〈alter database rebalance statement〉 to rebalance the shard of the entire table which was not re balanced when adding a cluster group and a cluster member.

〈alter database rebalance statement〉 is performed as the following concepts for tables which did not reb alance the shard.

```
ALTER TABLE t1 REBALANCE;
COMMIT;
ALTER TABLE t2 REBALANCE;
COMMIT;
ALTER TABLE t3 REBALANCE;
COMMIT;
...
...
ALTER TABLE t_n REBALANCE;
COMMIT;
```

The <alter database rebalance statement> is proceeded even when the rebalancing the shard of a specific table fails. It does not rollback the table which succeeded in rebalancing the shard.

Therefore, when performing <alter database rebalance statement> again after appropriately processed an error, then it rebalances only the shard for the table requiring the rebalancing. In this case, the table whic h succeeded in rebalancing the shard is not included in a target of the rebalancing.

Examples

The following is an example of performing <alter database rebalance statement>.

```
gSQL> ALTER DATABASE REBALANCE;
Database altered.
```

Compatibility

The SQL standard does not define the concepts of the cluster.

For More Information

Refer to **ALTER TABLE name REBALANCE**.

18.18 ALTER DATABASE REBALANCE EXCLUDE CLUSTER GROUP

Function

It rebalances shard of all tables excluding shards of a specific cluster group.

Syntax

```
<alter database rebalance exclude cluster group statement> ::=
    ALTER DATABASE REBALANCE EXCLUDE CLUSTER GROUP cluster_group_name
       [ ONLINE ¦ OFFLINE ]
       [ <shard divisor> ]
       [ <parallel clause> ]
    ;
<shard divisor> ::=
    SHARD DIVISOR integer
<parallel clause> ::=
    NOPARALLEL
  ¦ PARALLEL [ integer ]
```

Invocation and Access Rules

It can be performed in a cluster system.

ALTER DATABASE ON DATABASE privilege is required to perform <alter database rebalance exclude cluster group statement>.

Syntax Rules and Parameters

cluster_group_name

It is a name of the cluster group excluding a shard of the table.
If the specified cluster group is the only cluster group, then the statement can not be performed.

[ONLINE | OFFLINE]

It determines whether to allow DML when rebalancing table shard.

- ONLINE
  - It allows INSERT, UPDATE, and DELETE.
- OFFLINE
  - It does not allow INSERT, UPDATE, DELETE.
- When it is omitted, the default value is ONLINE.

〈shard divisor〉

It specifies the number of shard's partitions.

- It divides the shard as many as the number of partitions, then rebalances them in the remote server.
- The minimum value of an integer is 0 and the maximum value is 1000.
- If it is omitted, then it follows REBALANCE_SHARD_DIVISOR property.
- If the integer is smaller than the parallel integer, then it is revised to the same value as the parallel int eger.

〈parallel clause〉

It specifies the number of threads to use when rebalancing the table.

- NOPARALLEL
  - It does not rebalance tables in parallel.
- PARALLEL [integer]
  - It rebalances tables in parallel.
  - The minimum value of an integer is 0 and the maximum value is 64.
  - If the integer is omitted, then it is 0.
  - If the integer is 0, then the system determines the optimal value.

Description

To drop a cluster group by using **DROP CLUSTER GROUP**, there should not be a shard in the cluster grou p.

Perform <alter database rebalance exclude cluster group statement> to exclude a shard from the cluster g roup. <alter database rebalance exclude cluster group statement> is performed as the following concepts for tables which include a shard in the cluster group.

```
ALTER TABLE t1 REBALANCE EXCLUDE CLUSTER GROUP g3;
COMMIT;
ALTER TABLE t2 REBALANCE EXCLUDE CLUSTER GROUP g3;
COMMIT;
ALTER TABLE t3 REBALANCE EXCLUDE CLUSTER GROUP g3;
COMMIT;
...
...
ALTER TABLE t_n REBALANCE EXCLUDE CLUSTER GROUP g3;
COMMIT;
```

If the <alter database rebalance exclude cluster group statement> fails due to the lack of storage space, it does not rollback the table which succeed in excluding a shard.

Therefore, when performing <alter database rebalance exclude cluster group statement> again after appr opriately processed an error, then it excludes and rebalances only the shard for the table requiring the reb alancing. In this case, the table which succeeded in excluding the shard is not included in a target of the r ebalancing.

Examples

The following is an example of performing <alter database rebalance exclude cluster group statement>.

```
gSQL> ALTER DATABASE REBALANCE EXCLUDE CLUSTER GROUP g3;
Database altered.
```

Compatibility

The SQL standard does not define the concepts of the cluster.

For More Information

Refer to the followings.

- DROP CLUSTER GROUP
- ALTER TABLE name REBALANCE EXCLUDE CLUSTER GROUP cluster_group_list

18.19 ALTER DATABASE RECOVER

Function

It recovers the entire data file or part of the data files in the database by using the online and archive log files.

Syntax

```
<alter database recover statement> ::=
      <complete database recover statement>
    ¦ <datafile recover statement>
    ¦ <complete tablespace recover statement>
    ¦ <incomplete database recover statement>
    ;
<complete database recover statement> ::=
    ALTER DATABASE RECOVER
<datafile recover statement> ::=
    ALTER DATABASE RECOVER DATAFILE <datafile recovery clause>
<datafile recovery clause> ::=
    <datafile recovery object> [, ...]
<datafile recovery object> ::=
    'datafile_name' [<recovery using backup option>] [recovery corruption option>]
<recovery using backup option> ::=
    USING BACKUP 'backup_datafile_name'
<recovery corruption option> ::=
    CORRUPTION
<complete tablespace recover statement> ::=
    ALTER DATABASE RECOVER TABLESPACE tablespace_name
<incomplete database recover statement> ::=
      <batch incomplete recovery statement>
    ¦ <interactive incomplete recovery statement>
    ;
<batch incomplete recovery statement> ::=
    ALTER DATABASE RECOVER <until clause> [<using backup controlfile option>]

<until clause> ::=
```

```
      UNTIL CHANGE integer
<using backup controlfile option> ::=
    USING BACKUP CONTROLFILE
<interactive incomplete recovery statement> ::=
    ALTER DATABASE <incomplete recovery option> [<using backup controlfile option>]
<incomplete recovery option> ::=
      BEGIN INCOMPLETE RECOVERY
    ¦ END INCOMPLETE RECOVERY
    ¦ RECOVER 'logfile name'
    ¦ RECOVER AUTOMATICALLY
    ¦ RECOVER SUGGESTION
    ;
```

Invocation and Access Rules

ALTER DATABASE ON DATABASE privilege is required to perform <alter database recover statement>.

Syntax Rules and Parameters

<complete database recover statement>

The data files of the database are recovered up to date by using the online and archive log files.

- The recovery is performed for all tablespaces in the ONLINE state.
- The database should be in MOUNT phase and in ARCHIVELOG mode.
- If the required archived log file does not exist, it fails.

<datafile recover statement>

It recovers the backuped datafile, the datafile of the tablespace which requires the recovery by using the archive logfile due to an error during the backup, or the datafile of the tablespace which was set to offline by an immediate option, to the latest status.

- Datafile can be recovered on MOUNT phase or OPEN phase.
- The datafile of the tablespace which is in OFFLINE state can be recovered on OPEN phase, and datafile which is either in ONLINE/OFFLINE state can be recovered in MOUNT phase.
- If the required archive logfile does not exist, then the recovery fails.
- <datafile recovery clause>
    - It specifies one or more datafile object list which is a target of the recovery.

- 〈datafile recovery object〉
  - It sets the name of datafile which is a target of the recovery, and the recovery option.
- 〈recovery using backup option〉
  - It sets the name of backup datafile of the datafile which is a target of the recovery.
- 〈recovery corruption option〉
  - It determines whether to recover only the pages corrupted from the datafile which is a target of t he recovery.

〈complete tablespace recover statement〉

The data files of the tablespace is recovered up to date.

- Tablespace recovery should be performed when the database is in MOUNT or OPEN phase.
- The recovery in the OPEN phase can only be performed when the tablespaces is in the OFFLINE stage, and the recovery in MOUNT phase can be performed when the tablespace is either in ONLINE/ OFFLI NE stage.
- If the required archive log file does not exist, it fails.
- The following is the case which requires the tablespace recovery operation.
  - The tablespace became OFFLINE by IMMEDIATE.
  - The backed up data file is used.
  - A failure occurred during the entire backup.

〈incomplete database recover statement〉

〈batch incomplete database recover statement〉

The datafiles in the database are recovered in a batch up to a specific point of time by using the online an d archive logfile.

- The recovery is performed for all tablespaces in the ONLINE stage.
- The database should be in MOUNT phase and in ARCHIVELOG mode.
- It fails if using the data file containing data which is after the time of the incomplete recovery.
- The database should be OPEN by using RESETLOGS after completion of incomplete recovery.
- 〈until clause〉
  - A specific point of time for incomplete recovery
  - UNTIL CHANGE: The point of time is specified for incomplete recovery in log units
- 〈using backup controlfile〉
  - Deprecated

〈interactive incomplete database recover statement〉

The data files in the database are interactively recovered with a user up to a specific point of time by usin g online and archive log file.

- The recovery is performed for all tablespaces in the ONLINE stage.
- The database should be in MOUNT phase and in ARCHIVELOG mode.
- It fails if using the data file containing data which is after the time of the incomplete recovery.
- The database should be OPEN by using RESETLOGS after completion of incomplete recovery.
- 〈incomplete recovery option〉
  - It is an option to perform an interactive incomplete recovery in log units.
  - BEGIN INCOMPLETE RECOVERY: It starts an incomplete recovery.
  - END INCOMPLETE RECOVERY: It ends an incomplete recovery.
  - RECOVER 'logfile name': A user directly specifies the log file performing the recovery.
  - RECOVER AUTOMATICALLY: All recoverable archive log files are recovered.
  - RECOVER SUGGESTION: It recovers archive log files which are required for the recovery and reco mmended by the system.
- 〈using backup controlfile〉
  - Deprecated.

Description

Incomplete recovery of the database is not easy to find a recovery completion point at a time. Therefore, t he desired recovery point is found by performing it several times.
However, it becomes a new database if the database is started up with RESETLOGS option after an inco mplete recovery. Therefore, the incomplete recovery should be performed several times after creating a c opy of the archived log files and online redo log files.

Examples

The following is an example of a complete recovery for entire database.

```
ALTER DATABASE RECOVER;
```

The following is an example of a datafile recovery.

```
ALTER DATABASE RECOVER DATAFILE 'test.dbf';
```

The following is an example of a tablespace recovery.

```
ALTER DATABASE RECOVER TABLESPACE test_tbs;
```

The following is an example of incomplete recovery for the entire database until LSN is 11123.

```
ALTER DATABASE RECOVER UNTIL CHANGE 11123;
```

The following is an example of interactive incomplete recovery until the recoverable archive log files.

```
ALTER DATABASE BEGIN INCOMPLETE RECOVERY;
ALTER DATABASE RECOVER AUTOMATICALLY;
ALTER DATABASE END INCOMPLETE RECOVERY;
```

Compatibility

The SQL standard does not define ALTER DATABASE statement.

For More Information

Refer to the followings.

- **ALTER DATABASE BACKUP**
- **ALTER TABLESPACE name BACKUP**
- **ALTER SYSTEM {MOUNT | OPEN} DATABASE**

18.20 ALTER DATABASE REGISTER

Function

It registers unrecoverable segments in the database.

Syntax

```
<alter database register statement> ::=
    ALTER DATABASE REGISTER IRRECOVERALBE SEGMENT
        <segment physical identifier list>
    ;
<segment physical identifier list> ::=
      integer
    | <segment physical identifier list> , integer
```

Invocation and Access Rules

ALTER DATABASE ON DATABASE privilege is required to perform <alter database register statement>.

Syntax Rules and Parameters

<alter database register statement>

It registers the unrecoverable segments in the database. The statement can be used on the assumption th at the segment is not used any more, when the database is not recoverable and the backup does not exis t.

- The database should be in MOUNT phase.
- The registered segment identifier list is initialized at restart.
- If a server restart is successful, the registered segment becomes 'UNUSABLE' state, and that segment s should be deleted.

〈segment physical identifier list〉

The list of unrecoverable segment identifier
- Integer: 8 bytes integer segment identifier

Description

When a server restarts after abnormal termination, the database performs the recovery process. During t
his process, it executes pages again by using the REDO log to recover pages which was not reflected in th
e disk in the previous service stage.

If an unexpected failure occurs during execution of the REDO operation, that statement can be used to ig
nore the failure and to execute the recovery.

Example

The following is an example of giving up the recovery of the segment whose identifier is 402867932364
8.

```
ALTER DATABASE REGISTER IRRECOVERABLE SEGMENT 4028679323648;
```

Compatibility

The SQL standard does not define ALTER DATABASE statement.

For More Information

Refer to the followings.

- ALTER TABLESPACE name BACKUP
- ALTER DATABASE RECOVER

18.21 ALTER DATABASE RENAME GLOBAL TRANSACTION LOGFILE

Function

It renames the global transaction logfile in the database.

Syntax

```
<alter database rename global transaction logfile statement> ::=
    ALTER DATABASE RENAME GLOBAL TRANSACTION LOGFILE <source_clause>
       TO <target_clause>
    ;
<source_clause> ::= <logfile_list>
<target_clause> ::= <logfile_list>
<logfile_list> ::=
      'logfile_name'
    | <logfile_list>, 'logfile_name'
```

Invocation and Access Rules

ALTER DATABASE ON DATABASE privilege is required for performing <alter database rename global transaction logfile statement>.

Syntax Rules and Parameters

<alter database rename global transaction logfile statement>

- The database should be in MOUNT phase.
- source_clause
  - The list of the global transaction log file to be modified in the database.
- target_clause
  - The list of the global transaction log file to be modified in the database.

- An error occurs if the file does not exist.
- The length of the name including the path should be shorter than 1024 bytes.

Description

For more information, refer to the rules for each syntax.

Example

The following is an example of modifying the global transaction logfile.

```
ALTER DATABASE RENAME GLOBAL TRANSACTION LOGFILE
 'org_commit_0.log', 'org_commit_1.log' TO 'new_commit_0.log', 'new_commit_1.log';
```

Compatibility

The SQL standard does not define ALTER DATABASE statement.

For More Information

Refer to **ALTER DATABASE RENAME LOGFILE**.

18.22 ALTER DATABASE RENAME LOGFILE

Function

It renames the logfile in the database.

Syntax

```
<alter database rename logfile statement> ::=
    ALTER DATABASE RENAME LOGFILE <logfile_list> TO <logfile_list>
    ;
<logfile_list> ::=
      'logfile_name'
    | <logfile_list> , 'logfile_name'
```

Invocation and Access Rules

ALTER DATABASE ON DATABASE privilege is required for performing <alter database rename logfile statement>.

Syntax Rules and Parameters

<alter database rename logfile statement>

- The database should be in MOUNT phase.
- FROM <logfile_list>
  - The name list of the logfiles to modify in the database.
- TO <logfile_list>
  - The name list of the logfiles to be modified in the database.
  - <logfile_list> should be an existing file.
  - An error occurs if the file does not exist.

Description

For more information, refer to the rules for each syntax.

Example

The following is an example of modifying the existing 'logfile.log' logfile to 'newlogfile.log'.

```
ALTER DATABASE RENAME LOGFILE 'logfile.log' TO 'newlogfile.log';
```

Compatibility

The SQL standard does not define ALTER DATABASE statement.

For More Information

Refer to the followings.

- ALTER DATABASE ADD LOGFILE
- ALTER DATABASE DROP LOGFILE

18.23 ALTER DATABASE RESET LOCAL CLUSTER MEMBER

Function

It resets the local cluster member except for the tablespace object to the time of creating the database.

Syntax

```
<alter database reset local cluster member statement> ::=
    ALTER DATABASE RESET LOCAL CLUSTER MEMBER
    ;
```

Invocation and Access Rules

It can be performed in a cluster system.

The start-up phase should be LOCAL OPEN.

ADMINISTRATION ON DATABASE privilege is required to perform <alter database reset local cluster member statement>.

Description

It resets the local cluster member except for the tablespace object to the time of creating the database. It drops all objects created by a user except for the tablespace object.

<alter database reset local cluster member statement> statement resets an inactive cluster member, and makes the new cluster member to participate in a cluster system.
An inactive cluster member which is disconnected from the cluster system is processed as follows.

- If it can join in a cluster system again, then use JOIN statement to make it join.
  - **ALTER SYSTEM JOIN DATABASE**
- If it can not join in a cluster system again, then use DROP statement to exclude it.

- ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS

In this case, the device corresponding to the cluster member which is excluded from a cluster system can be used again by using the following two methods.

- Method 1: Recreate the database of the local cluster member.
- Method 2: Reset the local cluster member by using <alter database reset local cluster member statement>.

The method 2 reduces the cost of recreating the tablespace comparing to the method 1.

Examples

The following is an example of a reset by using <alter database reset local cluster member statement> after driving the local cluster member, which is excluded from the cluster system, up to LOCAL OPEN phase.

```
gSQL> \startup nomount
Startup success
gSQL> ALTER SYSTEM MOUNT DATABASE;
System altered.
gSQL> ALTER SYSTEM OPEN LOCAL DATABASE;
System altered.
gSQL> ALTER DATABASE RESET LOCAL CLUSTER MEMBER;
Database altered.
```

Compatibility

The SQL standard does not define the concepts of the cluster.

For More Information

Refer to the followings.

- ALTER SYSTEM JOIN DATABASE
- ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS

18.24 ALTER DATABASE RESTORE

Function

It restores the data files in the database or tablespace by using the incremental backup.

Syntax

```
<alter database restore statement> ::=
      <database restore statement>
    ¦ <tablespace restore statement>
    ¦ <controlfile restore statement>
    ;
<database restore statement> ::=
    ALTER DATABASE RESTORE [ <until clause> ]
<until clause> ::=
    UNTIL CHANGE integer
<tablespace restore statement> ::=
    ALTER DATABASE RESTORE TABLESPACE tablespace_name
<controlfile restore statement> ::=
    ALTER DATABASE RESTORE CONTROLFILE FROM 'file_name'
```

Invocation and Access Rules

ALTER DATABASE ON DATABASE privilege is required to perform <alter database restore statement>.

Syntax Rules and Parameters

<database restore statement>

It restores the data files in the database by using the incremental backup.
The database should be in MOUNT phase.

〈tablespace restore statement〉

It restores the data files in the tablespace by using the incremental backup.

- The database should be in MOUNT or OPEN phase.
- The recovery in OPEN state can be performed only for the tablespaces in OFFLINE state. The recovery in MOUNT phase can be performed for the tablespace is either in ONLINE state or OFFLINE state.

〈controlfile restore statement〉

The control file is recovered using 'file_name'.

- The database should be in NOMOUNT phase.
- The absolute path is recommended for 'file_name' but if relative path is described, then <GOLDILOCKS_HOME>/wal/'file_name' is used.

Description

The data recovery using full backup uses OS copy command to directly copy the backup file to the data file path. The data recovery using incremental restores only the deleted data files or old data files.

Examples

The following is an example of recovering the database by using the incremental backup.

```
ALTER DATABASE RESTORE;
```

The following is an example of recovering the tablespace by using the incremental backup.

```
ALTER DATABASE RESTORE TABLESPACE test_tbs;
```

The following is an example of recovering the database by using only the incremental backup whose LSN is smaller than 11123.

```
ALTER DATABASE RESTORE UNTIL CHANGE 11123;
```

The following is an example of recovering the control file by using controlfile.bak.

```
ALTER DATABASE RESTORE CONTROLFILE FROM 'controlfile.bak'
```

Compatibility

The SQL standard does not define ALTER DATABASE statement.

For More Information

Refer to the followings.

- ALTER DATABASE BACKUP
- ALTER TABLESPACE name BACKUP
- ALTER DATABASE RECOVER

18.25 ALTER DATABASE SYNCHRONIZE

Function

It remotely synchronizes shards and sequences in all tables.

Syntax

```
<alter database synchronize statement> ::=
    ALTER DATABASE SYNCHRONIZE
       [ <synchronize target> ]
       [ ONLINE ¦ OFFLINE ]
       [ <shard divisor> ]
       [ <parallel clause> ]
    ;
<synchronize target> ::=
    TABLE
  ¦ SEQUENCE
  ¦ TABLE AND SEQUENCE
  ¦ SEQUENCE AND TABLE
<shard divisor> ::=
    SHARD DIVISOR integer
<parallel clause> ::=
    NOPARALLEL
  ¦ PARALLEL [ integer ]
```

Invocation and Access Rules

It can be performed in the cluster system.

ALTER DATABASE ON DATABASE privilege is required to perform <alter database synchronize statement>

.

Syntax Rules and Parameters

〈synchronize target〉

It specifies the synchronization target object.

- TABLE
  - It synchronizes the table object.
- SEQUENCE
  - It synchronizes the sequence object.
- TABLE AND SEQUENCE or SEQUENCE AND TABLE
  - It synchronizes the table and the sequence object.
- If it is omitted, the default value is TABLE AND SEQUENCE.

[ONLINE | OFFLINE]

It determines whether to allow DML when performing the synchronization.

- ONLINE
  - It allows INSERT, UPDATE, DELETE.
- OFFLINE
  - It does not allow INSERT, UPDATE, DELETE.
- If it is omitted, the default value is ONLINE.

〈shard divisor〉

It specifies the number of shard's partitions.

- It divides the shard as many as the number of partitions, then synchronizes it with the remote server.
- The minimum value of an integer is 0 and the maximum value is 1000.
- If it is omitted, then it follows REBALANCE_SHARD_DIVISOR property.
- If the integer is smaller than the parallel integer, then it is revised to the same value as the parallel int eger.

If 〈synchronize target〉 specifies only SEQUENCE, then it is ignored.

〈parallel clause〉

It specifies the number of threads to use when synchronizing the table.

- NOPARALLEL

    - It does not synchronize tables in parallel.
- PARALLEL [integer]
    - It synchronize tables in parallel.
    - The minimum value of an integer is 0 and the maximum value is 64.
    - If the integer is omitted, then it is 0.
    - If the integer is 0, then the system determines the optimal value.

If <synchronize target> specifies only SEQUENCE, then it is ignored.

Description

It synchronizes all existing offline shards and sequences, then switch them to online. Unlike **ALTER DATA BASE REBALANCE**, it can be performed even when an inactive cluster member exists.

The inactive state of the cluster member means that it is not connected to the cluster system, and it occur s in the following situations.

- When that cluster member fails in the operating cluster system
- When it tries to start-up the cluster system without running that cluster member

<alter database synchronize statement> performs **<alter table synchronize statement>** per each table, and it is equivalent to the sum of following queries.

```
ALTER TABLE t1 SYNCHRONIZE;
COMMIT;
ALTER TABLE t2 SYNCHRONIZE;
COMMIT;
ALTER TABLE t3 SYNCHRONIZE;
COMMIT;
...
ALTER TABLE tn SYNCHRONIZE;
COMMIT;
```

<alter database synchronize statement> is not terminated even when an error occurs while synchronizing a specific table, but proceeds to synchronize the next table, and succeeds with the following warning.

```
gSQL> ALTER DATABASE SYNCHRONIZE;
ERR-42000(16555): of the total '5' tables, '1' tables failed to synchronize
Database altered.
```

The error message above means that one of the five tables failed.

If <alter database synchronize statement> is performed again after taking an appropriate action for the error, then it is operated only for the failed table.

For more information about the error, refer to the system trace log (system.trc) of the member which executed the statement.

Example

The following is an example of performing <alter database synchronize statement> statement.

```
gSQL> ALTER DATABASE SYNCHRONIZE;
Database altered.
```

Compatibility

The SQL standard does not define the concepts of the cluster.

For More Information

Refer to the followings.

- ALTER DATABASE REBALANCE
- ALTER TABLE name REBALANCE

18.26 ALTER INDEX

Function

It alters the index definition.

Syntax

```
〈alter index statement〉 ::=
      〈alter index physical attribute statement〉
    ¦ 〈rename index statement〉
    ¦ 〈aging index statement〉
    ¦ 〈rebuild index statement〉
    ¦ 〈index coalesce statement〉
    ;
```

Invocation and Access Rules

One of the following privilege is required to perform 〈alter index statement〉.

- The owner of that index
- The owner of the table to which the index belongs
- CONTROL TABLE ON TABLE for the table to which the index belongs
- (ALTER INDEX or CONTROL SCHEMA) ON SCHEMA for the schema to which the index belongs
- ALTER ANY INDEX ON DATABASE

Syntax Rules and Parameters

〈alter index physical attribute statement〉

It alters physical attributes of the index.
For more information, refer to **ALTER INDEX name STORAGE**.

〈rename index statement〉

It alters the index name.
For more information, refer to **ALTER INDEX name RENAME TO**.

〈aging index statement〉

It deletes the empty page of the index.
For more information, refer to **ALTER INDEX name AGING**.

〈rebuild index statement〉

It rebuilds the index.
For more information, refer to **ALTER INDEX name REBUILD**.

〈index coalesce statement〉

It drops the index fragmentation.
For more information, refer to **ALTER INDEX name COALESCE**.

Description

Refer to the descriptions of each detailed statement.

Examples

Refer to the examples of each detailed statement.

Compatibility

The SQL standard does not define the concepts of the index.

18.27 ALTER INDEX name AGING

Function

It deletes an empty page of the index.

Syntax

```
<aging index statement> ::=
    ALTER INDEX index_name AGING
    ;
```

Invocation and Access Rules

One of the following privilege is required to perform <aging index statement>.

- The owner of that index
- The owner of the table to which the index belongs
- CONTROL TABLE ON TABLE for the table to which the index belongs.
- (ALTER INDEX or CONTROL SCHEMA) ON SCHEMA for the schema to which the index belongs
- ALTER ANY INDEX ON DATABASE

Syntax Rules and Parameters

index_name

It is the name of the target index.

Description

This syntax returns pages whose all keys are deleted among index pages to a segment. Aging is processe d in two steps which are logical deletion and physical deletion. A logical deletion is disconnection of inde

x page, and it is performed when SCN of when deleting the last key of a page is smaller than the agable SCN of the system. Then the physical deletion is performed when the SCN of the logical deletion is smaller then the agable SCN of the system.

> ⚠ If the agable SCN of the system does not increase, then the empty page may not be deleted even though the index AGING statement succeeded.

Examples

The following is an example of aging the index.

```
gSQL> select index_name, empty_blocks from user_indexes where index_name = 'T1X';
INDEX_NAME EMPTY_BLOCKS
---------- ------------
T1X                   2
1 row selected.
gSQL> alter index t1x aging;
Index altered.
gSQL> select index_name, empty_blocks from user_indexes where index_name = 'T1X';
INDEX_NAME EMPTY_BLOCKS
---------- ------------
T1X                   0
1 row selected.
```

Compatibility

The SQL standard does not define the concepts of the index.

For More Information

Refer to the followings.

- CREATE INDEX
- ALTER INDEX
- DROP INDEX

18.28 ALTER INDEX name COALESCE

Function

It coalesces adjacent leaf pages of the index so that it decreases the index space in use.

Syntax

```
<index coalesce statement> ::=
    ALTER INDEX index_name COALESCE
    ;
```

Invocation and Access Rules

The user should satisfy the following conditions to perform <index coalesce statement>.

- The owner of that index
- The owner of the table to which the index belongs
- CONTROL TABLE ON TABLE for the table to which the index belongs.
- (ALTER INDEX or CONTROL SCHEMA) ON SCHEMA for the schema to which the index belongs
- ALTER ANY INDEX ON DATABASE

At least one of the following privileges for a tablespace in which the index is to be created is required.

- CREATE OBJECT ON TABLESPACE for that tablespace
- USAGE TABLESPACE ON DATABASE

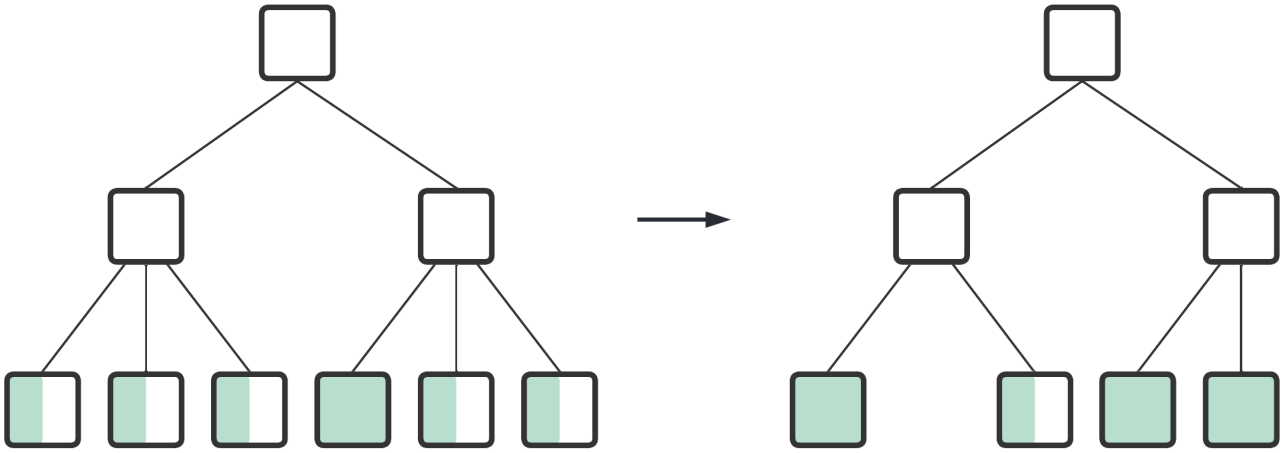
Syntax Rules and Parameters

index_name

It is the name of the target index.
The schema name can be specified, and the user's default schema name is used when it is omitted.

Description

Figure 1 Index coalesce



- It sequentially scans leaf pages and coalesces them when it is allowed to do so, then returns the delet ed pages to the segment.
- It can solve the fragmentation problem of leaf pages which occurred due to UPDATE/ DELETE.
- It drops keys related to invalid shards and frees the constraints about the shard sequence.
- It is operated only when the adjacent leaf pages are allowed to coalesce, so if the fragmentation level is low, then it may not be effective.
- If the fragmentation level of the index is high, then the processing time may take longer than INDEX REBUILD.

Table 18-2 Comparing to INDEX REBUILD

	INDEX REBUILD	INDEX COALESCE
Altering index attributes	Possible	Impossible
Moving tablespace	Possible	Impossible
Locking table	Required	Not required
Additional space for execution	Required	Not required
Decreasing tree height	Possible	Impossible

Examples

```
gsql> ALTER INDEX T1X COALESCE;
Index altered.
```

Compatibility

The SQL standard does not define the concepts of the index.

For More Information

Refer to the followings.

- ALTER INDEX
- ALTER INDEX name REBUILD

18.29 ALTER INDEX name REBUILD

Function

It rebuilds an index.

Syntax

```
<rebuild index statement> ::=
    ALTER INDEX index_name REBUILD
        [ ONLINE ¦ OFFLINE ]
        [ <index attributes> [...] ]
        [ TABLESPACE tablespace_name ]
    ;
<index attributes> ::=
      <physical attribute clause>
    ¦ STORAGE ( <segment attr clause> [...] )
    ¦ <parallel clause>
<physical attribute clause> ::=
      PCTFREE integer
    ¦ INITRANS integer
    ¦ MAXTRANS integer
<segment attr clause> ::=
      INITIAL <size_clause>
    ¦ NEXT <size_clause>
    ¦ MINSIZE <size_clause>
    ¦ MAXSIZE <size_clause>
<size clause> ::=
      integer [ K ¦ M ¦ G ¦ T ]
<parallel clause> ::=
      NOPARALLEL
    ¦ PARALLEL [ integer ]
```

Invocation and Access Rules

The user should satisfy the following conditions to perform <rebuild index statement>.

- The owner of that index
- The owner of the table to which the index belongs
- CONTROL TABLE ON TABLE for the table to which the index belongs.
- (ALTER INDEX or CONTROL SCHEMA) ON SCHEMA for the schema to which the index belongs
- ALTER ANY INDEX ON DATABASE

At least one of the following privileges for a tablespace in which the index is to be created is required.

- CREATE OBJECT ON TABLESPACE for that tablespace
- USAGE TABLESPACE ON DATABASE

Syntax Rules and Parameters

index_name

It is the name of the target index.
The schema name can be specified, and the user's default schema name is used when it is omitted.

[ONLINE | OFFLINE]

It determines whether to allow DML on the table when rebuilding the index.

- ONLINE
  - It allows INSERT, UPDATE, and DELETE.
- OFFLINE
  - It does not allow INSERT, UPDATE, DELETE.
- When it is omitted, the default value is ONLINE.

<physical attribute clause>

It defines the physical attribute information of an index.

- PCTFREE integer
  - Definition
    - It is the reserved space for adjusting the page split frequency caused by the key inserted in th e page.

- It can use the value from 0 to 99.
- If it is omitted, the default value is the value set in the existing index.

- INITRANS integer
  - Definition
    - It specifies the initial number of transactions which can simultaneously access the page.
    - If the number of users who access the index is small, INITRANS is set to low, and if the numb er of users who simultaneously access the index is big, INITRANS is set to high.
    - If necessary, it is automatically increased to the specified MAXTRANS.
  - It can use the value from 1 to 32.
  - If it is omitted, the default value is the value set in the existing index.

- MAXTRANS integer
  - Definition
    - It specifies the maximum number of transactions which can simultaneously access the page.
  - It can use the value from 1 to 32.
  - If it is omitted, the default value is the value set in the existing index.

〈segment attr clause〉

It specifies the information for the index storage space.

- INITIAL integer
  - Definition
    - It specifies the size of physical storage space which is initially allocated when creating the ind ex.
    - This size is aligned to the EXTENT size of the TABLESPACE to which the table belongs. (e.g. If the EXT size is 8192 bytes, 'INITIAL 100' is actually operated as 8192 bytes.)
    - The size (aligned to the EXTENT size of TABLESPACE) should be equal to or bigger than MIN EXTENTS, or it should be equal to or less than MAXEXTENTS.
  - The minimum value is 1, and the maximum value depends on the system environment.
  - If it is omitted, the default value is the value set in the existing index.

- NEXT integer
  - Definition
    - It specifies the physical space size to be allocated when adding the space to the table.
    - This size is aligned to the EXTENT size of the TABLESPACE to which the table belongs. (e.g. If the EXT size is 8192 bytes, 'NEXT 100' is actually operated as 8192 bytes.)
    - NEXT operates as follows, depending on the remaining space size of the index available curr ently. (Obtained by subtracting the amount of currently used space from the MAXEXTENTS si ze)

    - If the remaining space size is 0, then it can not extend the space.
    - If the remaining space size is bigger than 0, but smaller than NEXT, then it allocates the

space as big as the remaining space.
- If the remaining space size is bigger than NEXT, then it allocates the space as big as the NEXT.
  - The minimum value is 1 and the maximum value depends on the system environment.
  - If it is omitted, the default value is the value set in the existing index.

- MINSIZE integer
  - Definition
    - It is the minimum space size of the index.
    - The value should be equal to or smaller MAXSIZE.
  - This size is aligned to the EXTENT size of the TABLESPACE to which the index belongs.
  - The minimum value is 1 and the maximum value depends on the system environment.
  - If it is smaller than the size of two EXTENT, it is specified to the size of two EXTENT.
  - If it is omitted, the default value is the value set in the existing index.

- MAXSIZE integer
  - Definition
    - It is the maximum space size of the index.
    - The value should be equal to or bigger than MINSIZE.
  - This size is aligned to the EXTENT size of the TABLESPACE to which the index belongs.
  - The minimum value is 1 and the maximum value depends on the system environment.
  - If it is omitted, the default value is the value set in the existing index.

〈size clause〉

It specifies the file size in bytes. (If the unit is omitted, the default value is bytes.)

- K: Kilobytes
- M: Megabytes
- G: Gigabytes
- T: Terabytes

NOPARALLEL | PARALLEL [integer]

It specifies the number of threads to be used when rebuilding an index.

- NOPARALLEL
  - It does not rebuild an index in parallel.
- PARALLEL [integer]
  - It rebuilds an index in parallel.
  - If an integer is omitted or set as 0, then it follows INDEX_BUILD_PARALLEL_FACTOR property.
  - The minimum value of an integer is 0 and the maximum value is 64.
  - If the integer or the property value is 0, then the system determines the optimal value.

- If it is omitted, the default value is NOPARALLEL.

TABLESPACE tablespace_name

It specifies the name of the tablespace in which the index is to be rebuilt.

- When it specifies tablespace_name
  - if tablespace_name is data tablespace, then it is rebuilt as a LOGGING index.
  - if tablespace_name is temporary tablespace or nologging tablespace, then it is rebuilt as a NOLOGGING index.
- When TABLESPACE clause is omitted, then it is set to the tablespace of the existing index.

Description

- Dropping the index fragmentation
  - The fragmentation may occur on the index page, when DML is frequently performed in the index. If the tree becomes too big comparing to the valid data, then the index volume becomes larger and the performance is degraded. In this case, rebuilding the index can solve the index fragmentation issue so that the index volume is reduced and the index performance is recovered.
- Altering the tablespace in the index
  - The tablespace in the previously created index can be altered.
  - However, LOGGING should be set properly according to whether the tablespace is TEMPORARY or not.
- Altering LOGGING setting in the index
  - The data tablespace should be set in TABLESPACE option to switch to the LOGGING index.
  - The temporary tablespace or the nologging tablespace should be set in TABLESPACE option to switch to the NOLOGGING index.
- Dropping keys related to invalid shards
  - When shards are changed, the keys related to the previous shards may remain in the index. If they are not dropped but stacked, then *shard sequence exceed* error may occur. This error occurs when shards are frequently changed, and the solution is rebuilding the index.

Examples

The following is an example of altering the index logging setting and the tablespace.

```
gsql> SELECT INDEX_NAME, TABLESPACE_NAME FROM INDEXES AS IDX, TABLESPACES AS TBS WHERE
IDX.TABLESPACE_ID = TBS.TABLESPACE_ID AND IDX.INDEX_NAME = 'T1X';
INDEX_NAME TABLESPACE_NAME
```

```
---------- ---------------
T1X        MEM_TEMP_TBS
1 row selected.
gsql> ALTER INDEX T1X REBUILD TABLESPACE MEM_DATA_TBS;
SELECT INDEX_NAME, TABLESPACE_NAME FROM INDEXES AS IDX, TABLESPACES AS TBS WHERE
IDX.TABLESPACE_ID = TBS.TABLESPACE_ID AND IDX.INDEX_NAME = 'T1X';
INDEX_NAME TABLESPACE_NAME
---------- ---------------
T1X        MEM_DATA_TBS
1 row selected.
```

Compatibility

The SQL standard does not cover the concepts of the index.

For More Information

Refer to the followings.

- CREATE INDEX
- ALTER INDEX
- DROP INDEX

18.30 ALTER INDEX name RENAME TO

Function

It alters the index name.

Syntax

```
<rename index statement> ::=
    ALTER INDEX index_name
        RENAME TO new_index_name
    ;
```

Invocation and Access Rules

One of the following privilege is required to perform <rename index statement>.

- The owner of that index
- The owner of the table to which the index belongs
- CONTROL TABLE ON TABLE for the table to which the index belongs.
- (ALTER INDEX or CONTROL SCHEMA) ON SCHEMA for the schema to which the index belongs
- ALTER ANY INDEX ON DATABASE

Syntax Rules and Parameters

index_name

It is the name of the target index.
The schema name can not be described and it has the same schema name as same as that of the existing index.

new_index_name

It is the name of the new index, and it should be a unique index name within the schema.

Description

Refer to the syntax rules of each statement.

Examples

The following is an example of altering the index name.

```
gSQL> ALTER INDEX t1_idx1 RENAME TO idx_t1_id;
Index altered.
```

Compatibility

The SQL standard does not define the concepts of the index.

For More Information

Refer to the followings.

- CREATE INDEX
- ALTER INDEX
- DROP INDEX

18.31 ALTER INDEX name STORAGE

Function

It alters the physical attributes of the index.

Syntax

```
<alter index physical attribute statement> ::=
    ALTER INDEX index_name
    ¦ <physical attribute clause>
    ¦ [ STORAGE ( <segment attr clause> [...] ) ]
    ;
<physical attribute clause> ::=
      PCTFREE integer
    ¦ INITRANS integer
    ¦ MAXTRANS integer
<segment attr clause> ::=
      INITIAL <size_clause>
    ¦ NEXT <size_clause>
    ¦ MINSIZE <size_clause>
    ¦ MAXSIZE <size_clause>
<size clause> ::=
      integer [ K ¦ M ¦ G ¦ T ]
```

Invocation and Access Rules

One of the following privilege is required to perform <alter index physical attribute statement>.

- The owner of that index
- The owner of the table to which the index belongs
- CONTROL TABLE ON TABLE for the table to which the index belongs.
- (ALTER INDEX or CONTROL SCHEMA) ON SCHEMA for the schema to which the index belongs
- ALTER ANY INDEX ON DATABASE

Syntax Rules and Parameters

index_name

It is the target index name.

〈physical attribute clause〉

It defines the physical attribute information of the index.

- PCTFREE integer
  - Definition
    - The reserved space to adjust the frequency of page splits caused by inserting the key in the p age.
    - It is applied only when the index bottom-up build.
  - It can use of the value from 0 to 99.
  - If it is omitted, the value set in DEFAULT_INDEX_PCTFREE property is used by default.

- INITRANS integer
  - Definition
    - The initial number of transactions simultaneously accessing the page.
    - If the number of users accessing the index is small, then INITRANS is set low. If the number o f users simultaneously accessing the index is big, then INITRANS is set high.
    - If necessary, it is automatically increased to the specified MAXTRANS.
  - It can use the value from 1 to 32.
  - If it is omitted, the default value is 4.

- MAXTRANS integer
  - Definition
    - It specifies the maximum number of transaction simultaneously accessing the page.
  - It can use the value from 1 to 32.
  - If it is omitted, the default value is 8.

〈segment attr clause〉

It specifies the information for the index storage space.

- INITIAL integer
  - Definition
    - It specifies the size of physical storage space which is initially allocated when creating the ind ex.

        - This size is aligned to the EXTENT size of the TABLESPACE to which the table belongs. (e.g. If the EXT size is 8192 bytes, 'INITIAL 100' is actually operated as 8192 bytes.)
        - The size (aligned to the EXTENT size of TABLESPACE) should be equal to or bigger than MIN EXTENTS, or it should be equal to or less than MAXEXTENTS.
        - It is applied only when the index bottom-up build.
    - The minimum value is 1, and the maximum value depends on the system environment.
    - If it is omitted, the default value is one EXTENT size of TABLESPACE to which the table belongs.

- NEXT integer
    - Definition
        - It specifies the physical space size to be allocated when adding the space to the table.
        - This size is aligned to the EXTENT size of the TABLESPACE to which the table belongs. (e.g. If the EXT size is 8192 bytes, 'NEXT 100' is actually operated as 8192 bytes.)
        - NEXT operates as follows depending on the remaining space size of the currently available in dex. (Obtained by subtracting the amount of currently used space from the MAXEXTENTS siz e)

      - If the remaining space size is 0, then it can not extend the space.

      - If the remaining space size is bigger than 0, but smaller than NEXT, then it allocates the space as big as the remaining space.

      - If the remaining space size is bigger than NEXT, then it allocates the space as big as the NEXT.
    - The minimum value is 1 and the maximum value depends on the system environment.
    - If it is omitted, the default value is one EXTENT size of TABLESPACE to which the index belongs.

- MINSIZE integer
    - Definition
        - It is the minimum space size of the index.
        - The value should be equal to or smaller MAXSIZE.
    - This size is aligned to the EXTENT size of the TABLESPACE to which the index belongs.
    - The minimum value is 1 and the maximum value depends on the system environment.
    - If it is smaller than the size of two EXTENT, it is specified to the size of two EXTENT.
    - If it is omitted, the default value is the size of two EXTENT.

- MAXSIZE integer
    - Definition
        - It is the maximum space size of the index.
        - The value should be equal to or bigger than MINSIZE.
    - This size is aligned to the EXTENT size of the TABLESPACE to which the index belongs.
    - The minimum value is 1 and the maximum value depends on the system environment.
    - If it is smaller than the size of two EXTENT, it is specified to the size of two EXTENT.
    - If it is omitted, the default value is 32 terabytes (35,184,372,088,832).
    - Even though the value is set to over 32 terabytes, it is adjusted and set to 32 terabytes.
    - If the newly allocated space is smaller than the already allocated space, then an error occurs.

〈size clause〉

It specifies the file size in byte. (If it is omitted, the default unit is bytes.)

- K: Kilobytes
- M: Megabytes
- G: Gigabytes
- T: Terabytes

Description

Refer to the syntax rules of each statement.

Examples

The following is an example of altering the physical attributes of the index.

```
gSQL> ALTER INDEX idx_t1_id PCTFREE 10 INITRANS 4 MAXTRANS 8;
Index altered.
```

Compatibility

The SQL standard does not define the concepts of the index.

For More Information

Refer to the followings.

- CREATE INDEX
- ALTER INDEX
- DROP INDEX

18.32 ALTER PROFILE

Function

It alters the password management method.

Syntax

```
<alter profile statement> ::=
    ALTER PROFILE profile_name LIMIT
    { <password_parameters>, ...}
    ;
<password parameters> ::=
      FAILED_LOGIN_ATTEMPTS { integer ¦ UNLIMITED ¦ DEFAULT }
    ¦ PASSWORD_LOCK_TIME  { password_parameter_number_interval ¦ UNLIMITED ¦ DEFAULT }
    ¦ PASSWORD_LIFE_TIME  { password_parameter_number_interval ¦ UNLIMITED ¦ DEFAULT }
    ¦ PASSWORD_GRACE_TIME { password_parameter_number_interval ¦ UNLIMITED ¦ DEFAULT }
    ¦ PASSWORD_REUSE_MAX  { integer ¦ UNLIMITED ¦ DEFAULT }
    ¦ PASSWORD_REUSE_TIME { password_parameter_number_interval ¦ UNLIMITED ¦ DEFAULT }
    ¦ PASSWORD_VERIFY_FUNCTION { <verify_policy> ¦ NULL ¦ DEFAULT }
<verify_policy> ::=
      KISA_VERIFY_FUNCTION
    ¦ ORA12C_VERIFY_FUNCTION
    ¦ ORA12C_STRONG_VERIFY_FUNCTION
    ¦ VERIFY_FUNCTION_11G
    ¦ VERIFY_FUNCTION
<password_parameter_number_interval> ::=
   integer
 ¦ integer / integer
```

Invocation and Access Rules

ALTER PROFILE ON DATABASE privilege is required to perform <alter profile statement>.

Syntax Rules and Parameters

profile_name

It is a profile name to be altered.

FAILED_LOGIN_ATTEMPTS

It sets the number of consecutive login attempts allowed to fail.
For more information, refer to **CREATE PROFILE**.

PASSWORD_LOCK_TIME

It sets an account lockout duration (day) after the consecutive login failures.
For more information, refer to **CREATE PROFILE**.

PASSWORD_LIFE_TIME

It sets the password lifetime (day).
For more information, refer to **CREATE PROFILE**.

PASSWORD_GRACE_TIME

It sets a password expiration grace period when log in after PASSWORD_LIFE_TIME.
For more information, refer to **CREATE PROFILE**.

PASSWORD_REUSE_MAX

It specifies the number of the recent passwords which can not be reused when a user wants to reuse the old password.
For more information, refer to **CREATE PROFILE**.

PASSWORD_REUSE_TIME

It specifies the duration which the password can not be reused when a user wants to reuse the old passw ord.
For more information, refer to **CREATE PROFILE**.

PASSWORD_VERIFY_FUNCTION

It set the password complexity verification method.
For more information, refer to **CREATE PROFILE**.

Examples

The following is an example of changing the profile to control the account lockout.

```
gSQL> ALTER PROFILE prof1 LIMIT
        FAILED_LOGIN_ATTEMPTS 3
        PASSWORD_LOCK_TIME 3;
Profile altered.
gSQL> COMMIT;
Commit complete.
```

The following is an example of changing the profile to control the password lifetime.

```
gSQL> ALTER PROFILE prof1 LIMIT
        PASSWORD_LIFE_TIME 90
        PASSWORD_GRACE_TIME 7;
Profile altered.
gSQL> COMMIT;
Commit complete.
```

The following is an example of changing the profile to control the password reusability.

```
gSQL> ALTER PROFILE prof1 LIMIT
        PASSWORD_REUSE_MAX  DEFAULT
        PASSWORD_REUSE_TIME DEFAULT;
Profile altered.
gSQL> COMMIT;
Commit complete.
```

The following is an example of changing the profile to control the password complexity verification.

```
gSQL> ALTER PROFILE prof1 LIMIT
        PASSWORD_VERIFY_FUNCTION KISA_VERIFY_FUNCTION;
Profile altered.
gSQL> COMMIT;
```

```
Commit complete.
```

Compatibility

The SQL standard does not define the concepts of the profile.

For More Information

Refer to **DROP PROFILE**.

18.33 ALTER SEQUENCE

Function

It alters the sequence.

Syntax

```
<alter sequence generator statement> ::=
    ALTER SEQUENCE sequence_name <alter sequence generator options>
    ;
<alter sequence generator options> ::=
    <alter sequence generator option> [, ...]
<alter sequence generator option> ::=
      <alter sequence generator restart option>
    | <basic sequence generator option>
<alter sequence generator restart option> ::=
    RESTART [ WITH integer ]
<basic sequence generator option> ::=
      <sequence generator increment by option>
    | <sequence generator maxvalue option>
    | <sequence generator minvalue option>
    | <sequence generator cycle option>
    | <sequence generator cache option>
<sequence generator increment by option> ::=
    INCREMENT BY integer
<sequence generator maxvalue option> ::=
      MAXVALUE integer
    | (NO MAXVALUE | NOMAXVALUE)
<sequence generator minvalue option> ::=
      MINVALUE integer
    | (NO MINVALUE | NOMINVALUE)
<sequence generator cycle option> ::=
      CYCLE
    | (NO CYCLE | NOCYCLE)
<sequence generator cache option> ::=
      CACHE integer
```

```
    ¦ (NO CACHE ¦ NOCACHE)
```

Invocation and Access Rules

One of the following privilege is required to perform <alter sequence generator statement>.

- The owner of that sequence
- (ALTER SEQUENCE or CONTROL SCHEMA) ON SCHEMA for the schema to which the sequence belo ngs
- ALTER ANY SEQUENCE ON DATABASE

Syntax Rules and Parameters

sequence_name

It is the sequence name to be altered.
It can define schema to which the sequence belongs such as schema_name.sequence_name and if schem a_name is omitted, the default schema name of the user performing the statement is used.

<alter sequence generator restart option>

It sets NEXT VALUE of the sequence.
However, it does not change the value of START WITH which is defined in **CREATE SEQUENCE** statement.

- RESTART
  - If the value is not specified, the value of START WITH defined in <sequence generator definition> is set as the next value of the sequence.
- RESTART WITH integer
  - It sets an integer value as the next value of the sequence.
  - The integer value should be between MINVALUE and MAXVALUE.

If <alter sequence generator restart option> clause is not specified, it changes the sequence attributes bas ed on the current sequence value.

<sequence generator increment by option>

It changes the interval of the sequence number.
The constraints and characteristics are as follows.

- A positive or negative value can be used, but 0 can not be used.
- The absolute value of the interval should be smaller than the difference between MINVALUE and MA XVALUE.
- If it is a positive value, it an ascending sequence. If it is a negative value, it is a descending sequence.

〈sequence generator maxvalue option〉

It changes the maximum value which the sequence can generate.
However, the MAXVALUE should not be smaller than the current sequence value.

- MAXVALUE integer
  - The maximum value is in the range between the minimum (-9,223,372,036,854,775,808) and t he maximum (+9,223,372,036,854,775,807) of 64 bit integer.
  - It should be equal to or bigger than the value of START WITH, and bigger than the value of MINV ALUE.
- NO MAXVALUE | NOMAXVALUE
  - It changes the maximum value as follows.
    - If it is an ascending sequence, it is the maximum value (+9,223,372,036,854,775,807) of the 64 bit integer.
    - If it is a descending sequence, the value is -1.
    - NO MAXVALUE (SQL standard) and NOMAXVALUE are the reserved words with the same m eaning, and either of them can be used.

〈sequence generator minvalue option〉

It changes the minimum value which the sequence can generate.
However, the MINVALUE should not be bigger than the current sequence value.

- MINVALUE integer
  - The minimum value is in the range between the minimum (-9,223,372,036,854,775,808) and th e maximum (+9,223,372,036,854,775,807) of 64bit integer.
  - It should be equal to or smaller than the value of START WITH, and smaller than the value of MA XVALUE.
- NO MINVALUE | NOMINVALUE
  - It changes the minimum value as follows.
    - If it is an ascending sequence, the value is 1.
    - If it is a descending sequence, it is the minimum value (—9,223,372,036,854,775,808) of th e 64bit integer.
  - NO MINVALUE (SQL standard) and NOMINVALUE are the reserved words with the same meanin

g, and either of them can be used.

〈sequence generator cycle option〉

It changes whether to continue generating a value when the sequence value becomes the maximum or minimum value.

- CYCLE
  - If an ascending sequence becomes the maximum value, it generates the value again from the minimum value.
  - If a descending sequence becomes the minimum value, it generates the value again from the maximum value.
- NO CYCLE | NOCYCLE
  - It can not generate the value sequence when it becomes the maximum value or the minimum value.
  - NO CYCLE (SQL standard) and NOCYCLE are the reserved words with the same meaning, and either of them can be used.

〈sequence generator cache option〉

For quick access of a sequence, it defines the number of sequence values to be pre-loaded on the memory.
When restarting the database, the sequence value loaded on the memory is lost, and it starts from the value after loading.

- CACHE integer
  - The CACHE value should be equal to or bigger than 2.
  - If CYCLE exists, the CACHE value should not be bigger than the length of CYCLE.
    - The length of CYCLE: CEIL(MAXVALUE - MINVALUE) / ABS(INCREMENT)
- NO CACHE | NOCACHE
  - It does not pre-load the sequence value in memory.

Description

It can not change START WITH which is one of the sequence attributes defined in **CREATE SEQUENCE** statement. To change START WITH attribute, it should be re-created by performing **CREATE SEQUENCE** statement after performing **DROP SEQUENCE** statement.

Examples

The following is an example of restating the sequence value by using RESTART option, then assigning a n ew ID.

```
gSQL> SELECT id, name FROM t1 ORDER BY 1;
 ID NAME
--- ------
 10 leekmo
 42 mkkim
 51 jhkim
172 ehpark
4 rows selected.
gSQL> ALTER SEQUENCE seq1 RESTART;
Sequence altered.
gSQL> UPDATE t1 SET id = seq1.NEXTVAL;
4 rows updated.
gSQL> SELECT id, name FROM t1 ORDER BY 1;
ID NAME
-- ------
 1 leekmo
 2 mkkim
 3 jhkim
 4 ehpark
4 rows selected.
```

Compatibility

The SQL standard does not define CACHE/ NO CACHE statement.

Table 18-3 SQL standard compatibility

Feature ID	Description	Compatibility
T176	Sequence generator support	O
T177	Sequence generator support: simple restart option	O

For More Information

Refer to the followings.

- CREATE SEQUENCE
- DROP SEQUENCE

18.34 ALTER SESSION CLEANUP GLOBAL TEMPORARY SEGMENT POOL;

Function

It returns all segments which were caught to be reused in a session to tablespaces.

Syntax

```
<alter session cleanup global temporary segment pool statement> ::=
    ALTER SESSION CLEANUP GLOBAL TEMPORARY SEGMENT POOL
    ;
```

Description

It cleans up only the segments of a segment cache in the performed session.

Examples

The following is an example of cleaning up the segment cache of the session.

```
gSQL> ALTER SESSION CLEANUP GLOBAL TEMPORARY SEGMENT POOL;
Session altered.
```

Compatibility

The SQL standard does not define the concepts of the segment cache of a global temporary table and a global temporary index.

For More Information

Refer to **Global Temporary Table**.

18.35 ALTER SESSION SET property_name

Function

It sets the property value of the session.

Syntax

```
<alter session set statement> ::=
    ALTER SESSION SET <property name> { = <property value> | TO DEFAULT }
    ;
```

Syntax Rules and Parameters

<property name>

It is the property name to be set.
For more information, refer to **Server Property** in an administration manual.

<property value>

It is the property value to be set.

TO DEFAULT

It sets the session property value as a system property value.

Description

For more information about property, refer to **Server Property** in an administration manual.

Examples

The following is an example of an error when setting ERROR HINT property so the hint clause includes an error.

```
gSQL> ALTER SESSION SET HINT_ERROR = ON;
Session altered.
gSQL> SELECT /*+ INDEX( t1, invalid_index ) */ name FROM t1 WHERE id = 1;
ERR-42000(16058): not applicable hint :
SELECT /*+ INDEX( t1, invalid_index ) */ name FROM t1 WHERE id = 1
           *
ERROR at line 1:
```

The following is an example of setting the session property value as the system property value.

```
gSQL> ALTER SESSION SET HINT_ERROR TO DEFAULT;
Session altered.
```

Compatibility

The SQL standard does not define the concepts of the session property.

For More Information

Refer to **ALTER SESSION SET property_name**.

18.36 ALTER SYSTEM CHECKPOINT

Function

It performs CHECKPOINT.

Syntax

```
<alter system checkpoint statement> ::=
    ALTER SYSTEM CHECKPOINT
    [ AT <domain name> ]
    ;
```

Invocation and Access Rules

ALTER SYSTEM ON DATABASE privilege is required to perform <alter system checkpoint statement>.

Syntax Rules and Parameters

<alter system checkpoint statement>

CHECKPOINT is an operation to ensure that all altered data by the committed transactions are written to disk.

- The database should be in OPEN phase.
- The database should be in TDS mode.
- When a full backup is in progress, the altered pages are not recorded in the data file, but only the RE DO logs and control files are written to the disk. If the server is abnormally terminated in this situatio n, a media recovery should be performed.

<domain name>

It is a name of a member or a group for which the statement is performed.
If it is omitted, it is performed for all groups.

Description

The checkpoint operation records all changes by the committed transactions to disk, so it enables a rapid recovery at system error.

Example

The following is an example of performing CHECKPOINT.

```
ALTER SYSTEM CHECKPOINT;
```

Compatibility

The SQL standard does not define the concepts of CHECKPOINT.

18.37 ALTER SYSTEM CLEANUP BUFFER_CACHE

Function

It clears all buffer pages which can be free from the buffer cache.

Syntax

```
<alter system cleanup buffer_cache statement> ::=
    ALTER SYSTEM CLEANUP BUFFER_CACHE
    [ AT <domain name> ]
    ;
```

Invocation and Access Rules

ALTER SYSTEM ON DATABASE privilege is required to perform <alter system cleanup buffer_cache statement>.

Syntax Rules and Parameters

<alter system cleanup buffer_cache statement>

Syntax rules and parameters do not exist for <alter system cleanup buffer_cache statement>.

<domain name>

It is a name of a member or a group for which the statement is performed.
If it is omitted, it is performed for all groups.

Description

It flushes and frees all free buffer pages cached in the buffer.

It should be used to clear the buffer cache before the performance measuring.
If it is used on the operating server, then it could have fatal effect for the performance.

Example

The following is an example of performing CLEANUP BUFFER_CACHE.

```
ALTER SYSTEM CLEANUP BUFFER_CACHE;
```

Compatibility

The SQL standard does not define the concepts of CLEANUP BUFFER_CACHE.

18.38 ALTER SYSTEM CLEANUP PLAN

Function

It cleans up all SQL plans.

Syntax

```
<alter system cleanup plan statement> ::=
    ALTER SYSTEM CLEANUP PLAN
    [ AT <domain name> ]
    ;
```

Invocation and Access Rules

ALTER SYSTEM ON DATABASE privilege is required to perform <alter system cleanup plan statement>.

Syntax Rules and Parameters

<alter system cleanup plan statement>

There is not any syntax rules or parameters for <alter system cleanup plan statement>.

<domain name>

It is a name of a member or a group for which the statement is performed.
If it is omitted, it is performed for all groups.

Description

It cleans up all of the cached SQL plan. However, the plan whose V$SQL CACHE.REF COUNT is bigger tha n 0 (the plan referenced by the prepared statement) is excluded from cleanup.

Examples

The following is an example of executing CLEANUP PLAN.

```
ALTER SYSTEM CLEANUP PLAN;
```

Compatibility

The SQL standard does not define the concepts of CLEANUP PLAN.

18.39 ALTER SYSTEM IRRECOVERABLE CLUSTER MEMBER

Function

It specifies an irrecoverable cluster member.

Syntax

```
<alter system irrecoverable cluster member statement> ::=
    ALTER SYSTEM IRRECOVERABLE CLUSTER MEMBER <domain name>
    ;
```

Invocation and Access Rules

ALTER SYSTEM ON DATABASE privilege is required to perform <alter system irrecoverable cluster member statement>.

Syntax Rules and Parameters

<alter system irrecoverable cluster member statement>

There is not any syntax rules or parameters for <alter system irrecoverable cluster member statement>.

<domain name>

It is a name of an irrecoverable member.
it is not allowed to specify all members in a group as an irrecoverable member.

Description

It is used to restart the system excluding the corresponding member if the cluster failed to restart due to an irrecoverable member. The corresponding member should be dropped by using **ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS** after the system succeeded to restart.

Examples

The following is an example of executing IRRECOVERABLE CLUSTER MEMBER.

```
gSQL> ALTER SYSTEM IRRECOVERABLE CLUSTER MEMBER g1n1;
```

Compatibility

The SQL standard does not define the concepts of IRRECOVERABLE CLUSTER MEMBER.

18.40 ALTER SYSTEM JOIN DATABASE

Function

It includes a specific inactive cluster member in a cluster system again.

Syntax

```
<alter system join database statement> ::=
    ALTER SYSTEM JOIN DATABASE
    ;
```

Invocation and Access Rules

It can be performed in a cluster system.

ADMINISTRATION ON DATABASE privilege is required to perform <alter system join database statement>.

Description

The inactive state of a cluster member means that it is not connected to the cluster system, and it occurs in the following cases.

- An error occurs on a cluster member in an operating cluster system.
- Trying to start-up the cluster system without driving the cluster member included in the cluster system.

If a specific cluster member is inactive, then the member can be included in a cluster system again according to the following procedure.

- Start-up the unstarted cluster member to the local open phase.

```
$ gsql sys gliese --as sysdba --dsn=G3N2
gSQL> \startup
```

- Include it in a cluster system by using <alter system join database statement>.

```
$ gsql sys gliese --as sysdba --dsn=G3N2
gSQL> ALTER SYSTEM JOIN DATABASE;
```

Use <alter system join database statement> to make an inactive cluster member which is started up to the local open phase to participate in the cluster system without shutting it down.

To make the inactive cluster member to participate in the cluster system again, the database state of the cluster system and that of the inactive cluster member should be same.

The inactive cluster member can not participate in the cluster system again after the transaction altering t he database in the cluster system completed.

To operate the cluster system normally the inactive cluster members should be dropped according to the following procedure when multiple inactive cluster members exist.

1. JOIN inactive cluster members which can participate in the cluster system.

```
gSQL> ALTER SYSTEM JOIN DATABASE;
```

2. DROP inactive cluster members which can not participate in the cluster system.

```
gSQL> ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS;
```

All inactive cluster members which can participate in the cluster system should be included in the cluster s ystem before dropping because all inactive cluster members are dropped from the cluster system when p erforming <alter database drop inactive cluster member statement>.

Examples

```
gSQL> ALTER SYSTEM JOIN DATABASE;
```

Compatibility

The SQL standard does not define the concepts of the cluster.

For More Information

Refer to **ALTER DATABASE DROP INACTIVE CLUSTER MEMBERS**.

18.41 ALTER SYSTEM [KILL | DISCONNECT] SESSION

Function

It terminates a session.

Syntax

```
<alter system end session statement> ::=
      ALTER SYSTEM DISCONNECT SESSION [<member_position>,] <session_id>,
           <serial#> [<disconnect_option>] [AT <domain name>]
    ¦  ALTER SYSTEM KILL SESSION [<member_position>,]
           <session_id>, <serial#> [AT <domain name>]
    ;
<disconnect_option> ::=
      POST_TRANSACTION
    ¦ IMMEDIATE
```

Invocation and Access Rules

ALTER SYSTEM ON DATABASE privilege is required to perform <alter system end session statement>.

Syntax Rules and Parameters

<member_position>

It is a member position of a session which is a disconnect/kill target in a cluster environment.

<session_id>

It is the session ID.

〈serial#〉

It is the SERIAL NUMBER of the session.

〈disconnect_option〉

- POST_TRANSACTION: The session is terminated after completion of the transaction.
- IMMEDIATE: The session is immediately terminated without waiting for the completion of the transac tion.

If 〈disconnect_option〉 is not used, then it is operated in IMMEDIATE.

〈domain name〉

It is a name of a member or a group for which the statement is performed.
If it is omitted, it is performed for all groups.

Description

DISCONNECT SESSION can specify the options such as POST TRANSACTION and IMMEDIATE.
POST TRANSACTION terminates the session after the currently running transaction is completed. IMMEDI ATE terminates the session after immediately cleaning up the currently running transaction.

KILL SESSION terminates the abnormal session which remains on the system without its process.

Example

```
gSQL> SELECT USER_NAME, SESSION_ID, SERIAL_NO, SESSION_STATUS, PROGRAM_NAME FROM V$SESSION
WHERE USER_NAME = 'TEST';
USER_NAME SESSION_ID SERIAL_NO SESSION_STATUS PROGRAM_NAME
--------- ---------- --------- -------------- ------------
TEST              62        49 CONNECTED      gsql
TEST              65       109 CONNECTED      gsqlnet
TEST              66       130 CONNECTED      gsql
3 rows selected.
gSQL> ALTER SYSTEM DISCONNECT SESSION 65, 109;
System altered.
```

Compatibility

The SQL standard does not define it.

18.42 ALTER SYSTEM {MOUNT | OPEN} DATABASE

Function

It mounts the database on system, or alters the database to the state which is available for the service.

Syntax

```
<alter system database statement> ::=
    ALTER SYSTEM <alter system database clause>
    ;
<alter system database clause> ::=
      MOUNT DATABASE
    | OPEN [ <database_scope> ] DATABASE [ <open_database_option> ]
<open_database_option> ::=
      NORESETLOGS
    | RESETLOGS
<database_scope> ::=
      LOCAL
    | GLOBAL
```

Invocation and Access Rules

ADMINISTRATION ON DATABASE privilege is required to perform <alter system database statement>.

Syntax Rules and Parameters

<alter system database clause>

- MOUNT DATATABASE
  - It mounts the database on the system.
- OPEN DATABASE

- It changes the database to the state which is available for the service.

〈open database option〉

- RESETLOGS / NORESETLOGS
  - It determines whether to keep the online redo logs after recovering the database.
  - NORESETLOGS maintains the existing redo log, but RESETLOGS initializes it.
  - RESETLOGS should be specified when the database is incompletely recovered.
  - If it is omitted, NORESETLOGS is specified by default.

〈database_scope〉

- LOCAL
  - It starts up the LOCAL server to the OPEN phase.
- GLOBAL
  - It starts up the GLOBAL server, the entire server, to the OPEN phase.
- If it is omitted in a cluster environment, it starts up the GLOBAL server.

Examples

The following is an example of initializing the online redo logs.

```
ALTER SYSTEM OPEN DATABASE RESETLOGS;
```

Compatibility

The SQL standard does not define the concepts of MOUNT or OPEN in the database.

For More Information

Refer to **ALTER DATABASE RECOVER**.

18.43 ALTER SYSTEM RECONNECT GLOBAL CONNECTION

Function

It determines whether to reconnect to the session which is connected in GLOBAL CONNECTION form.

Syntax

```
<alter system reconnect global connection statement> ::=
    ALTER SYSTEM RECONNECT GLOBAL CONNECTION
    ;
```

Invocation and Access Rules

ALTER SYSTEM ON DATABASE privilege is required to perform <alter system reconnect global connection statement>.

Description

Whether the GLOBAL CONNECTION client reconnects is determined by comparing SCN of a system object acquired from a server at the first connection and SCN of current server system object. This statement leads the client to reconnect by increasing SCN of the system object.

The client dose not necessarily reconnect immediately after this statement is performed. The client reconnects by comparing SCN when the client executes a command in a server, and it does not try to reconnect if connections to all members from a client are valid.

Examples

The following is an example of executing the statement.

```
gSQL> ALTER SYSTEM RECONNECT GLOBAL CONNECTION;
System altered.
```

Compatibility

The SQL standard does not define the concepts of GLOBAL CONNECTION.

18.44 ALTER SYSTEM RESET property_name

Function

It removes a property value from the property file.

Syntax

```
<alter system reset statement> ::=
    ALTER SYSTEM { RESET ¦ UNSET } <property name>
        [ SCOPE = { FILE ¦ SPFILE } ]
        [ AT <domain name>]
    ;
```

Invocation and Access Rules

ALTER SYSTEM ON DATABASE privilege is required to perform <alter system reset statement>.

Syntax Rules and Parameters

{ RESET | UNSET }

RESET and UNSET are the reserved words with the same meaning, so either of them can be used.

<property name>

It is the property name to be removed.
For more information, refer to **Server Property** in an administration manual.

[SCOPE = { FILE | SPFILE }]

It removes the property from a property file, so only SCOPE=FILE/SPFILE can be used.

- SCOPE = FILE

- FILE and SPFILE are the reserved words with the same meaning, so either of them can be used.
- A property is removed from FILE, and is not applied to the current state.
- When restarting the database, the changes are applied.

If SCOPE clause is not specified, the default value is SCOPE = FILE.

〈domain name〉

It is a name of a member or a group for which the statement is performed.
If it is omitted, it is performed for all groups.

Description

If a property is altered by using SCOPE=FILE/SPFILE, the updated property value is stored in the property fi le, and it is applied when restarting the database.

When executing RESET, the updated property value is removed from the property file and the default val ue is used when restarting the database.

Examples

The following is an example of altering the property by using SCOPE=FILE.

```
gSQL> ALTER SYSTEM SET PROCESS_MAX_COUNT=128 SCOPE=FILE;
System altered.
```

The following is an example of removing the property altered above.

```
gSQL> ALTER SYSTEM RESET PROCESS_MAX_COUNT SCOPE=FILE;
System altered.
gSQL> ALTER SYSTEM RESET PROCESS_MAX_COUNT SCOPE=SPFILE;
System altered.
gSQL> ALTER SYSTEM RESET PROCESS_MAX_COUNT;
System altered.
gSQL> ALTER SYSTEM UNSET PROCESS_MAX_COUNT;
System altered.
```

Compatibility

The SQL standard does not define the concepts of the system property.

For More Information

Refer to **ALTER SYSTEM SET property_name**.

18.45 ALTER SYSTEM SET property_name

Function

It sets the system property value.

Syntax

```
<alter system set statement> ::=
    ALTER SYSTEM SET <property name> { = <property value> ¦ TO DEFAULT }
        [ DEFERRED ]
        [ SCOPE = [ MEMORY ¦ { FILE ¦ SPFILE } ¦ BOTH ] ]
    [AT <domain name>]
    ;
```

Invocation and Access Rules

ALTER SYSTEM ON DATABASE privilege is required to perform <alter system set statement>.

Syntax Rules and Parameters

<property name>

It is the property name to be set.
For more information, refer to **Server Property** in an administration manual.

<property value>

It is the property value to be set.

TO DEFAULT

It sets the system property value as the initial value of system driving.

[DEFERRED]

It defines the point of time to apply the altered property.

- DEFERRED
  - It does not effect the current SESSION, but it is applied to the newly generated SESSION.
  - It can be applied when ISSYS_MODIFIABL property value is IMMEDIATE/DEFERRED. It should be explicitly specified.
  - It is not applicable when the ISSYS_MODIFIABLE property value is FALSE.

If ISSYS_MODIFIABLE property value is IMMEDIATE, and DEFERRED is not explicitly specified, then it is im mediately applied to all sessions.

[SCOPE = [MEMORY | { FILE | SPFILE } | BOTH]]

It specifies the range which is affected by the property changes of the system.

- SCOPE = MEMORY
  - The changes are applied only to the current state, and the value is lost when restarting the datab ase.
- SCOPE = FILE
  - FILE and SPFILE are the reserved words with the same meaning, so either of them can be used.
  - The changes are stored in FILE, and not applied to the current state.
  - The changes are applied when restarting the database.
- SCOPE = BOTH
  - The changes are stored in FILE, and applied to the current state.

If SCOPE clause is omitted, the default value is SCOPE = MEMORY.
If ISSYS_MODIFIABLE property value is FALSE, it should be specified as SCOPE=FILE/SPFILE.

〈domain name〉

It is a name of a member or a group for which the statement is performed.
If it is omitted, it is performed for all groups.

Description

For more information, refer to **Server Property** in an administration manual.

Examples

The following is an example of changing the property whose ISSYS_MODIFIABLE property is DEFERRED.

```
gSQL> ALTER SYSTEM SET SPIN_COUNT=1000;
ERR-22000(13019): Specified property cannot be modified.(SPIN_COUNT)
gSQL> ALTER SYSTEM SET SPIN_COUNT=1000 DEFERRED;
System altered.
```

The following is an example of changing the property whose ISSYS_MODIFIABLE property is FALSE.

```
gSQL> ALTER SYSTEM SET PROCESS_MAX_COUNT=128;
ERR-22000(13018): Specified property cannot be modified with this SCOPE
option.(PROCESS_MAX_COUNT)
gSQL> ALTER SYSTEM SET PROCESS_MAX_COUNT=128 SCOPE=FILE;
System altered.
```

The following is an example of changing the altered property to the default value of when the session wa s connected.

```
gSQL> ALTER SYSTEM SET TRANSACTION_COMMIT_WRITE_MODE=0;
System altered.
gSQL> ALTER SYSTEM SET TRANSACTION_COMMIT_WRITE_MODE TO DEFAULT;
System altered.
gSQL> ALTER SYSTEM SET TRANSACTION_COMMIT_WRITE_MODE TO DEFAULT DEFERRED;
System altered.
```

Compatibility

The SQL standard does not define the concepts of the system property.

For More Information

Refer to **ALTER SYSTEM RESET property_name**.

18.46 ALTER SYSTEM SWITCH LOGFILE

Function

It alters the log files in CURRENT state to ACTIVE state in database.

Syntax

```
<alter system switch logfile statement> ::=
    ALTER SYSTEM SWITCH LOGFILE
    [ AT <domain name> ]
    ;
```

Invocation and Access Rules

ALTER SYSTEM ON DATABASE privilege is required to perform <alter system switch logfile statement>.

Syntax Rules and Parameters

<alter system switch logfile statement>

The database should be in MOUNT or OPEN phase.

<domain name>

It is a name of a member or a group for which the statement is performed.
If it is omitted, it is performed for all groups.

Description

Basically, if the log file in CURRENT state is filled, the log switch automatically occurs. That statement is used to forcibly execute log switch in special circumstances.

Example

```
ALTER SYSTEM SWITCH LOGFILE;
```

Compatibility

The SQL standard does not define the concepts of the LOGFILE.

For More Information

Refer to **ALTER SYSTEM {MOUNT | OPEN} DATABASE**.

18.47 ALTER TABLE

Function

It alters the table definition.

Syntax

```
<alter table statement> ::=
      <alter table physical attribute statement>
    ¦ <rename table statement>
    ¦ <add column definition>
    ¦ <drop column definition>
    ¦ <alter column definition>
    ¦ <rename column statement>
    ¦ <add table constraint definition>
    ¦ <drop table constraint definition>
    ¦ <alter table constraint definition>
    ¦ <alter table drop offline segments statement>
    ¦ <rename table constraint statement>
    ¦ <add table supplemental log statement>
    ¦ <drop table supplemental log statement>
    ¦ <rebalance statement>
    ¦ <move shard statement>
    ¦ <merge shards statement>
    ¦ <split shard statement>
    ¦ <alter table synchronize statement>
    ¦ <rename shard statement>
    ¦ <read { only ¦ write } statement>
    ;
```

Invocation and Access Rules

One of the following privilege is required to perform <alter table statement>.

- (ALTER or CONTROL TABLE) ON TABLE for the table
- (ALTER TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- ALTER ANY TABLE ON DATABASE

Syntax Rules and Parameters

⟨alter table physical attribute statement⟩

It alters physical attributes of a table.
For more information, refer to **ALTER TABLE name STORAGE**.

⟨rename table statement⟩

It renames the table.
For more information, refer to **ALTER TABLE name RENAME TO**.

⟨add column definition⟩

It adds columns to the table.
For more information, refer to **ALTER TABLE name ADD COLUMN**.

⟨drop column definition⟩

It drops a column from the table.
For more information, refer to **ALTER TABLE name SET UNUSED COLUMN**.

⟨alter column definition⟩

It alters the column definition in the table.
For more information, refer to **ALTER TABLE name ALTER COLUMN**.

⟨rename column statement⟩

It renames the column in the table.
For more information, refer to **ALTER TABLE name RENAME COLUMN**.

〈add table constraint definition〉

It adds constraints to the table.
For more information, refer to **ALTER TABLE name ADD CONSTRAINT**.

〈drop table constraint definition〉

It drops the constraints of the table.
For more information, refer to **ALTER TABLE name DROP CONSTRAINT**.

〈alter table constraint definition〉

It alters the constraints of the table.
For more information, refer to **ALTER TABLE name ALTER CONSTRAINT**.

〈alter table drop offline segments statement〉

It drops offline shards in the table.
For more information, refer to **ALTER TABLE name DROP OFFLINE SEGMENTS**.

〈rename table constraint statement〉

It renames the constraints of the table.
For more information, refer to **ALTER TABLE name RENAME CONSTRAINT**.

〈add table supplemental log statement〉

It sets to add information to the redo log when the data is altered in the table.
For more information, refer to **ALTER TABLE name ADD SUPPLEMENTAL LOG**.

〈drop table supplemental log statement〉

It sets not to add information to the redo log when the data is altered in the table.
For more information, refer to **ALTER TABLE name DROP SUPPLEMENTAL LOG**.

〈rebalance statement〉

It restores the data file matching by rebalancing the shard of the table or by synchronizing the broken sha rd in a cluster environment.
For more information, refer to **ALTER TABLE name REBALANCE**.

〈alter table synchronize statement〉

It restores the data file matching by synchronizing the existing offline shards in cluster environment.
For more information, refer to **ALTER TABLE name SYNCHRONIZE**.

〈move shard statement〉

It rebalances a specific shard of a table on a specific cluster group in a cluster environment.
For more information, refer to **ALTER TABLE name MOVE SHARD** .

〈merge shards statement〉

It merges specific shards in a table in a cluster environment, then rebalances them.
For more information, refer to **ALTER TABLE name MERGE SHARDS**.

〈split shard statement〉

It rebalances a specific shard of a table on a specific cluster group by splitting the shard in a cluster enviro nment.
For more information, refer to **ALTER TABLE name SPLIT SHARD**.

〈rename shard statement〉

It renames a specific shard of a table in cluster environment.
For more information, refer to **ALTER TABLE name RENAME SHARD**.

〈read { only | write } statement〉

It sets READ (only | write } to a table.
For more information, refer to **ALTER TABLE name READ { ONLY | WRITE }**.

Description

For more information, refer to the description of each detailed statement.

Example

Refer to the examples of each detailed statement.

Compatibility

The SQL standard does not define the following statements.

- 〈alter table physical attribute statement〉
- 〈rename table statement〉
- 〈rename column statement〉
- 〈rename table constraint statement〉
- 〈add table supplemental log statement〉
- 〈drop table supplemental log statement〉
- 〈rebalance statement〉
- 〈move shard statement〉
- 〈split shard statement〉
- 〈rename shard statement〉
- 〈read { only | write } statement〉

18.48 ALTER TABLE name ADD COLUMN

Function

It adds a column to the table.

Syntax

```
<add column definition> ::=
      ALTER TABLE table_name ADD [ COLUMN ] <column definition>
    ¦ ALTER TABLE table_name ADD [ COLUMN ] ( <column definition> [, ...] )
    ;
```

Invocation and Access Rules

The user should satisfy the following conditions to perform <add column definition>.

- At least one of the following privileges is required to alter the table.
  - (ALTER or CONTROL TABLE) ON TABLE for that table
  - (ALTER TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - ALTER ANY TABLE ON DATABASE

- If the constraints are specified with the added columns, the conditions should be satisfied to generate the constraints as follows.
  - One of the following privileges is required for the schema in which constraints are to be generated.
    - (ADD CONSTRAINT or CONTROL SCHEMA) ON SCHEMA for the schema
    - ALTER ANY TABLE ON DATABASE
  - If the key constraint is to be generated, one of the following privileges is required for the tablespace in which an index is to be created.
    - CREATE OBJECT ON TABLESPACE for the tablespace
    - USAGE TABLESPACE ON DATABASE

- The table owner has the following privileges for the added columns.
  - Privileges on all added columns
    - SELECT(columns) ON TABLE WITH GRANT OPTION

        - INSERT(columns) ON TABLE WITH GRANT OPTION
        - UPDATE(columns) ON TABLE WITH GRANT OPTION
        - REFERENCES(columns) ON TABLE WITH GRANT OPTION
    - Privilege on the constraint generated together
        - The owner of that constraint
        - The index owner generated together with the constraint

Syntax Rules and Parameters

table_name

It is the table name to be altered.
It can define the schema to which the table belongs, such as schema_name.table_name. If schema_name is omitted, the default schema name of the user performing the statement is used.

ADD [COLUMN]

The reserved word COLUMN can be omitted.

〈column definition〉

It defines the column to be added. For more information, refer to **〈column definition〉** clause of **CREATE TABLE** statement.
There should not be columns with the same name in a table.

If DEFAULT clause is specified when defining the column, the default value of all rows are stored in the added column.
If 〈identity column specification〉 clause is specified when defining the column, each automatically generated value of all rows is stored in the added column.
If NOT NULL constraint is specified when defining the column, the table should be empty or it should be specified together with DEFAULT or 〈identity column specification〉 clause.

(〈column definition〉 [, ...])

It adds multiple columns.
It lists multiple 〈column definition〉 inside the parentheses.

Description

The added column is positioned at the end of the existing columns.
When specifying DEFAULT or <identity column specification> clause, the processing time is increased in p roportion to the number of the rows in the table.

Examples

The following is an example of adding a column.

```
gSQL> ALTER TABLE region ADD COLUMN r_new_comment VARCHAR(152);
Table altered.
```

The following is an example of adding multiple columns.

```
gSQL> ALTER TABLE partsupp ADD COLUMN (
   ps_retailprice NUMERIC(12,2),
   ps_acctbal NUMERIC(12,2), ps_mktsegment  CHAR(10) );
Table altered.
```

The following is an example of adding the identity column and the column including DEFAULT clause.

```
gSQL> ALTER TABLE region ADD COLUMN (
    r_regionkey INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    r_comment   VARCHAR(152) DEFAULT 'N/A' );
Table altered.
gSQL> SELECT r_regionkey, r_name, r_comment FROM region;
R_REGIONKEY R_NAME                    R_COMMENT
----------- ------------------------- ---------
          1 AFRICA                    N/A
          2 AMERICA                   N/A
          3 ASIA                      N/A
          4 EUROPE                    N/A
          5 MIDDLE EAST               N/A
5 rows selected.
```

The following is an example of adding the column including the deferrable constraint.

```
gSQL> ALTER TABLE t1 ADD COLUMN ( id INTEGER CONSTRAINT t1_uk UNIQUE DEFERRABLE );
Table altered.
gSQL> COMMIT;
Commit complete.
```

Compatibility

The SQL standard does not define of adding multiple column definitions.

For More Information

Refer to the followings.

- ALTER TABLE
- ALTER TABLE name SET UNUSED COLUMN
- ALTER TABLE name ALTER COLUMN
- ALTER TABLE name RENAME COLUMN

18.49 ALTER TABLE name ADD CONSTRAINT

Function

It adds a table constraint.

Syntax

```
<add table constraint definition> ::=
    ALTER TABLE table_name
        ADD <table constraint definition>
    ;
```

Invocation and Access Rules

The user should satisfy the following conditions to perform <add table constraint definition> clause.

- One of the following privileges is required on the table to create the constraint.
  - (ALTER or CONTROL TABLE) ON TABLE for that table
  - (ALTER TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - ALTER ANY TABLE ON DATABASE

- One of the following privileges is required on the schema to create the constraint.
  - (ADD CONSTRAINT or CONTROL SCHEMA) ON SCHEMA for the schema
  - ALTER ANY TABLE ON DATABASE

- One of the following privileges is required for the tablespace in which the index is to be created to cr eate the key constraint
  - CREATE OBJECT ON TABLESPACE for the tablespace
  - USAGE TABLESPACE ON DATABASE

- The owner of the created constraint is determined as follow.
  - The owner of the schema to which the constraint belongs.
  - If the schema to which the constraint belongs is PUBLIC, then it is the user who executed the stat ement.


Constraints of PRIMARY KEY, UNIQUE in a cluster system should include all sharding keys.

Syntax Rules and Parameters

table_name

It is the table name to be altered.
It can define the schema to which the table belongs, such as schema_name.table_name. If schema_name is omitted, the default schema name of the user performing the statement is used.

〈table constraint definition〉

It defines the constraint to be added.
NOT NULL constraint can not be added by using ALTER TABLE .. ADD CONSTRAINT statement, and it can be defined by using **ALTER TABLE name ALTER COLUMN** statement as follows.

```
ALTER TABLE t1 ALTER COLUMN c1 SET NOT NULL;
```

For more information, refer to **〈table constraint definition〉** clause of **CREATE TABLE** statement.

Description

When adding the key constraints such as primary key, unique key, the index is automatically created for t hem.

Examples

The following is an example of adding the primary key constraint to the table.

```
gSQL> ALTER TABLE t1 ADD PRIMARY KEY ( id );
Table altered.
```

The following is an example of specifying the constraint name when adding a primary key constraint to th e table.

```
gSQL> ALTER TABLE t1 ADD CONSTRAINT t1_pk PRIMARY KEY ( id );
Table altered.
```

The following is an example of adding a DEFERRABLE constraint.

```
gSQL> ALTER TABLE t1 ADD CONSTRAINT t1_uk UNIQUE ( id ) DEFERRABLE INITIALLY DEFERRED;
Table altered.
gSQL> COMMIT;
Commit complete.
```

Compatibility

Table 18-4 SQL standard compatibility

Feature ID	Description	Compatibility
F381	Extended schema manipulation	O

For More Information

Refer to the followings.

- CREATE TABLE
- CREATE INDEX
- ALTER TABLE
- ALTER TABLE name DROP CONSTRAINT

18.50 ALTER TABLE name ADD GLOBAL SECONDARY INDEX

Function

It creates a global secondary index in a table.

Syntax

```
<alter table add global secondary index definition> ::=
    ALTER TABLE table_name
        ADD GLOBAL SECONDARY INDEX
        [ <index attributes> [...] ] [ TABLESPACE tablespace_name ]
    ;
<index attributes> ::=
      <physical attribute clause>
    ¦ STORAGE ( <segment attr clause> [...] )
    ¦ <parallel clause>
<physical attribute clause> ::=
      PCTFREE integer
    ¦ INITRANS integer
    ¦ MAXTRANS integer
<segment attr clause> ::=
      INITIAL <size_clause>
    ¦ NEXT <size_clause>
    ¦ MINSIZE <size_clause>
    ¦ MAXSIZE <size_clause>
<size clause> ::=
      integer [ K ¦ M ¦ G ¦ T ]
<parallel clause> ::=
      NOPARALLEL
    ¦ PARALLEL [ integer ]
```

Invocation and Access Rules

〈alter table add global secondary index definition〉 can be defined in a cluster system, and a user should s atisfy the following conditions.

- At least one of the following privileges for a table in which the index is to be created is required.
  - (ALTER or CONTROL TABLE) ON TABLE for that table
  - (ALTER TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs.
  - ALTER ANY TABLE ON DATABASE

- At least one of the following privileges for a tablespace in which the index is to be created is required.
  - CREATE OBJECT ON TABLESPACE for that tablespace
  - USAGE TABLESPACE ON DATABASE

Syntax Rules and Parameters

table_name

It is the name of a table in which the index is to be created.
It can define a schema to which the table belongs such as schema_name.table_name.
If schema_name is omitted, the default schema name of the user performing the statement is used.

〈physical attribute clause〉

It defines the physical attribute information of the index.

- PCTFREE integer
  - Definition
    - The reserved space to adjust the frequency of page splits caused by inserting the key in the p age.
    - It is applied only when the index bottom-up build.
  - It can use of the value from 0 to 99.
  - If it is omitted, the value set in DEFAULT_INDEX_PCTFREE property is used by default.

- INITRANS integer
  - Definition
    - The initial number of transactions simultaneously accessing the page.
    - If the number of users accessing the index is small, then INITRANS is set low. If the number o f users simultaneously accessing the index is big, then INITRANS is set high.
    - If necessary, it is automatically increased to the specified MAXTRANS.

- It can use the value from 1 to 32.
- If it is omitted, the default value is 4.

- MAXTRANS integer
  - Definition
    - It specifies the maximum number of transaction simultaneously accessing the page.
  - It can use the value from 1 to 32.
  - If it is omitted, the default value is 8.

〈segment attr clause〉

It specifies the information for the index storage space.

- INITIAL integer
  - Definition
    - It specifies the size of physical storage space which is initially allocated when creating the ind ex.
    - This size is aligned to the EXTENT size of the TABLESPACE to which the table belongs. (e.g. If the EXT size is 8192 bytes, 'INITIAL 100' is actually operated as 8192 bytes.)
    - The size (aligned to the EXTENT size of TABLESPACE) should be equal to or bigger than MIN EXTENTS, or it should be equal to or less than MAXEXTENTS.
  - The minimum value is 1, and the maximum value depends on the system environment.
  - If it is omitted, the default value is one EXTENT size of TABLESPACE to which the table belongs.

- NEXT integer
  - Definition
    - It specifies the physical space size to be allocated when adding the space to the index.
    - This size is aligned to the EXTENT size of the TABLESPACE to which the table belongs. (e.g. If the EXT size is 8192 bytes, 'NEXT 100' is actually operated as 8192 bytes.)
    - NEXT operates as follows, depending on the remaining space size of the index available curr ently. (Obtained by subtracting the amount of currently used space from the MAXEXTENTS si ze)

    - If the remaining space size is 0, then it can not extend the space.
    - If the remaining space size is bigger than 0, but smaller than NEXT, then it allocates the space as big as the remaining space.
    - If the remaining space size is bigger than NEXT, then it allocates the space as big as the NEXT.
  - The minimum value is 1 and the maximum value depends on the system environment.
  - If it is omitted, the default value is one EXTENT size of TABLESPACE to which the index belongs.

- MINSIZE integer
  - Definition
    - It is the minimum space size of the index.
    - The value should be equal to or smaller MAXSIZE.

    - This size is aligned to the EXTENT size of the TABLESPACE to which the index belongs.
    - The minimum value is 1 and the maximum value depends on the system environment.
    - If it is smaller than the size of two EXTENT, it is specified to the size of two EXTENT.
    - If it is omitted, the default value is the size of two EXTENT.

- MAXSIZE integer
    - Definition
        - It is the maximum space size of the index.
        - The value should be equal to or bigger than MINSIZE.
    - This size is aligned to the EXTENT size of the TABLESPACE to which the index belongs.
    - The minimum value is 1 and the maximum value depends on the system environment.
    - If it is omitted, the default value is EXTENT size * 2147483647(The maximum positive integer of I NT32).

〈size clause〉

It specifies the file size in byte. (If it is omitted, the default unit is bytes.)

- K: Kilobytes
- M: Megabytes
- G: Gigabytes
- T: Terabytes

NOPARALLEL | PARALLEL [integer]

It specifies the number of threads to be used when building an index.

- NOPARALLEL
    - It does not build an index in parallel.
- PARALLEL [integer]
    - It builds an index in parallel.
    - If an integer is omitted or set as 0, then it follows the property (INDEX_BUILD_PARALLEL_FACTO R).
    - The minimum value of an intege is 0 and the maximum value is 16.
    - If the property value is 0, then the system determines the optimal value.
- If it is omittted, the default value is PARALLEL.

TABLESPACE tablespace_name

It specifies the name of the tablespace in which the index is to be stored.

- If it specifies tablespace_name

    - tablespace_name should be a data tablespace to switch to the LOGGING index.
    - tablespace_name should be a temporary tablespace or a nologging tablespace to switch to the NOLOGGING index

- If it omits TABLESPACE clause, then it follows the settings of the existing index.

Description

A non-deterministic query requires the global secondary index. LOGGING index and NOLOGGING inde have the following trade-offs.

- LOGGING index
    - Advantage: It does not separately build an index because the index is automatically restored by using the log when starting up the system.
    - Disadvantage: A disk I/O occur because the changes on the index is recorded on the log when altering the row.
- NOLOGGING index
    - Advantage: A disk I/O does not occur for the changes on the index when altering the row.
    - Disadvantage: It automatically rebuilds the index when starting up the system because the log information of the index does not exist.

Examples

The following is an example of adding a global secondary index to the table T1.

```
gSQL> ALTER TABLE T1 ADD GLOBAL SECONDARY INDEX;
Table altered.
gSQL> COMMIT;
Commit complete.
```

The following is an example of creating a global secondary index as a logging index on the tablespace USER_DATA_TBS of table T1.

```
gSQL> ALTER TABLE T1 ADD GLOBAL SECONDARY INDEX LOGGING TABLESPACE USER_DATA_TBS;
Table altered.
gSQL> COMMIT;
Commit complete.
```

The following is an example of creating a global secondary index as a nologging index on the tablespace

USER_TEMP_TBS of table T1.

```
gSQL> ALTER TABLE T1 ADD GLOBAL SECONDARY INDEX NOLOGGING TABLESPACE USER_TEMP_TBS;
Table altered.
gSQL> COMMIT;
Commit complete.
```

Compatibility

The SQL standard does not define the concepts of the global secondary index.

For More Information

Refer to the followings.

- ALTER TABLE name DROP GLOBAL SECONDARY INDEX
- ALTER TABLE name ALTER GLOBAL SECONDARY INDEX
- CREATE TABLE

18.51 ALTER TABLE name ADD SUPPLEMENTAL LOG

Function

If the primary key exists in the table when table data is altered, it sets to add the primary key value to redo log.

Syntax

```
<add table supplemental log statement> ::=
    ALTER TABLE table_name
        ADD SUPPLEMENTAL LOG DATA ( PRIMARY KEY ) COLUMNS
    ;
```

Invocation and Access Rules

One of the following privileges is required to perform <add table supplemental log statement>.

- (ALTER or CONTROL TABLE) ON TABLE for that table
- (ALTER TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- ALTER ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the table name to be altered.
It can define the schema to which the table belongs, such as schema_name.table_name. If schema_name is omitted, the default schema name of the user performing the statement is used.

Even when the primary key does not exist in the table, the statement can be executed.

Description

It additionally records SUPPLEMENTAL LOG when executing UPDATE/DELETE on the corresponding TABLE. The recorded SUPPLEMENTAL LOG is used to analyze logs or tools such as CDC.

To record SUPPLEMENTAL LOG of every TABLE, set the property as *SUPPLEMENTAL_LOG_DATA_PRIMARY_KEY = YES*.

Example

The following is an example of setting to additionally add a primary key value to the redo log when changing the data in the table.

```
gSQL> ALTER TABLE t1 ADD SUPPLEMENTAL LOG DATA ( PRIMARY KEY ) COLUMNS;
Table altered.
```

Compatibility

The SQL standard does not cover <add table supplemental log statement>.

For More Information

Refer to **ALTER TABLE name DROP SUPPLEMENTAL LOG**.

18.52 ALTER TABLE name ALTER COLUMN

Function

It alters the column definition.

Syntax

```
<alter column definition> ::=
    ALTER TABLE table_name
        ALTER [ COLUMN ] column_name <alter column action>
<alter column action> ::=
      <set column default clause>
    ¦ <drop column default clause>
    ¦ <set column not null clause>
    ¦ <drop column not null clause>
    ¦ <alter column data type clause>
    ¦ <alter identity column specification>
    ¦ <drop identity property clause>
    ;
<set column default clause> ::=
    SET DEFAULT <default option>
<drop column default clause> ::=
    DROP DEFAULT
<set column not null clause> ::=
    SET [ CONSTRAINT constraint_name ] NOT NULL [ <constraint characteristics> ]
<constraint characteristics> ::=
      [ NOT ] DEFERRABLE [ <constraint check time> ]
    ¦ <constraint check time> [ [ NOT ] DEFERRABLE ]
<constraint check time> ::=
      INITIALLY DEFERRED
    ¦ INITIALLY IMMEDIATE
<drop column not null clause> ::=
    DROP NOT NULL
<alter column data type clause> ::=
    SET DATA TYPE <data type>
<alter identity column specification> ::=
```

```
     <set identity column generation clause> [ <alter identity column option> ... ]
   ¦ <alter identity column option> ...
<set identity column generation clause> ::=
   SET GENERATED { ALWAYS ¦ BY DEFAULT }
<alter identity column option> ::=
     <alter sequence generator restart option>
   ¦ [ SET ] <basic sequence generator option>
<alter sequence generator restart option> ::=
    RESTART [ WITH integer ]
<basic sequence generator option> ::=
      <sequence generator increment by option>
    ¦ <sequence generator maxvalue option>
    ¦ <sequence generator minvalue option>
    ¦ <sequence generator cycle option>
    ¦ <sequence generator cache option>
<drop identity property clause> ::=
    DROP IDENTITY
```

Invocation and Access Rules

One of the following privileges is required to performing <alter column definition>.

- (ALTER or CONTROL TABLE) ON TABLE for that table
- (ALTER TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- ALTER ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the table name to be altered.
It can define the schema to which the table belongs, such as schema_name.table_name. If schema_name is omitted, the default schema name of the user performing the statement is used.

ALTER [COLUMN]

The reserved word COLUMN can be omitted.

column_name

It is the column name to be altered.

〈set column default clause〉

It sets the default value of the column.
It should not be an identity column.

The default value set when using the DEFAULT clause is used in INSERT statement later.

The data type of DEFAULT expression should be compatible with the data type of the column.
If the data type is not compatible or the expression is not valid, an error occurs.

For more information, refer to **〈default clause〉** of **CREATE TABLE** statement.

〈drop column default clause〉

It drops the default value of the column.
It should not be an identity column.
If the default value is dropped, NULL is set when using DEFAULT clause in INSERT statement.

〈set column not null clause〉

- SET [CONSTRAINT constraint_name] NOT NULL [〈constraint characteristics〉]
  - It sets NOT NULL constraint on the column.
  - NULL is not allowed as the column value.
  - NULL should not exist in the column.

- If [CONSTRAINT constraint_name] is omitted, the constraint name is automatically given.
- If 〈constraint characteristics〉 is omitted, it has NOT DEFERRABLE INITIALLY IMMEDIATE property.
- The Identity column can not have DEFERRABLE property.

For more information about the DEFERRABLE constraint, refer to **SET CONSTRAINTS**.

〈drop column not null clause〉

- DROP NOT NULL
  - It drops NOT NULL constraint from the column.

〈alter column data type clause〉

- SET DATA TYPE 〈data type〉
  - It changes the data type of the column.


SET DATA TYPE is a DDL statement which is automatically committed.

The type conversion can be executed among the same family, and it should satisfy the following conditio ns.

Table 18-5 Conversion of character string type

from ₩ to	CHAR(n)	VARHCAR(n)	LONG VARCHAR
CHAR(m)	X	X	X
VARCHAR(m)	X	n >= m	X
LONG VARCHAR	X	X	O

The conversion of char length unit should satisfy the following condition.

Table 18-6 Conversion of character length unit

from ₩ to	OCTETS	CHARACTERS
OCTETS	O	O
CHARACTERS	X	O

Table 18-7 Conversion of binary string type

from ₩ to	BINARY(n)	VARBINARY(n)	LONG VARBINARY
BINARY(m)	X	X	X
VARBINARY(m)	X	n >= m	X
LONG VARBINARY	X	X	O

Table 18-8 Conversion of numeric type

from ₩ to	SMALLINT	INTEGER	BIGINT	NUMERIC	NUMERIC(q)	NUMERIC(q,t)	NUMBER(q)	NUMBER(q,t)	REAL	DOUBLE PRECISION	FLOAT	FLOAT(q)	NUMBER
SMALLINT	O	O	O	O	q >= 5	q >= 5 t >= 0 (q-t) >= 5	q >= 5	q >= 5 t >= 0 (q-t) >= 5	O	O	O	ddc(q) >= 5	O
INTEGER	X	O	O	O	q >= 10	q >= 10 t >= 0 (q-t) >= 10	q >= 10	q >= 10 t >= 0 (q-t) >= 10	X	O	O	ddc(q) >= 10	O
BIGINT	X	X	O	O	q >= 19	q >= 19 t >= 0 (q-t) >= 19	q >= 19	q >= 19 t >= 0 (q-t) >= 19	X	X	O	ddc(q) >= 19	O
NUMERIC	X	X	X	O	q == 38	q == 38 t == 0	q == 38	q == 38 t == 0	X	X	O	ddc(q) == 38	O
NUMERIC(p)	5 >= p	10 >= p	19 >= p	O	q >= p	q >= p t >= 0 (q-t) >= p	q >= p	q >= p t >= 0 (q-t) >= p	8 >= p	16 >= p	O	ddc(q) >= p	O
NUMERIC(p,s)	5 >= p 0 >= s 5 >= (p-s)	10 >= p 0 >= s 10 >= (p-s)	19 >= p 0 >= s 19 >= (p-s)	38 >= p 0 >= s 38 >= (p-s)	q >= p 0 >= s q >= (p-s)	q >= p t >= s (q-t) >= (p-s)	q >= p 0 >= s q >= (p-s)	q >= p t >= s (q-t) >= (p-s)	8 >= p	16 >= p	O	ddc(q) >= p	O
NUM	5 >=				q >=	q >= p t >=	q >=	q >= p	8 >=	16 >=		ddc(q)	

from ₩ to	SMALLINT	INTEGER	BIGINT	NUMERIC	NUMERIC(q)	NUMERIC(q,t)	NUMBER(q)	NUMBER(q,t)	REAL	DOUBLE PRECISION	FLOAT	FLOAT(q)	NUMBER
BER(p)	p	10 >= p	19 >= p	O	p	0 (q-t) >= p	p	t >= 0 (q-t) >= p	p	p	O	>= p	O
NUMBER(p,s)	5 >= p 0 >= s 5 >= (p-s)	10 >= p 0 >= s 10 >= (p-s)	19 >= p 0 >= s 19 >= (p-s)	38 >= p 0 >= s 38 >= (p-s)	q >= p 0 >= s q >= (p-s)	q >= p t >= s (q-t) >= (p-s)	q >= p 0 >= s q >= (p-s)	q >= p t >= s (q-t) >= (p-s)	8 >= p	16 >= p	O	ddc(q) >= p	O
REAL	X	X	X	X	X	X	X	X	O	O	O	ddc(q) >= 8	O
DOUBLE PRECISION	X	X	X	X	X	X	X	X	X	O	O	ddc(q) >= 16	O
FLOAT	X	X	X	X	X	X	X	X	X	X	O	ddc(q) == 38	O
FLOAT(p)	X	X	X	X	X	X	X	X	8 >= ddc(p)	16 >= ddc(p)	O	ddc(q) >= ddc(p)	O
NUMBER	X	X	X	X	X	X	X	X	X	-	O	ddc(q) == 38	O

Decimal digit count (ddc) value for the FLOAT (p) is as follows.

Table 18-9 Decimal digit count (ddc) value

FLOAT(p)	ddc(p)
1 ~ 3	1
4 ~ 6	2
7 ~ 9	3
10 ~ 13	4
14 ~ 16	5
17 ~ 19	6
20 ~ 23	7
24 ~ 26	8
27 ~ 29	9

FLOAT(p)	ddc(p)
30 ~ 33	10
34 ~ 36	11
37 ~ 39	12
40 ~ 43	13
44 ~ 46	14
47 ~ 49	15
50 ~ 53	16
54 ~ 56	17
57 ~ 59	18
60 ~ 63	19
64 ~ 66	20
67 ~ 69	21
70 ~ 73	22
74 ~ 76	23
77 ~ 79	24
80 ~ 83	25
84 ~ 86	26
87 ~ 89	27
90 ~ 93	28
94 ~ 96	29
97 ~ 99	30
100 ~ 103	31
104 ~ 106	32
107 ~ 109	33
110 ~ 113	34
114 ~ 116	35
117 ~ 119	36
120 ~ 123	37
124 ~ 126	38

All numeric types are managed in the same structure, and each numeric type is as same as the following NUMBER (p, s) expression.

Table 18-10 NUMBER expressions of the numeric types

Numeric type	NUMBER(p,s) expression
SMALLINT	NUMBER(5,0)
INTEGER	NUMBER(10,0)
BIGINT	NUMBER(19,0)
NUMERIC	NUMBER(38,0)
NUMERIC(p)	NUMBER(p,0)

Numeric type	NUMBER(p,s) expression
NUMERIC(p,s)	NUMBER(p,s)
NUMBER(p)	NUMBER(p,0)
NUMBER(p,s)	NUMBER(p,s)
REAL	NUMBER(8,N/A) <= FLOAT(24)
DOUBLE PRECISION	NUMBER(16,N/A) <= FLOAT(53)
FLOAT	NUMBER(38,N/A) <= FLOAT(126)
FLOAT(p)	NUMBER(ddc(p), N/A)
NUMBER	NUMBER(38, N/A)

Native numeric type is as same with as C language numeric type, and it can not be converted to another t ype.

Table 18-11 Conversion of native numeric type

from ₩ to	NATIVE_SMALLINT	NATIVE_INTEGER	NATIVE_BIGINT	NATIVE_REAL	NATIVE_DOUBLE
NATIVE_SMALLINT	O	X	X	X	X
NATIVE_INTEGER	X	O	X	X	X
NATIVE_BIGINT	X	X	O	X	X
NATIVE_REAL	X	X	X	O	X
NATIVE_DOUBLE	X	X	X	X	O

Table 18-12 Conversion of boolean type

from ₩ to	BOOLEAN
BOOLEAN	O

Table 18-13 Conversion of date/time type (TZ: WITH TIME ZONE)

from ₩ to	DATE	TIME	TIME(g)	TIME TZ	TIME(g) TZ	TIMESTAMP	TIMESTAMP(g)	TIMESTAMP TZ	TIMESTAMP(g) TZ
DATE	O	X	X	X	X	X	X	X	X
TIME	X	O	g >= 6	X	X	X	X	X	X
TIME(f)	X	6 >= f	g >= f	X	X	X	X	X	X
TIME TZ	X	X	X	O	g >= 6	X	X	X	X
TIME(f) TZ	X	X	X	6 >= f	g >= f	X	X	X	X
TIMESTAMP	X	X	X	X	X	O	g >= 6	X	X
TIMESTAMP(f)	X	X	X	X	X	6 >= f	g >= f	X	X
TIMESTAMP	X	X	X	X	X	X	X	O	g >= 6

from ₩ to	DATE	TIME	TIME(g)	TIME TZ	TIME(g) TZ	TIMESTAMP	TIMESTAMP(g)	TIMESTAMP TZ	TIMESTAMP(g) TZ
TZ									
TIMESTAMP(f) TZ	X	X	X	X	X	X	X	6 >= f	g >= f

Table 18-14 Type conversion of INTERVAL YEAR TO MONTH family (If p,q are omitted, then it is 2.)

from ₩ to	YEAR(q)	MONTH(q)	YEAR(q) TO MONTH
YEAR(p)	q >= p	X	X
MONTH(p)	X	q >= p	X
YEAR(p) TO MONTH	X	X	q >= p

Table 18-15 Type conversion of INTERVAL DAY TO TIME family (If p,q are omitted, then it is 2.) (If f,g are omitted, then it is 6.)

from ₩ to	DAY(q)	HOUR(q)	MINUTE(q)	SECOND(q, g)	DAY(q) TO HOUR	DAY(q) TO MINUTE	DAY(q) TO SECOND(g)	HOUR(q) TO MINUTE	HOUR(q) TO SECOND(g)	MINUTE(q) TO SECOND(g)
DAY(p)	q >= p	X	X	X	X	X	X	X	X	X
HOUR(p)	X	q >= p	X	X	X	X	X	X	X	X
MINUTE(p)	X	X	q >= p	X	X	X	X	X	X	X
SECOND(p,f)	X	X	X	q >= p g >= f	X	X	X	X	X	X
DAY(p) TO HOUR	X	X	X	X	q >= p	X	X	X	X	X
DAY(p) TO MINUTE	X	X	X	X	X	q >= p	X	X	X	X
DAY(p) TO SECOND(f)	X	X	X	X	X	X	q >= p g >= f	X	X	X
HOUR(p) TO MINUTE	X	X	X	X	X	X	X	q >= p	X	X
HOUR(p) TO SECOND(f)	X	X	X	X	X	X	X	X	q >= p g >= f	X
MINUTE(p)	X	X	X	X	X	X	X	X	X	q >= p

from ₩ to	DAY (q)	HOUR (q)	MINUTE (q)	SECOND(q, g)	DAY(q) TO HOUR	DAY(q) TO MINUTE	DAY(q) TO SECOND(g)	HOUR(q) TO MINUTE	HOUR(q) TO SECOND (g)	MINUTE(q) TO SECOND (g)
TO SECOND (f)										g >= f

Table 18-16 Conversion of ROWID type

from ₩ to	ROWID
ROWID	O

〈alter identity column specification〉

It alters the identity property of the column.
The column should be an identity column.

- SET GENERATED [ALWAYS | BY DEFAULT]
  - It changes the method of generating the identity column.
  - For more information, refer to **〈identity column specification〉** of **CREATE TABLE** statement.
- 〈alter sequence generator restart option〉
  - It changes NEXT VALUE of the identity column.
  - For more information, refer to **〈alter sequence generator restart option〉** clause of **ALTER SEQUENCE** statement.
- 〈basic sequence generator option〉
  - It changes the property of the identity column.
  - In SQL standard, it is defined to be described in the form of SET 〈basic sequence generator option〉, but it can be omitted.
  - For more information, refer to **ALTER SEQUENCE** statement.

〈drop identity property clause〉

It drops the identity property of the column.
The column should be the identity column.

Description

SET NOT NULL clause requires the time for checking null in proportion to the number of table rows.

The following columns do not allow NULL values. In other words, even if DROP NOT NULL clause is perfor med, NULL is not allowed in the following cases.

- A column which includes NOT NULL constraint
- A column which is included in primary key constraint
- An identity column

The change of the default value using SET DEFAULT clause and the change of the identity property using <alter identity column specification> clause, is applied to INSERT or UPDATE statement which is performe d later.

Examples

The following is an example of setting DEFAULT property to the column.

```
gSQL> ALTER TABLE region ALTER COLUMN r_comment SET DEFAULT 'N/A';
Table altered.
```

The following is an example of dropping DEFAULT property from the column.

```
gSQL> ALTER TABLE region ALTER COLUMN r_comment DROP DEFAULT;
Table altered.
```

The following is an example of setting NOT NULL constraint to the column.

```
gSQL> ALTER TABLE region ALTER COLUMN r_regionkey SET NOT NULL;
Table altered.
```

The following is an example of dropping the NOT NULL constraint from the column.

```
gSQL> ALTER TABLE region ALTER COLUMN r_regionkey DROP NOT NULL;
Table altered.
```

The following is an example of extending the data type size of the column.

```
gSQL> ALTER TABLE region ALTER COLUMN r_comment SET DATA TYPE VARCHAR(512);
Table altered.
```

The following is an example of restarting the next value of the identity column.

```
gSQL> ALTER TABLE region ALTER COLUMN r_regionkey RESTART;
Table altered.
```

The following is an example of dropping the identity property from the column.

```
gSQL> ALTER TABLE region ALTER COLUMN r_regionkey DROP IDENTITY;
Table altered.
```

Compatibility

Table 18-17 The SQL satndards compatibility

Feature ID	Description	Compatibility
F381	Extended schema manipulation	X
F382	Alter column data type	O
F383	Set column not null clause	O
F384	Drop identity property value	O
F385	Drop column generation expression clause	X
F386	Set identity column generation clause	O
S043	Enhanced reference types	X
T174	Identity columns	O
T178	Identity columns: simple restart option	O

For More Information

Refer to the followings.

- **ALTER TABLE**
- **ALTER TABLE name ADD COLUMN**
- **ALTER TABLE name SET UNUSED COLUMN**
- **ALTER TABLE name RENAME COLUMN**

18.53 ALTER TABLE name ALTER CONSTRAINT

Function

It alters the characteristics of the table constraint.

Syntax

```
<alter table constraint definition> ::=
    ALTER TABLE table_name
        ALTER <constraint object> <constraint characteristics>
    ;
<constraint object> ::=
      CONSTRAINT constraint_name
    ¦ PRIMARY KEY
    ¦ UNIQUE ( column_name [, ...] )
<constraint characteristics> ::=
      [ NOT ] DEFERRABLE [ <constraint check time> ]
    ¦ <constraint check time> [ [ NOT ] DEFERRABLE ]
<constraint check time> ::=
      INITIALLY DEFERRED
    ¦ INITIALLY IMMEDIATE
```

Invocation and Access Rules

One of the following privileges is required to perform <alter table constraint definition>.

- (ALTER or CONTROL TABLE) ON TABLE for that table
- (ALTER TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- ALTER ANY TABLE ON DATABASE

Cluster does not support the deferrable constraints.

Syntax Rules and Parameters

table_name

It is the table name to be altered.
It can define the schema to which the table belongs, such as schema_name.table_name. If schema_name is omitted, the default schema name of the user performing the statement is used.

〈constraint object〉

The constraint to be altered is specified as follows.

- CONSTRAINT constraint_name
  - The constraint name to be altered.
- PRIMARY KEY
  - PRIMARY KEY constraint of the table
- UNIQUE(column [,...])
  - UNIQUE constraint which satisfies the column list.

DEFERRABLE | NOT DEFERRABLE

It alters whether the constraint state is deferrable.

- DEFERRABLE
  - The constraint is altered to be deferrable.
- NOT DEFERRABLE
  - The constraint is altered not to be deferrable.

For more information about the deferrable constraints, refer to **SET CONSTRAINTS**.

INITIALLY IMMEDIATE | INITIALLY DEFERRED

It alters an initial value of the check point for the constraint.

- INITIALLY IMMEDIATE
  - It checks the constraints at the time of DML.
- INITIALLY DEFERRED
  - It checks the constraints at the time of COMMIT.

The constraints defined as NOT DEFERRABLE can not be altered to INITIALLY DEFERRED.

Description

For more information about the deferrable constraints, refer to **SET CONSTRAINTS**.

Example

The following is an example that the constraint t1_uk is set as deferrable and its checking time is set as D EFERRED.

```
gSQL> ALTER TABLE t1 ALTER CONSTRAINT t1_uk DEFERRABLE INITIALLY DEFERRED;
Table altered.
gSQL> COMMIT;
Commit complete.
```

Compatibility

The SQL standard does not define the following clauses.

- ALTER PRIMARY KEY clause
- ALTER UNIQUE(column [,...]) clause

Table 18-18 SQL standard compatibility

Feature ID	Description	Compatibility
F492	Optional table constraint enforcement	X

18.54 ALTER TABLE name ALTER GLOBAL SECONDARY INDEX

Function

It alters the physical attributes of the global secondary index in the table.

Syntax

```
<alter table alter global secondary index storage statement> ::=
    ALTER TABLE table_name ALTER GLOBAL SECONDARY INDEX
      <physical attribute clause>
    ¦ [ STORAGE ( <segment attr clause> [...] ) ]
    ;
<physical attribute clause> ::=
      PCTFREE integer
    ¦ INITRANS integer
    ¦ MAXTRANS integer
<segment attr clause> ::=
      INITIAL <size_clause>
    ¦ NEXT <size_clause>
    ¦ MINSIZE <size_clause>
    ¦ MAXSIZE <size_clause>
<size clause> ::=
      integer [ K ¦ M ¦ G ¦ T ]
```

Invocation and Access Rules

One of the following privileges is required to perform <alter table alter global secondary index storage statement>.

- (ALTER or CONTROL TABLE) ON TABLE for that table
- (ALTER TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- ALTER ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the name of a table in which the index is to be created.
It can define a schema to which the table belongs such as schema_name.table_name.
If schema_name is omitted, the default schema name of the user performing the statement is used.

〈physical attribute clause〉

It defines the physical attribute information of the index.

- PCTFREE integer
  - Definition
    - The reserved space to adjust the frequency of page splits caused by inserting the key in the page.
    - It is applied only when the index bottom-up build.
  - It can use of the value from 0 to 99.
  - If it is omitted, the value set in DEFAULT_INDEX_PCTFREE property is used by default.

- INITRANS integer
  - Definition
    - The initial number of transactions simultaneously accessing the page.
    - If the number of users accessing the index is small, then INITRANS is set low. If the number of users simultaneously accessing the index is big, then INITRANS is set high.
    - If necessary, it is automatically increased to the specified MAXTRANS.
  - It can use the value from 1 to 32.
  - If it is omitted, the default value is 4.

- MAXTRANS integer
  - Definition
    - It specifies the maximum number of transaction simultaneously accessing the page.
  - It can use the value from 1 to 32.
  - If it is omitted, the default value is 8.

〈segment attr clause〉

It specifies the information for the index storage space.

- INITIAL integer
  - Definition

        - It specifies the size of physical storage space which is initially allocated when creating the ind ex.
        - This size is aligned to the EXTENT size of the TABLESPACE to which the table belongs. (e.g. If the EXT size is 8192 bytes, 'INITIAL 100' is actually operated as 8192 bytes.)
        - The size (aligned to the EXTENT size of TABLESPACE) should be equal to or bigger than MIN EXTENTS, or it should be equal to or less than MAXEXTENTS.
    - The minimum value is 1, and the maximum value depends on the system environment.
    - If it is omitted, the default value is one EXTENT size of TABLESPACE to which the table belongs.

- NEXT integer
    - Definition
        - It specifies the physical space size to be allocated when adding the space to the table.
        - This size is aligned to the EXTENT size of the TABLESPACE to which the table belongs. (e.g. If the EXT size is 8192 bytes, 'NEXT 100' is actually operated as 8192 bytes.)
        - NEXT operates as follows, depending on the remaining space size of the index available curr ently. (Obtained by subtracting the amount of currently used space from the MAXEXTENTS si ze)

      - If the remaining space size is 0, then it can not extend the space.

      - If the remaining space size is bigger than 0, but smaller than NEXT, then it allocates the space as big as the remaining space.

      - If the remaining space size is bigger than NEXT, then it allocates the space as big as the NEXT.
    - The minimum value is 1 and the maximum value depends on the system environment.
    - If it is omitted, the default value is one EXTENT size of TABLESPACE to which the index belongs.

- MINSIZE integer
    - Definition
        - It is the minimum space size of the index.
        - The value should be equal to or smaller MAXSIZE.
    - This size is aligned to the EXTENT size of the TABLESPACE to which the index belongs.
    - The minimum value is 1 and the maximum value depends on the system environment.
    - If it is smaller than the size of two EXTENT, it is specified to the size of two EXTENT.
    - If it is omitted, the default value is the size of two EXTENT.

- MAXSIZE integer
    - Definition
        - It is the maximum space size of the index.
        - The value should be equal to or bigger than MINSIZE.
    - This size is aligned to the EXTENT size of the TABLESPACE to which the index belongs.
    - The minimum value is 1 and the maximum value depends on the system environment
    - If it is smaller than the size of two EXTENT, it is specified to the size of two EXTENT.
    - If it is omitted, the default value is 32 terabytes (35,184,372,088,832).
    - Even though the value is set to over 32 terabytes, it is adjusted and set to 32 terabytes.
    - If the newly allocated space is smaller than the already allocated space, then an error occurs.

〈size clause〉

It specifies the file size in byte. (If it is omitted, the default unit is bytes.)

- K: Kilobytes
- M: Megabytes
- G: Gigabytes
- T: Terabytes

Description

A global secondary index is required to enquire a non-deterministic query.

Examples

It alters the maximum available size to be used by a global secondary index in the table T1 to 100 MBytes.

```
gSQL> ALTER TABLE T1 ALTER GLOBAL SECONDARY INDEX STORAGE( MAXSIZE 100M );
Table altered.
gSQL> COMMIT;
Commit complete.
```

It alters the INITRANS value and the MAXTRANS value to 2 and 4 each which are to be used by a global s econdary index in the table T1

```
gSQL> ALTER TABLE T1 ALTER GLOBAL SECONDARY INDEX INITRANS 2 MAXTRANS 4;
Table altered.
gSQL> COMMIT;
Commit complete.
```

Compatibility

The SQL standard does not define the concepts of the global secondary index.

For More Information

Refer to the followings.

- **ALTER TABLE name ADD GLOBAL SECONDARY INDEX**
- **ALTER TABLE name DROP GLOBAL SECONDARY INDEX**

18.55 ALTER TABLE name ALTER GLOBAL SECONDARY INDEX COALESCE

Function

It drops the fragmentation of the global secondary index.

Syntax

```
<global secondary index coalesce statement> ::=
    ALTER TABLE table_name ALTER GLOBAL SECONDARY INDEX COALESCE
    ;
```

Invocation and Access Rules

The user should satisfy the following conditions to perform <global secondary index coalesce.

- At least one of the following privileges for a table in which the index is to be rebuilt is required.
  - (ALTER or CONTROL TABLE) ON TABLE for that table
  - (ALTER TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs.
  - ALTER ANY TABLE ON DATABASE

- At least one of the following privileges for a tablespace in which the index is to be created is required.
  - CREATE OBJECT ON TABLESPACE for that tablespace
  - USAGE TABLESPACE ON DATABASE

Syntax Rules and Parameters

table_name

It is the name of the target table.
It can define the schema to which the table belongs, such as schema_name.table_name. If schema_name is omitted, the default schema name of the user performing the statement is used.

Description

- It sequentially scans leaf pages and coalesces them when it is allowed to do so, then returns the delet ed pages to the segment.
- It can solve the fragmentation problem of leaf pages which occurred due to UPDATE/ DELETE.
- It is operated only when the adjacent leaf pages are allowed to coalesce, so if the fragmentation level is low, then it may not be effective.
- If the fragmentation level of the index is high, then the processing time may take longer than INDEX REBUILD.

Table 18-19 Comparing to INDEX REBUILD

	INDEX REBUILD	INDEX COALESCE
Altering index attributes	Possible	Impossible
Moving tablespace	Possible	Impossible
Locking table	Required	Not required
Additional space for execution	Required	Not required
Decreasing tree height	Possible	Impossible

Examples

It drops the fragmentation of the global secondary index in table T1.

```
gSQL> ALTER TABLE T1 ALTER GLOBAL SECONDARY INDEX COALESCE;
Table altered.
```

Compatibility

The SQL standard does not define the concepts of the global secondary index.

For More Information

Refer to **ALTER TABLE name ALTER GLOBAL SECONDARY INDEX REBUILD**.

18.56 ALTER TABLE name ALTER GLOBAL SECONDARY INDEX REBUILD

Function

It rebuilds a global secondary index

Syntax

```
<global secondary index rebuild statement> ::=
    ALTER TABLE table_name ALTER GLOBAL SECONDARY INDEX REBUILD
        [ ONLINE ¦ OFFLINE ]
        [ <index attributes> [...] ]
        [ TABLESPACE tablespace_name ]
    ;
<index attributes> ::=
      <physical attribute clause>
    ¦ STORAGE ( <segment attr clause> [...] )
    ¦ <parallel clause>
<physical attribute clause> ::=
      PCTFREE integer
    ¦ INITRANS integer
    ¦ MAXTRANS integer
<segment attr clause> ::=
      INITIAL <size_clause>
    ¦ NEXT <size_clause>
    ¦ MINSIZE <size_clause>
    ¦ MAXSIZE <size_clause>
<size clause> ::=
      integer [ K ¦ M ¦ G ¦ T ]
<parallel clause> ::=
      NOPARALLEL
    ¦ PARALLEL [ integer ]
```

Invocation and Access Rules

The user should satisfy the following conditions to perform <global secondary index rebuild statement>.

- One of the following privileges is required for the table on which the index is to be rebuilt.
  - (ALTER or CONTROL TABLE) ON TABLE for the table
  - (ALTER TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - ALTER ANY TABLE ON DATABASE

- One of the following privileges is required for the tablespace on which the index is to be rebuilt.
  - CREATE OBJECT ON TABLESPACE for the tablespace
  - USAGE TABLESPACE ON DATABASE

Syntax Rules and Parameters

table_name

It is the table name on which the index is to be rebuilt.
It can define the schema to which the table belongs, such as schema_name.table_name. If schema_name is omitted, the default schema name of the user performing the statement is used.

[ONLINE | OFFLINE]

It determines whether to allow DML on the table when rebuilding the index.

- ONLINE
  - It allows INSERT, UPDATE, and DELETE.
- OFFLINE
  - It does not allow INSERT, UPDATE, DELETE.
- When it is omitted, the default value is ONLINE.

<physical attribute clause>

It defines the physical attribute information of an index.

- PCTFREE integer
  - Definition
    - It is the reserved space for adjusting the page split frequency caused by the key inserted in th e page.

- It can use the value from 0 to 99.
- If it is omitted, the default value is the value set in the existing index.

- INITRANS integer
  - Definition
    - It specifies the initial number of transactions which can simultaneously access the page.
    - If the number of users who access the index is small, INITRANS is set to low, and if the numb er of users who simultaneously access the index is big, INITRANS is set to high.
    - If necessary, it is automatically increased to the specified MAXTRANS.
  - It can use the value from 1 to 32.
  - If it is omitted, the default value is the value set in the existing index.

- MAXTRANS integer
  - Definition
    - It specifies the maximum number of transactions which can simultaneously access the page.
  - It can use the value from 1 to 32.
  - If it is omitted, the default value is the value set in the existing index.

〈segment attr clause〉

It specifies the information for the index storage space.

- INITIAL integer
  - Definition
    - It specifies the size of physical storage space which is initially allocated when creating the ind ex.
    - This size is aligned to the EXTENT size of the TABLESPACE to which the table belongs. (e.g. If the EXT size is 8192 bytes, 'INITIAL 100' is actually operated as 8192 bytes.)
    - The size (aligned to the EXTENT size of TABLESPACE) should be equal to or bigger than MIN EXTENTS, or it should be equal to or less than MAXEXTENTS.
  - The minimum value is 1, and the maximum value depends on the system environment.
  - If it is omitted, the default value is the value set in the existing index.

- NEXT integer
  - Definition
    - It specifies the physical space size to be allocated when adding the space to the table.
    - This size is aligned to the EXTENT size of the TABLESPACE to which the table belongs. (e.g. If the EXT size is 8192 bytes, 'NEXT 100' is actually operated as 8192 bytes.)
    - NEXT operates as follows, depending on the remaining space size of the index available curr ently. (Obtained by subtracting the amount of currently used space from the MAXEXTENTS si ze)

    - If the remaining space size is 0, then it can not extend the space.
    - If the remaining space size is bigger than 0, but smaller than NEXT, then it allocates the

space as big as the remaining space.
- If the remaining space size is bigger than NEXT, then it allocates the space as big as the NEXT.
  - The minimum value is 1 and the maximum value depends on the system environment.
  - If it is omitted, the default value is the value set in the existing index.

- MINSIZE integer
  - Definition
    - It is the minimum space size of the index.
    - The value should be equal to or smaller MAXSIZE.
  - This size is aligned to the EXTENT size of the TABLESPACE to which the index belongs.
  - The minimum value is 1 and the maximum value depends on the system environment.
  - If it is smaller than the size of two EXTENT, it is specified to the size of two EXTENT.
  - If it is omitted, the default value is the value set in the existing index.

- MAXSIZE integer
  - Definition
    - It is the maximum space size of the index.
    - The value should be equal to or bigger than MINSIZE.
  - This size is aligned to the EXTENT size of the TABLESPACE to which the index belongs.
  - The minimum value is 1 and the maximum value depends on the system environment.
  - If it is omitted, the default value is the value set in the existing index.

〈size clause〉

It specifies the file size in bytes. (If the unit is omitted, the default value is bytes.)

- K: Kilobytes
- M: Megabytes
- G: Gigabytes
- T: Terabytes

NOPARALLEL | PARALLEL [integer]

It specifies the number of threads to be used when rebuilding an index.

- NOPARALLEL
  - It does not rebuild an index in parallel.
- PARALLEL [integer]
  - It rebuilds an index in parallel.
  - If an integer is omitted or set as 0, then it follows INDEX_BUILD_PARALLEL_FACTOR property.
  - The minimum value of an integer is 0 and the maximum value is 64.
  - If the integer or the property value is 0, then the system determines the optimal value.

- If it is omitted, the default value is NOPARALLEL.

TABLESPACE tablespace_name

It specifies the name of the tablespace in which the index is to be rebuilt.

- When it specifies tablespace_name
  - if tablespace_name is data tablespace, then it is rebuilt as a LOGGING index.
  - if tablespace_name is temporary tablespace or nologging tablespace, then it is rebuilt as a NOLOGGING index.
- When TABLESPACE clause is omitted, then it is set to the tablespace of the existing index.

Description

- Dropping the index fragmentation
  - The fragmentation may occur on the index page, when update DML is frequently performed in the index. If the tree becomes too big comparing to the valid data, then the index volume becomes larger and the performance is degraded. In this case, rebuilding the index can solve the index fragmentation issue so that the index volume is reduced and the index performance is recovered.
- Altering the tablespace in the index
  - The tablespace in the previously created index can be altered.
  - However, LOGGING should be set properly according to whether the tablespace is TEMPORARY or not.
- Altering LOGGING setting in the index
  - The LOGGING setting in the previously created index can be altered by using TABLESPACE option.
  - The data tablespace should be set in TABLESPACE option to switch to the LOGGING index.
  - The temporary tablespace or the nologging tablespace should be set in TABLESPACE option to switch to the NOLOGGING index.

Examples

Rebuild the global secondary index in the table T1.

```
gSQL> ALTER TABLE T1 ALTER GLOBAL SECONDARY INDEX REBUILD;
```

Alter the tablespace and logging settings of the global secondary index in the table T1.

```
gSQL> ALTER TABLE T1 ALTER GLOBAL SECONDARY INDEX REBUILD TABLESPACE MEM_DATA_TBS;
gSQL> ALTER TABLE T1 ALTER GLOBAL SECONDARY INDEX REBUILD TABLESPACE MEM_TEMP_TBS;
```

Compatibility

The SQL standard does not cover the concepts of the global secondary index.

For More Information

Refer to the followings.

- ALTER TABLE name ADD GLOBAL SECONDARY INDEX
- ALTER TABLE name DROP GLOBAL SECONDARY INDEX
- ALTER INDEX name REBUILD

18.57 ALTER TABLE name DROP CONSTRAINT

Function

It drops a table constraint.

Syntax

```
<drop table constraint definition> ::=
    ALTER TABLE table_name
        DROP <constraint object>
        [ <drop behavior> ]
    ;
<constraint object> ::=
      CONSTRAINT constraint_name
    ¦ PRIMARY KEY
    ¦ UNIQUE ( column_name [, ...] )
<drop behavior> ::=
      RESTRICT
    ¦ CASCADE
    ¦ CASCADE CONSTRAINTS
```

Invocation and Access Rules

One of the following privileges is required to perform <drop table constraint definition>.

- The owner of that constraint
- (ALTER or CONTROL TABLE) ON TABLE for that table
- (ALTER TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- ALTER ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the table name to be altered.
It can define the schema to which the table belongs, such as schema_name.table_name. If schema_name is omitted, the default schema name of the user performing the statement is used.

CONSTRAINT constraint_name

It is the constraint name to be dropped.

PRIMARY KEY

It is the primary key constraint for the table.

UNIQUE(column_name [, ...])

It is the unique constraint for the columns.

〈drop behavior〉

When it is omitted, the default value is RESTRICT.
Currently, RESTRICT/CASCADE is operated in the same way.

Description

〈drop column not null clause〉 of **ALTER TABLE name ALTER COLUMN** is used to drop NOT NULL constrai nt without using the constraint name.

Examples

The following is an example of dropping a primary key constraint from the table.

```
gSQL> ALTER TABLE t1 DROP PRIMARY KEY;
Table altered.
```

The following is an example of dropping the table constraint by specifying the constraint name.

```
gSQL> ALTER TABLE t1 DROP CONSTRAINT t1_pk;
Table altered.
```

Compatibility

The SQL standard does not define the following clauses.

- DROP PRIMARY KEY
- DROP UNIQUE (column_name [, ...])
- CASCADE CONSTRAINTS

Table 18-20 SQL standard compatibility

Feature ID	Description	Compatibility
F381	Extended schema manipulation	O

For More Information

Refer to the followings.

- **ALTER TABLE**
- **ALTER TABLE name ADD CONSTRAINT**
- **DROP INDEX**

18.58 ALTER TABLE name DROP GLOBAL SECONDARY INDEX

Function

It drops a global secondary index from the table.

Syntax

```
<alter table drop global secondary index definition> ::=
    ALTER TABLE table_name
        DROP GLOBAL SECONDARY INDEX
    ;
```

Invocation and Access Rules

<alter table drop global secondary index definition> statement can be defined in a cluster system, and the user should satisfy the following conditions.

- The following privilege for the table from which the index is to be dropped is required
  - (ALTER or CONTROL TABLE) ON TABLE for that table
  - (ALTER TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - ALTER ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the name of a table from which the index is to be dropped.

Description

A global secondary index is required to enquire a non-deterministic query.

Examples

It drops a global secondary index from the table T1.

```
gSQL> ALTER TABLE T1 DROP GLOBAL SECONDARY INDEX;
Table altered.
gSQL> COMMIT;
Commit complete.
```

Compatibility

The SQL standard does not define the concepts of the global secondary index.

For More Information

Refer to the followings.

- ALTER TABLE name ADD GLOBAL SECONDARY INDEX
- ALTER TABLE name ALTER GLOBAL SECONDARY INDEX

18.59 ALTER TABLE name DROP OFFLINE SEGMENTS

Function

It drops segments of offline shards.

Syntax

```
<alter table drop offline segments statement> ::=
    ALTER TABLE table_name
        DROP OFFLINE SEGMENTS
;
```

Invocation and Access Rules

It can be performed in the cluster system.

One of the following privilege is required to perform <alter table drop offline segments statement>.

- (ALTER or CONTROL TABLE) ON TABLE for the table
- (ALTER TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- ALTER ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the table name.
It can define the schema to which the table belongs, such as schema_name.table_name. If schema_name is omitted, the default schema name of the user performing the statement is used.

Description

It drops segments of offline shards.

<alter table drop offline segments statement> can be performed even when an inactive cluster member exists.

If it can not satisfy the following conditions, then it fails.

- At least one member of the cluster system should have the online replica of the cloned table to drop the segments of the cloned table.
- At least one member per group should have the online replica of the sharded table to drop the segments of the sharded table.

For example, if all replicas in cluster group G3 of the sharded table t1 are offline, then the following error occurs.

```
gSQL> ALTER TABLE t1 DROP OFFLINE SEGMENTS;
ERR-42000(16361): sharded table "PUBLIC"."T1" must be accessible to at least one member of
group 'G3'
```

Use **<alter database drop offline segments statement>** to perform it for all tables.

Example

The following is an example of performing <alter table drop offline segments statement> for the table T1.

```
gSQL> ALTER TABLE t1 DROP OFFLINE SEGMENTS;
Table altered.
```

Compatibility

The SQL standard does not define the concepts of the cluster.

For More Information

Refer to **ALTER DATABASE DROP OFFLINE SEGMENTS**.

18.60 ALTER TABLE name DROP SUPPLEMENTAL LOG

Function

It sets not to leave the primary key information on the redo log when changing the data in the table.

Syntax

```
<add table supplemental log statement> ::=
    ALTER TABLE table_name
        DROP SUPPLEMENTAL LOG DATA ( PRIMARY KEY ) COLUMNS
    ;
```

Invocation and Access Rules

One of the following privileges is required to perform <drop table supplemental log statement>.

- (ALTER or CONTROL TABLE) ON TABLE for that table
- (ALTER TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- ALTER ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the table name to be altered.
It can define the schema to which the table belongs, such as schema_name.table_name. If schema_name is omitted, the default schema name of the user performing the statement is used.
It should have been set by using **ALTER TABLE name ADD SUPPLEMENTAL LOG** statement.

Description

For more information, refer to the rules for each syntax.

Example

The following is an example of setting not to leave the primary key information on the redo log when cha nging the data in the table.

```
gSQL> ALTER TABLE t1 DROP SUPPLEMENTAL LOG DATA ( PRIMARY KEY ) COLUMNS;
Table altered.
```

Compatibility

The SQL standard does not cover <drop table supplemental log statement>.

18.61 ALTER TABLE name MERGE SHARDS

Function

It merges specific shards in a table in a cluster environment, and rebalances them.

Syntax

```
<alter table merge shards statement> ::=
    ALTER TABLE table_name MERGE SHARDS <source shard list>
       INTO dest_shard_name [ <dest shard placement> ]
    ;
<source shard list> ::=
    source_shard_name [, ...]
  | start_shard_name TO end_shard_name
<dest shard placement> ::=
    AT CLUSTER GROUP dest_group_name
```

Invocation and Access Rules

It can be performed in the cluster system.

One of the following privilege is required to perform <alter table merge shards statement>.

- (ALTER or CONTROL TABLE) ON TABLE for that table
- (ALTER TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- ALTER ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the name of a table.
It can define a schema to which the table belongs such as schema_name.table_name. If schema_name is

omitted, the default schema name of the user performing the statement is used.
The statement can be performed only when the table is a cluster-specific, and a list shard or a range shar d.

〈source shard list〉

It is the list of original shards to be merged.
The shard specified by a list should exist in the table.

source_shard_name

It is the name of an original shard to be merged.
If the shard does not exist in the table, then the statement can not be performed.

start_shard_name

It is the name of the first shard in the range of merging.
It is used only in a range shard.

end_shard_name

It is the name of the last shard in the range of merging.
It is used only in a range shard.

dest_shard_name

It is the name of a target shard.

〈dest shard placement〉

It is the name of a cluster group in which the target shard is to be placed.
If the corresponding clause is omitted, *dest_shard_name* should be included in 〈source shard list〉.

Description

It merges specific shards in a specific table, then places them in an arbitrary cluster group.

- It can not be performed in a standalone database.

- It can not be performed in a hash sharded table nor in a cloned table.
- It can not be performed in a table created as cluster wide.
- DML can not be performed for source shards while merging is in progress.
- If it is a range shard, the beginning and ending original shards to be merged can be defined.
- Original shards to be merged can be listed in a range shard or a list shard.
  In this case, shards listed in a range shard should be the neighboring shard.

The following is an error which occurred when merging shards which are not neighboring in a range shar ded table in a way of listing.

```
CREATE TABLE t1( i1 INTEGER )
    SHARDING BY RANGE (i1)
    SHARD shard1 VALUES LESS THAN ( 200 )      AT CLUSTER GROUP G1,
    SHARD shard2 VALUES LESS THAN ( 400 )      AT CLUSTER GROUP G2,
    SHARD shard3 VALUES LESS THAN ( MAXVALUE ) AT CLUSTER GROUP G3
;
Table created.
ALTER TABLE t1 MERGE SHARDS shard1, shard3 INTO shard4 AT CLUSTER GROUP G2;
ERR-42000(16488): shards being merged are not adjacent :
ALTER TABLE t1 MERGE SHARDS shard1, shard3 INTO shard4 AT CLUSTER GROUP G2
                                    *
ERROR at line 1:
```

Examples

The following is an example of merging shards in a way of listing.

```
gSQL> ALTER TABLE t1 MERGE SHARDS shard1, shard2, shard3 INTO shard4 AT CLUSTER GROUP G2;
Table altered
```

The following is an example of merging shards in a way of ranging.

```
gSQL> ALTER TABLE t1 MERGE SHARDS shard1 TO shard3 INTO shard4 AT CLUSTER GROUP G2;
Table altered
```

Compatibility

The SQL standard does not define the concepts of the cluster.

For More Information

Refer to the followings.

- ALTER TABLE name MOVE SHARD
- ALTER TABLE name SPLIT SHARD

18.62 ALTER TABLE name MOVE SHARD

Function

It rebalances a specific shard of a table, or the entire shard in a specific cluster group to a specific cluster group.

Syntax

```
<alter table move shard statement> ::=
    ALTER TABLE table_name MOVE SHARD
        { shard_name_list ¦ FROM CLUSTER GROUP src_cluster_group }
        TO CLUSTER GROUP dest_cluster_group
       [ ONLINE ¦ OFFLINE ]
       [ <shard divisor> ]
       [ <parallel clause> ]
    ;
<shard divisor> ::=
    SHARD DIVISOR integer
<parallel clause> ::=
    NOPARALLEL
  ¦ PARALLEL [ integer ]
```

Invocation and Access Rules

It can be performed in a cluster system.

One of the following privileges is required to perform <alter table move shard statement>.

- (ALTER or CONTROL TABLE) ON TABLE for that table
- (ALTER TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- ALTER ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the table name.
It can define a schema to which the table belongs such as schema_name.table_name.
If schema_name is omitted, the default schema name of the user performing the statement is used.
The statement can be performed only when that table is a cluster group specific table.

shard_name_list

It is the shard name list to be rebalanced.
If the shard does not exist in that table, then the statement can not be performed.

src_cluster_group

It is the name of a specific cluster group to be rebalanced.

dest_cluster_group

It is the name of a target cluster group on which the shard of the table is to be rebalanced.
If the cluster group specified by the shard of that table already exists then the statement can not be perfo rmed.

[ONLINE | OFFLINE]

It determines whether to allow DML when rebalancing table shard.

- ONLINE
  - It allows INSERT, UPDATE, and DELETE.
- OFFLINE
  - It does not allow INSERT, UPDATE, DELETE.
- When it is omitted, the default value is ONLINE.

〈shard divisor〉

It specifies the number of shard's partitions.

- It divides the shard as many as the number of partitions, then rebalances them in the remote server.

- The minimum value of an integer is 0 and the maximum value is 1000.
- If it is omitted, then it follows REBALANCE_SHARD_DIVISOR property.
- If the integer is smaller than the parallel integer, then it is revised to the same value as the parallel int eger.

〈parallel clause〉

It specifies the number of threads to use when rebalancing the table.

- NOPARALLEL
  - It does not rebalance tables in parallel.
- PARALLEL [integer]
  - It rebalances tables in parallel.
  - The minimum value of an integer is 0 and the maximum value is 64.
  - If the integer is omitted, then it is 0.
  - If the integer is 0, then the system determines the optimal value.

Description

It rebalances a specific shard of the table from a specific cluster group to another cluster group.

To drop a specific cluster group, rebalance the shard of the table then perform the **DROP CLUSTER GROU P** statement.

To move shards of all tables from a specific cluster group to another cluster group, then perform the ALT ER DATABASE MOVE SHARD FROM CLUSTER GROUP TO CLUSTER GROUP statement.

If it is a CLONED table or a CLUSTER WIDE table, then an error occurs and it fails.

Examples

The following is an example of executing the 〈alter table move shard statement〉 statement.

```
gSQL> ALTER TABLE t1 MOVE SHARD shard1, shard2 TO CLUSTER GROUP g3;
Table altered.
gSQL> ALTER TABLE t1 MOVE SHARD FROM CLUSTER GROUP g1 TO CLUSTER GROUP g3;
Table altered.
```

The following is an example of which a CLONED table and a CLUSTER WIDE table fails to move shard.

```
gSQL> CREATE TABLE T1 ( C1 INTEGER ) SHARDING BY RANGE (C1)
         AT CLUSTER WIDE
         SHARD s1 VALUES LESS THAN (10),
         SHARD s2 VALUES LESS THAN (MAXVALUE);
Table created.
gSQL> ALTER TABLE T1 MOVE SHARD s1 TO CLUSTER GROUP g2;
ERR-42000(16440): cannot execute on cluster wide sharded tables
gSQL> CREATE TABLE T2 ( C1 INTEGER ) CLONED AT CLUSTER GROUP g1, g2;
Table created.
gSQL> ALTER TABLE T2 MOVE SHARD FROM CLUSTER GROUP g1 TO CLUSTER GROUP g3;
ERR-42000(16437): cannot execute on cloned tables
```

Compatibility

The SQL standard does not define the concepts of the cluster.

For More Information

Refer to **ALTER DATABASE MOVE SHARD**.

18.63 ALTER TABLE name READ { ONLY | WRITE }

Function

It sets READ { ONLY | WRITE } in a table.

Syntax

```
<alter table read { only ¦ write } statement> :==
     ALTER TABLE table_name
         READ { ONLY ¦ WRITE }
     ;
```

Invocation and Access Rules

One of the following privileges is required to perform <alter table read { only | write } statement>.

- (ALTER or CONTROL TABLE) ON TABLE for that table
- (ALTER TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- ALTER ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the table name.
It can define a schema to which the table belongs such as schema_name.table_name.
If schema_name is omitted, the default schema name of the user performing the statement is used.

Description

It sets table property to READ { ONLY | WRITE }.

If it is set to READ ONLY, neither SELECT .. FOR UPDATE statement, nor DML/ DDL statement which updates table data can be used. However, DDL statement which does not update the table data is allowed.

Disallowed SQL statements when it is set to READ ONLY

- INSERT, UPDATE, DELETE
- TRUNCATE
- SELECT .. FOR UPDATE
- ALTER TABLE RENAME/DROP COLUMN
- ALTER TABLE SET COLUMN UNUSED

Allowed SQL statements when it is set to READ ONLY

- SELECT
- CREATE/ALTER/DROP INDEX
- ALTER TABLE ADD/ALTER COLUMN
- ALTER TABLE ADD/ALTER/RENAME/DROP CONSTRAINT
- ALTER TABLE for physical property changes
- ALTER TABLE DROP UNUSED COLUMNS
- ALTER TABLE RENAME TO
- DROP TABLE
- ALTER TABLE ADD/DROP SUPPLEMENTAL LOG
- LOCK TABLE

Examples

The following is an example of executing <alter table read { only | write } statement>.

```
gSQL> ALTER TABLE t1 READ ONLY;
Table altered.
gSQL> ALTER TABLE t1 READ WRITE;
Table altered.
```

Compatibility

The SQL standard does not define <alter table read { only | write } statement>.

For More Information

Refer to **ALTER TABLE**.

18.64 ALTER TABLE name REBALANCE

Function

It rebalances the shard in a table.

Syntax

```
<alter table rebalance statement> ::=
    ALTER TABLE table_name REBALANCE
       [ ONLINE ¦ OFFLINE ]
       [ <shard divisor> ]
       [ <parallel clause> ]
    ;
<shard divisor> ::=
    SHARD DIVISOR integer
<parallel clause> ::=
    NOPARALLEL
  ¦ PARALLEL [ integer ]
```

Invocation and Access Rules

It can be performed in a cluster system.

One of the following privileges is required to perform <alter table rebalance statement>.

- (ALTER or CONTROL TABLE) ON TABLE for that table
- (ALTER TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- ALTER ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the table name.
It can define the schema to which the table belongs, such as schema_name.table_name. If schema_name is omitted, the default schema name of the user performing the statement is used.

[ONLINE | OFFLINE]

It determines whether to allow DML when rebalancing table shard.

- ONLINE
  - It allows INSERT, UPDATE, and DELETE.
- OFFLINE
  - It does not allow INSERT, UPDATE, DELETE.
- When it is omitted, the default value is ONLINE.

〈shard divisor〉

It specifies the number of shard's partitions.

- It divides the shard as many as the number of partitions, then rebalances them in the remote server.
- The minimum value of an integer is 0 and the maximum value is 1000.
- If it is omitted, then it follows REBALANCE_SHARD_DIVISOR property.
- If the integer is smaller than the parallel integer, then it is revised to the same value as the parallel int eger.

〈parallel clause〉

It specifies the number of threads to use when rebalancing the table.

- NOPARALLEL
  - It does not rebalance tables in parallel.
- PARALLEL [integer]
  - It rebalances tables in parallel.
  - The minimum value of an integer is 0 and the maximum value is 64.
  - If the integer is omitted, then it is 0.
  - If the integer is 0, then the system determines the optimal value.

Description

It does not rebalance shards in a table when adding a cluster member or a cluster group by using the foll owing statements.

- CREATE CLUSTER GROUP
- ALTER CLUSTER GROUP name ADD MEMBER

To rebalance the shards of a table in the added cluster group and the cluster member, perform the <alter table rebalance statement> statement. The operation succeeds without a separate rebalancing if the shar d of the table is already rebalanced.

To rebalance shards in all tables, perform the **ALTER DATABASE REBALANCE** statement.

Examples

The following is an example of executing the <alter table rebalance statement> statement.

```
gSQL> ALTER TABLE t1 REBALANCE;
Table altered.
```

Compatibility

The SQL standard does not define the concepts of the cluster.

18.65 ALTER TABLE name REBALANCE EXCLUDE CLUSTER GROUP cluster_group_list

Function

It rebalances the shard of the table not to include a shard in a specific cluster group.

Syntax

```
<alter table rebalance exclude cluster group statement> ::=
    ALTER TABLE table_name REBALANCE
        EXCLUDE CLUSTER GROUP cluster_group_list
       [ ONLINE ¦ OFFLINE ]
       [ <shard divisor> ]
       [ <parallel clause> ]
    ;
<shard divisor> ::=
    SHARD DIVISOR integer
<parallel clause> ::=
    NOPARALLEL
  ¦ PARALLEL [ integer ]
```

Invocation and Access Rules

It can be performed in a cluster system.

One of the following privileges is required to perform <alter table rebalance exclude cluster group statem ent>.

- (ALTER or CONTROL TABLE) ON TABLE for that table
- (ALTER TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- ALTER ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the table name.
It can define a schema to which the table belongs such as schema_name.table_name.
If schema_name is omitted, the default schema name of the user performing the statement is used.
The statement can be performed only when that table is a cluster-wide table.

cluster_group_list

It is a list of the cluster group which does not include a shard of a table.
If the cluster group to be excluded from the rebalancing is the entire group, the statement can not be per formed.

[ONLINE | OFFLINE]

It determines whether to allow DML when rebalancing table shard.

- ONLINE
  - It allows INSERT, UPDATE, and DELETE.
- OFFLINE
  - It does not allow INSERT, UPDATE, DELETE.
- When it is omitted, the default value is ONLINE.

〈shard divisor〉

It specifies the number of shard's partitions.

- It divides the shard as many as the number of partitions, then rebalances them in the remote server.
- The minimum value of an integer is 0 and the maximum value is 1000.
- If it is omitted, then it follows REBALANCE_SHARD_DIVISOR property.
- If the integer is smaller than the parallel integer, then it is revised to the same value as the parallel int eger.

〈parallel clause〉

It specifies the number of threads to use when rebalancing the table.

- NOPARALLEL

    - It does not rebalance tables in parallel.
- PARALLEL [integer]
    - It rebalances tables in parallel.
    - The minimum value of an integer is 0 and the maximum value is 64.
    - If the integer is omitted, then it is 0.
    - If the integer is 0, then the system determines the optimal value.

Description

It excludes a specific cluster group and rebalances the shard of the table.
If the shard of the table does not exist in that cluster group, the operation succeeds without a separate re balancing.
It rebalances the shard based on the cluster group in which the shard of the table is located.

To drop a specific cluster group, rebalance the shard of the table and perform **DROP CLUSTER GROUP** st atement.
To rebalance the shard excluding a cluster group from all tables, perform the **ALTER DATABASE REBALA NCE EXCLUDE CLUSTER GROUP** statement.

Examples

The following is an example of executing the <alter table rebalance exclude cluster group statement> stat ement.

```
gSQL> ALTER TABLE t1 REBALANCE EXCLUDE CLUSTER GROUP g3;
Table altered.
```

Compatibility

The SQL standard does not define the concepts of the cluster.

18.66 ALTER TABLE name RENAME COLUMN

Function

It renames the table column.

Syntax

```
<rename column statement> ::=
    ALTER TABLE table_name
        RENAME COLUMN old_column_name TO new_column_name
    ;
```

Invocation and Access Rules

One of the following privileges is required to perform <rename column statement>.

- (ALTER or CONTROL TABLE) ON TABLE for that table
- (ALTER TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- ALTER ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the table name to be altered.
It can define the schema to which the table belongs, such as schema_name.table_name. If schema_name is omitted, the default schema name of the user performing the statement is used.

old_column_name

It is the old column name to be altered.

new_column_name

It is the new column name to be altered.
The same column name should not exist in a table.

Description

Even when the column name is altered it does not require the object change such as index, constraint wh ich is generated based on the previous column.

Example

The following is an example of exchanging the names of two columns, col_1 and col_2.

```
gSQL> ALTER TABLE t1 RENAME COLUMN col_1 TO col_temp;
Table altered.
gSQL> ALTER TABLE t1 RENAME COLUMN col_2 TO col_1;
Table altered.
gSQL> ALTER TABLE t1 RENAME COLUMN col_temp TO col_2;
Table altered.
gSQL> COMMIT;
Commit complete.
```

Compatibility

The SQL standard does not define <rename column statement>.

For More Information

Refer to the followings.

- ALTER TABLE
- ALTER TABLE name ADD COLUMN
- ALTER TABLE name SET UNUSED COLUMN
- ALTER TABLE name ALTER COLUMN

18.67 ALTER TABLE name RENAME CONSTRAINT

Function

It renames the table constraints.

Syntax

```
<rename table constraint statement> ::=
    ALTER TABLE table_name
        RENAME <constraint object> TO new_constraint_name
    ;
<constraint object> ::=
      CONSTRAINT constraint_name
    ¦ PRIMARY KEY
    ¦ UNIQUE ( column_name [, ...] )
```

Invocation and Access Rules

One of the following privileges is required to perform <rename table constraint statement>.

- (ALTER or CONTROL TABLE) ON TABLE for that table
- (ALTER TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- ALTER ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the table name to be altered.
It can define the schema to which the table belongs, such as schema_name.table_name. If schema_name is omitted, the default schema name of the user performing the statement is used.

〈constraint object〉

The existing name of the constraint to be altered is specified as follows.

- CONSTRAINT constraint_name
  - The constraint name to be altered
- PRIMARY KEY
  - PRIMARY KEY constraint of the table
- UNIQUE(column [,...])
  - UNIQUE constraint which satisfies the column list

new_column_name

It is the new name of a constraint to be altered.

Description

The index name which was automatically created with a key constraint such as primary key, unique key is not altered. Use **ALTER INDEX name RENAME TO** statement to rename the index.

Examples

The following is an example of renaming the primary key constraint of the table.

```
gSQL> ALTER TABLE t1 RENAME PRIMARY KEY TO pk_t1;
Table altered.
```

The following is an example of renaming the table constraint by specifying the constraint name.

```
gSQL> ALTER TABLE t1 RENAME CONSTRAINT pk_t1 TO t1_pk;
Table altered.
```

Compatibility

The SQL standard does not define the 〈rename table constraint statement〉 statement.

For More Information

Refer to the followings.

- ALTER TABLE
- ALTER TABLE name ADD CONSTRAINT
- ALTER TABLE name DROP CONSTRAINT
- ALTER TABLE name ALTER CONSTRAINT

18.68 ALTER TABLE name RENAME SHARD

Function

It renames a specific shard of a table in cluster environment.

Syntax

```
<alter table rename shard statement> ::=
    ALTER TABLE table_name
        RENAME SHARD shard_name TO new_shard_name
    ;
```

Invocation and Access Rules

It can be performed in a cluster system.

One of the following privileges is required to perform <alter table rename shard statement>.

- (ALTER or CONTROL TABLE) ON TABLE for that table
- (ALTER TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- ALTER ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the table name to be altered.
It can define a schema to which the table belongs such as schema_name.table_name.
If schema_name is omitted, the default schema name of the user performing the statement is used.

shard_name

It is the existing name of a shard to be altered.
If the shard does not exist in that table, then the statement can not be performed.

new_shard_name

It is the new name of a shard to be altered.
The same shard name should not exist in the table.

Description

It alters the name of a specific shard of a hash, a range, or a list table. This statement can not be perform ed for a cloned table.

Examples

The following is an example of executing <alter table rename shard statement> statement.

```
gSQL> ALTER TABLE t_range RENAME SHARD r_01 TO r_new_01;
Table altered.
gSQL> ALTER TABLE t_list RENAME SHARD l_01 TO l_new_01;
Table altered.
gSQL> ALTER TABLE t_hash RENAME SHARD shard_000000 TO h_new_00;
Table altered.
gSQL> COMMIT;
Commit complete.
```

Compatibility

The SQL standard does not define the concepts of the cluster.

For More Information

Refer to the followings.

- ALTER TABLE
- ALTER TABLE name MOVE SHARD
- ALTER TABLE name SPLIT SHARD
- ALTER TABLE name REBALANCE

18.69 ALTER TABLE name RENAME TO

Function

It renames the table.

Syntax

```
<rename table statement> ::=
    ALTER TABLE table_name
        RENAME TO new_table_name
    ;
```

Invocation and Access Rules

One of the following privileges is required to perform <rename table statement>.

- (ALTER or CONTROL TABLE) ON TABLE for that table
- (ALTER TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- ALTER ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the existing name of the table.
It can define the schema to which the table belongs, such as schema_name.table_name. If schema_name is omitted, the default schema name of the user performing the statement is used.

new_table_name

It is a new name of the table.
The same table name should not exist in the schema.

Description

Even when the table is renamed, the object referring to the table such as index, constraint does not need to be renamed.

Example

The following is an example of exchanging the name of the two tables t1, t2.

```
gSQL> ALTER TABLE t1 RENAME TO t_temp;
Table altered.
gSQL> ALTER TABLE t2 RENAME TO t1;
Table altered.
gSQL> ALTER TABLE t_temp RENAME TO t2;
Table altered.
gSQL> COMMIT;
Commit complete.
```

Compatibility

The SQL standard does not define <rename table statement>.

For More Information

Refer to **ALTER TABLE**.

18.70 ALTER TABLE name SET UNUSED COLUMN

Function

It drops a table column.

Syntax

```
<drop column definition> ::=
    ALTER TABLE table_name <drop column clause>
    ;
<drop column clause> ::=
      SET UNUSED [ COLUMN ] <column_name_list> [ <drop behavior> ]
<column name list> ::=
      column_name
    ¦ ( column_name [, ...] )
<drop behavior> ::=
      RESTRICT
    ¦ CASCADE
    ¦ CASCADE CONSTRAINTS
```

Invocation and Access Rules

One of the following privileges is required to perform <drop column definition>.

- (ALTER or CONTROL TABLE) ON TABLE for that table
- (ALTER TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- ALTER ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the table name to be altered.
It can define the schema to which the table belongs, such as schema_name.table_name. If schema_name is omitted, the default schema name of the user performing the statement is used.

SET UNUSED [COLUMN]

It sets the column not to be used.

column_name_list

One or more column names to be dropped.

- e.g. ALTER TABLE t1 SET UNUSED COLUMN c1
- e.g. ALTER TABLE t1 SET UNUSED COLUMN (c1, c2)

column_name

It is the column name to be dropped.
It also drops the constraints and indexes which use the column.

drop behavior

When it is omitted, the default value is RESTRICT.
Currently, RESTRICT/CASCADE is operated in the same way.

Description

SET UNUSED COLUMN does not delete the data physically, so it ensures consistent performance regardle ss of the number of the rows.

Example

The following is an example of setting the column not to be used.

```
gSQL> ALTER TABLE t1 SET UNUSED COLUMN ( addr );
Table altered.
```

Compatibility

The SQL standard does not define the following clauses.

- SET UNUSED
- CASCADE CONSTRAINTS
- Listing multiple columns

Table 18-21 SQL standard compatibility

Feature ID	Description	Compatibility
F033	ALTER TABLE statement: DROP COLUMN clause	X

For More Information

Refer to the followings.

- **ALTER TABLE**
- **ALTER TABLE name ADD COLUMN**
- **ALTER TABLE name ALTER COLUMN**
- **ALTER TABLE name RENAME COLUMN**

18.71 ALTER TABLE name SPLIT SHARD

Function

It rebalances a specific shard of a table by splitting it in a cluster environment.

Syntax

```
<alter table split shard statement> ::=
    ALTER TABLE table_name SPLIT SHARD source_shard_name
        INTO ( <split shard placement> [, ...] )
    ;
<split shard placement> ::=
    <split shard bound def> AT CLUSTER GROUP dest_group_name
<split shard bound def> ::=
    <split list shard def>
  ¦ <split range shard def>
<split list shard def> :=
    SHARD dest_shard_name VALUES IN ( <split list value clause> )
<split list value clause> :=
    <split list value> [, ...]
<split list value> :=
    constant
  ¦ NULL
<split range shard def> :=
    SHARD dest_shard_name VALUES LESS THAN ( <split range value clause> )
<split range value clause> :=
    <split range value> [, ...]
<split range value> :=
    constant
```

Invocation and Access Rules

It can be performed in a cluster system.

One of the following privileges is required to perform 〈alter table split shard statement〉.

- (ALTER or CONTROL TABLE) ON TABLE for that table
- (ALTER TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- ALTER ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the table name.
It can define a schema to which the table belongs such as schema_name.table_name.
If schema_name is omitted, the default schema name of the user performing the statement is used.
The statement can be performed only when that table is a cluster group specific table, and when the shar d is a list shard or a range shard.

source_shard_name

It is the name of an original shard to be splitted.
If the shard does not exist in that table, the statement can not be performed.

〈split shard placement〉

It defines the target shard to which the original shard is rebalanced by splitting.

〈split shard bound def〉

It defines the bound of a target shard to be splitted.

It can be defined as one of two following bound defs.

- 〈split list shard def〉
- 〈split range shard def〉

〈split list shard def〉

It defines a split shard bound for a list shard.

- dest_shard_name
  - It is the name of a target shard.

- <split list value clause>
  - <split list value> should be an integer.
  - <split list value> can not be NULL.
  - <split list value> can not be DEFAULT.
  - S1 : (1, 11, 21, 31, NULL) SPLIT SHARD S1 INTO (<split list value clause> ..)
    - (O) SHARD S11 VALUES IN (1)
    - (O) SHARD S11 VALUES IN (1, NULL)
    - (O) SHARD S11 VALUES IN (1, 11, 21, 31)
    - (X) SHARD S11 VALUES IN (2)
    - (X) SHARD S11 VALUES IN (DEFAULT)
    - (X) SHARD S11 VALUES IN (1, 11, 21, 31, NULL)

<split range shard def>

It defines a split shard bound for a range shard.

- <split range value clause>
  - <split list value> should be an integer.
  - <split list value> can not be NULL.
  - <split list value> can not be MAXVALUE.
  - S1 : (100, 100), S2 : (50, 50) SPLIT SHARD S1 INTO (<split range value clause> ..)
    - (O) SHARD S11 VALUES IN (50, 100)
    - (O) SHARD S11 VALUES IN (100, 50)
    - (O) SHARD S11 VALUES IN (60, 60)
    - (X) SHARD S11 VALUES IN (50, NULL)
    - (X) SHARD S11 VALUES IN (50, 50)
    - (X) SHARD S11 VALUES IN (100, 100)
    - (X) SHARD S11 VALUES IN (100, 110)
    - (X) SHARD S11 VALUES IN (MAXVALUE, 100)

dest_group_name

It is the name of a cluster group in which the splitted shard is to be rebalanced.

Description

It splits a specific shard of a specific table and rebalances it to a random cluster group.
This is used to distribute records and loads by splitting the shards when records corresponding to a specifi c shard is too much or when a specific group member is overloaded.

Examples

The following is an example of executing the <alter table split shard statement> statement.

```
gSQL> ALTER TABLE t1 SPLIT SHARD shard1 INTO ( SHARD shard11 VALUES IN ( 11 ) AT CLUSTER GROUP
G2 );
Table altered.
gSQL> ALTER TABLE t1 SPLIT SHARD shard1 INTO ( SHARD shard11 VALUES LESS THAN ( 11 ) AT
CLUSTER GROUP G2 );
Table altered.
```

Compatibility

The SQL standard does not define the concepts of the cluster.

For More Information

Refer to the followings.

- **ALTER TABLE name REBALANCE**
- **ALTER TABLE name REBALANCE EXCLUDE CLUSTER GROUP cluster_group_list**
- **ALTER TABLE name MOVE SHARD**
- **ALTER TABLE name MERGE SHARDS**

18.72 ALTER TABLE name STORAGE

Function

It alters physical attributes of a table.

Syntax

```
<alter table physical attribute statement> ::=
    ALTER TABLE table_name
      [ <physical attribute clause> ]
    ¦ [ STORAGE ( <segment attr clause> [...] ) ]
    ;
<physical attribute clause> ::=
      PCTFREE integer
    ¦ PCTUSED integer
    ¦ INITRANS integer
    ¦ MAXTRANS integer
<segment attr clause> ::=
    NEXT <size_clause>
    ¦ MINSIZE <size_clause>
    ¦ MAXSIZE <size_clause>
<size clause> ::=
      integer [ K ¦ M ¦ G ¦ T ]
```

Invocation and Access Rules

One of the following privileges is required to perform <alter table physical attribute statement>.

- (ALTER or CONTROL TABLE) ON TABLE for that table
- (ALTER TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- ALTER ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the table name to be altered.
It can define the schema to which the table belongs, such as schema_name.table_name. If schema_name is omitted, the default schema name of the user performing the statement is used.

〈physical attribute clause〉

It alters the physical attribute of a page which configures the table.
It is not applied to the already allocated page, but is applied to the newly allocated page.
For more information, refer to **〈table physical attribute clause〉** of **CREATE TABLE**.

〈segment attr clause〉

It alters the physical attribute of the extent configuring the segment. It is not applied to the already allocated extent but is applied to the newly allocated extent.

- MAXSIZE integer
  - It alters the space size of the segment which can be allocated.
  - If the newly allocated space is smaller than the already allocated space, then an error occurs.

Description

For more information, refer to the rules for each syntax.

Example

The following is an example of changing the physical attribute of the table.

```
gSQL> ALTER TABLE t1 PCTFREE 10 PCTUSED 40 STORAGE ( NEXT 10M  MAXSIZE  100M );
Table altered.
```

Compatibility

The SQL standard does not define the physical attribute of a table.

For More Information

Refer to **ALTER TABLE**.

18.73 ALTER TABLE name SYNCHRONIZE

Function

It remotely synchronizes shards of the existing table.

Syntax

```
<alter table synchronize statement> ::=
    ALTER TABLE table_name SYNCHRONIZE
       [ ONLINE ¦ OFFLINE ]
       [ <shard divisor> ]
       [ <parallel clause> ]
    ;
<shard divisor> ::=
    SHARD DIVISOR integer
<parallel clause> ::=
    NOPARALLEL
  ¦ PARALLEL [ integer ]
```

Invocation and Access Rules

It can be performed in the cluster system.

One of the following privilege is required to perform <alter table synchronize statement>.

- (ALTER or CONTROL TABLE) ON TABLE for the table
- (ALTER TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- ALTER ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the table name.
It can define the schema to which the table belongs, such as schema_name.table_name. If schema_name is omitted, the default schema name of the user performing the statement is used.

[ONLINE | OFFLINE]

It determines whether to allow DML when synchronizing the table shard.

- ONLINE
  - It allows INSERT, UPDATE, DELETE.
- OFFLINE
  - It does not allow INSERT, UPDATE, DELETE.
- If it is omitted, the default value is ONLINE.

〈shard divisor〉

It specifies the number of shard's partitions.

- It divides the shard as many as the number of partitions, then synchronizes it with the remote server.
- The minimum value of an integer is 0 and the maximum value is 1000.
- If it is omitted, then it follows REBALANCE_SHARD_DIVISOR property.
- If the integer is smaller than the parallel integer, then it is revised to the same value as the parallel int eger.

〈parallel clause〉

It specifies the number of threads to use when synchronizing the table.

- NOPARALLEL
  - It does not synchronize tables in parallel.
- PARALLEL [integer]
  - It synchronize tables in parallel.
  - The minimum value of an integer is 0 and the maximum value is 64.
  - If the integer is omitted, then it is 0.
  - If the integer is 0, then the system determines the optimal value.

Description

The table synchronization synchronizes the existing offline shards to restore the data file matching. Unlike **〈alter table rebalance statement〉**, it can be performed even when an inactive cluster member exists.

If it can not satisfy the following conditions, then it fails.

- At least one member of the cluster system should have the online replica of the cloned table to synch ronize shards of the cloned table.
- At least one member per group should have the online replica of the sharded table to synchronize sh ards of the sharded table.

For example, if all replicas in cluster group G3 of the sharded table t1 are offline, then the following error occurs.

```
gSQL> ALTER TABLE t1 SYNCHRONIZE;
ERR-42000(16546): sharded table "PUBLIC"."T1" must have at least one online replica of group
'G3'
```

Perform **〈alter database synchronize statement〉** to synchronize shards in all tables.

Example

The following is an example of performing〈alter table synchronize statement〉 for the table T1.

```
gSQL> ALTER TABLE t1 SYNCHRONIZE;
Table altered.
```

Compatibility

The SQL standard does not define the concepts of the cluster.

For More Information

Refer to the followings.

- ALTER TABLE

- ALTER TABLE name REBALANCE
- ALTER DATABASE SYNCHRONIZE

18.74 ALTER TABLESPACE

Function

It alters the tablespace definition.

Syntax

```
<alter tablespace statement> ::=
      <rename tablespace statement>
    ¦ <backup tablespace statement>
    ¦ <on/offline tablespace statement>
    ¦ <add file statement>
    ¦ <drop file statement>
    ¦ <rename datafile statement>
    ;
```

Invocation and Access Rules

ALTER TABLESPACE privilege is required to perform <alter tablespace statement>.

Syntax Rules and Parameters

<rename tablespace statement>

It renames the tablespace.
For more information, refer to **ALTER TABLESPACE name RENAME TO** statement.

<backup tablespace statement>

It backs up the tablespace.
For more information, refer to **ALTER TABLESPACE name BACKUP** statement.

〈on-offline tablespace statement〉

It changes all files in the tablespace to the online state or offline state.
For more information, refer to **ALTER TABLESPACE name [ONLINE|OFFLINE]** statement.

〈add file statement〉

It adds a file to the tablespace.
For more information, refer to **ALTER TABLESPACE name ADD [DATAFILE|MEMORY]** statement.

〈drop file statement〉

It drops a file from the tablespace.
For more information, refer to **ALTER TABLESPACE name DROP [DATAFILE|MEMORY]** statement.

〈rename datafile statement〉

It renames the datafile in the data tablespace.
For more information, refer to **ALTER TABLESPACE name RENAME DATAFILE** statement.

Description

Unlike other Data Definition Language (DDL), ALTER TABLESPACE statement is not allowed to ROLLBAC K, and its transaction is automatically committed after executing the statement.

Example

Refer to the examples of each detailed statement.

Compatibility

The SQL standard does not define the concepts of the tablespace.

For More Information

Refer to the followings.

- CREATE TABLESPACE
- DROP TABLESPACE

18.75 ALTER TABLESPACE name ADD [DATAFILE| MEMORY]

Function

It extends the space of the tablespace.

Syntax

```
<add space statement> ::=
    ALTER TABLESPACE tablespace_name ADD <space specification>
        [AT <domain name>]
    ;
<space specification> ::=
      MEMORY <memory clause> [, ...]
    ¦ DATAFILE <add datafile clause> [, ...]
<size clause> ::=
    integer [ K ¦ M ¦ G ¦ T ]
<memory clause>
     'memory_name' { SIZE <size clause> }
<add datafile clause> ::=
     'filename'
        { SIZE <size clause> ¦ REUSE ¦ SIZE <size clause> REUSE }
        [ <autoextend clause> ]
<autoextend clause>
    AUTOEXTEND { ON [ <next size clause> ] [ <max size clause> ] ¦ OFF }
<next size clause>
    NEXT <size clause>
<max size clause>
    MAXSIZE { <size clause> ¦ UNLIMITED }
```

Invocation and Access Rules

ALTER TABLESPACE ON DATABASE privilege is required to perform <add space statement>.

Syntax Rules and Parameters

tablespace_name

It is the tablespace name to be altered.

<file specification>

The following syntax should be used according to the tablespace type.

- Memory data tablespace
  - DATAFILE <add datafile clause>
- Memory temporary tablespace
  - MEMORY <memory clause>

<add datafile clause>

It defines the memory datafile to be added.

- 'filename'
  - It is the file name to store and manage the data.
  - It is the space to store the checkpoint image for the memory data.
  - The filename can be either the new file or existing file.
  - The length of filename should be shorter than 1024 bytes.

- SIZE <size clause>
  - For the new file, the initial size is specified by using SIZE clause.
  - An error occurs if the file exists.
  - The file size can be specified from 1M to 30G.

- REUSE
  - If the file exists, it uses REUSE clause.
  - If the file does not exist, a new file is created.
  - The size of newly created file
    - For the data tablespace, it is determined by MEMORY_DATA_TABLESPACE_SIZE property.

        - For the temporary table space, it is determined by MEMORY_TEMP_TABLESPACE_SIZE property.

- SIZE 〈size clause〉 REUSE
    - If both of SIZE clause and REUSE clause are specified, it is operated as follows based on the presence of the filename.
        - For the new filename, the initial file size is specified by using SIZE clause.
        - For the existing filename, it is adjusted to the SIZE clause value by using the existing file.

〈memory clause〉

- 〈size clause〉
    - It defines the memory to be added.

For more information, refer to **〈memory clause〉** of **CREATE MEMORY TEMPORARY TABLESPACE** statement.

〈autoextend clause〉

It sets the automatic extending property when adding the data file of the disk tablespace. Set the automatic extending property to *ON* or *OFF*. If it is set to *ON*, then it can define the size of the automatic extending and the maximum size of the data file.

〈next size clause〉

It defines the space size to be extended when there is not any space available in the current data file in use.

〈max size clause〉

It defines the maximum size of the data file which can be extended.

〈domain name〉

It is a name of a member or a group for which the statement is performed.
If it is omitted, it is performed for all groups.

Description

For more information, refer to the rules for each syntax.

Example

The following is an example of adding datafile to the tablespace.

```
gSQL> ALTER TABLESPACE space1 ADD DATAFILE 'test_file_a2.dbf' SIZE 10M REUSE;
Tablespace altered.
```

Compatibility

The SQL standard does not define the concepts of the tablespace.

For More Information

Refer to the followings.

- CREATE MEMORY DATA TABLESPACE
- CREATE MEMORY TEMPORARY TABLESPACE
- ALTER TABLESPACE

18.76 ALTER TABLESPACE name BACKUP

Function

It switches the tablespace to backup enabled state and backup disabled state to perform backup.

Syntax

```
<backup tablespace statement> ::=
      <tablespace begin backup statement>
    ¦ <tablespace end backup statement>
    ¦ <tablespace incremental backup statement>
    ;
<tablespace begin backup statement> ::=
    ALTER TABLESPACE tablespace_name BEGIN BACKUP [AT <domain_name>]
    ;
<tablespace end backup statement> ::=
    ALTER TABLESPACE tablespace_name END BACKUP [AT <domain_name>]
    ;
<tablesapce incremental backup statement> ::=
    ALTER TABLESPACE tablespace_name
        BACKUP INCREMENTAL <incremental backup option> [AT <domain_name>]    ;
<incremental backup option> ::=
      LEVEL integer [ CUMULATIVE ¦ DIFFERENTIAL ]
```

Invocation and Access Rules

ALTER TABLESPACE ON DATABASE privilege is required to perform <backup space statement>.

Syntax Rules and Parameters

〈tablespace begin backup statement〉

It sets the tablespace to the backup enabled state.

- The tablespace being used is set to the backup enabled state.
- The backup state of the tablespace such as OFFLINE/ temporary can not be switched.

tablespace_name

It is the tablespace name whose backup state is to be switched.

〈tablespace end backup statement〉

It sets the tablespace to the backup disabled state.

〈tablesapce incremental backup statement〉

It performs the incremental backup of the tablespace.
The database is in OPEN phase and it should be operated in ARCHIVELOG mode.

〈incremental backup option〉

- An 'Integer' can be specified from 0 to 4.
- 'LEVEL 0' can not specify CUMULATIVE or DIFFERENTIAL.
- CUMULATIVE | DIFFERENTIAL
  - CUMULATIVE
    - If 'integer' is n, it backs up all pages which are altered after the most recent backups of LEVE L 0 ~ LEVEL n-1.
  - DIFFERENTIAL
    - If 'integer' is n, it backs up all pages which are altered after the most recent backups of LEVE L 0 ~ LEVEL n.
  - If it is omitted, DIFFERENTIAL is specified by default.

〈domain name〉

It is a name of a member or a group for which the statement is performed.
If it is omitted, it is performed for all groups.

Description

It backs up the datafiles which are created in the tablespace. A full backup of the tablespace begins with BEGIN BACKUP, and copies the datafiles by OS file copy and ends with END BACKUP. The incremental ba ckup file is created in the path set by BACKUP_DIR 1 property using a single statement.

Examples

The following is an example of setting the full backup state to 'ACTIVE' for the tablespace DICTIONARY_ TBS.

```
ALTER TABLESPACE DICTIONARY_TBS BEGIN BACKUP;
```

The following is an example of setting the full backup state to 'INACTIVE' for the tablespace DICTIONARY _TBS.

```
ALTER TABLESPACE DICTIONARY_TBS END BACKUP;
```

The following is an example of generating the LEVEL 0 incremental backup of the tablespace DICTIONAR Y_TBS.

```
ALTER TABLESPACE DICTIONARY_TBS BACKUP INCREMENTAL LEVEL 0;
```

Compatibility

The SQL standard does not define the concepts of the tablespace.

For More Information

Refer to the followings.

- ALTER TABLESPACE
- ALTER TABLESPACE name [ONLINE|OFFLINE]

18.77 ALTER TABLESPACE name DROP [DATAFILE|MEMORY]

Function

It reduces the space of the tablespace.

Syntax

```
<drop space statement> ::=
    ALTER TABLESPACE tablespace_name DROP <file specification>
    [ AT <domain name> ]
    ;
<file specification> ::=
      DATAFILE 'filename'
    | MEMORY 'memory_name'
```

Invocation and Access Rules

ALTER TABLESPACE ON DATABASE privilege is required to perform <drop space statement>.

Syntax Rules and Parameters

tablespace_name

It is the tablespace name to be altered.

<file specification>

The following syntax should be used according to the tablespace type.

- Memory data tablespace
  - DATAFILE 'filename'

- Memory temporary tablespace
  - MEMORY 'memory_name'

The file of OFFLINE tablespace can not be dropped.
The first file of the tablespace can not be dropped.
The data file which has been used once can not be dropped.

〈domain name〉

It is a name of a member or a group for which the statement is performed.
If it is omitted, it is performed for all groups.

Description

For more information, refer to the rules for each syntax.

Example

The following is an example of dropping the file from the tablespace.

```
gSQL> ALTER TABLESPACE space1 DROP DATAFILE 'test_file_f2.dbf';
Tablespace altered.
```

Compatibility

The SQL standard does not define the concepts of the tablespace.

For More Information

Refer to the followings.

- **ALTER TABLESPACE**
- **ALTER TABLESPACE name ADD [DATAFILE|MEMORY]**
- **ALTER TABLESPACE name RENAME DATAFILE**

18.78 ALTER TABLESPACE name [ONLINE|OFFLINE]

Function

It alters the tablespace status.

Syntax

```
<on/off tablespace statement> ::=
    ALTER TABLESPACE tablespace_name { ONLINE ¦ OFFLINE [ NORMAL ¦ IMMEIDATE ] }
    [ AT <domain name> ]
    ;
```

Invocation and Access Rules

ALTER TABLESPACE ON DATABASE privilege is required to perform <on/off tablespace statement>.

Syntax Rules and Parameters

ONLINE

It alters the tablespace status in OFFLINE state to ONLINE state.

OFFLINE NORMAL

It alters the tablespace status in ONLINE state to OFFLINE state.

The media recovery is not required in ONLINE state because the tablespace which was altered to OFFLINE state is in consistent state.

OFFLINE NORMAL is not allowed in MOUNT phase.
(However, if the previous instance is terminated by \SHUTDOWN NORMAL, OFFLINE NORMAL i s allowed.)

OFFLINE IMMEDIATE

It alters the tablespace status in ONLINE state to OFFLINE state.

The media recovery is required in ONLINE state because the tablespace which was altered to OFFLINE stat e is in inconsistent state.

The SYSTEM tablespace can not be altered to OFFLINE state.
OFFLINE IMMEDIATE requires the media recovery, so it can be performed only in ARCHIVELOG mode.

〈domain name〉

It is a name of a member or a group for which the statement is performed.
If it is omitted, it is performed for all groups.

Description

For more information, refer to the rules for each syntax.

Examples

The following is an example of setting the tablespace to OFFLINE.

```
gSQL> ALTER TABLESPACE space1 OFFLINE;
Tablespace altered.
```

The following is an example of that OFFLINE NORMAL for the tablespace fails in MOUNT phase.

```
gSQL> ALTER TABLESPACE space1 OFFLINE;
ERR-42000(16290): OFFLINE NORMAL is only allowed if the database is in OPEN phase :
```

```
ALTER TABLESPACE space1 OFFLINE
                 *
ERROR at line 1:
gSQL> ALTER TABLESPACE space1 OFFLINE NORMAL;
ERR-42000(16290): OFFLINE NORMAL is only allowed if the database is in OPEN phase :
ALTER TABLESPACE space1 OFFLINE NORMAL
                 *
ERROR at line 1:
```

Compatibility

The SQL standard does not define the concepts of the tablespace.

For More Information

Refer to the followings.

- ALTER TABLESPACE
- ALTER TABLESPACE name BACKUP

18.79 ALTER TABLESPACE name RENAME DATAFILE

Function

It renames the datafiles which configures the tablespace.

Syntax

```
<rename datafile statement> ::=
    ALTER TABLESPACE tablespace_name RENAME DATAFILE <filename_list> TO <filename_list>
    ;
    <filename_list> ::=
        'filename' [, ...]
```

Invocation and Access Rules

ALTER TABLESPACE ON DATABASE privilege is required to perform <rename datafile statement>.

ONLINE tablespace file can not be altered when it is in TDS mode and the database is in OPEN phase. (Except for the temporary memory tablespace.)
The file should exist even after the alteration.

Syntax Rules and Parameters

tablespace_name

It is the tablespace name to be altered.

'filename'

The memory temporary tablespace is 'memory_name' and the other kinds of tablespace is 'filename'.

〈domain name〉

It is a name of a member or a group for which the statement is performed.
If it is omitted, it is performed for all groups.

Description

The tablespace status determines whether the operation can be performed.

- OFFLINE: It can be performed in MOUNT or OPEN phase.
- ONLINE: It can be performed only in MOUNT phase.

Example

The following is an example of renaming 'test.dbf' to 'test1.dbf'.

```
gSQL> ALTER TABLESPACE TEST_TBS RENAME DATAFILE 'test.dbf' TO 'test1.dbf';
Tablespace altered.
```

Compatibility

The SQL standard does not define the concepts of the tablespace.

For More Information

Refer to the followings.

- ALTER TABLESPACE
- ALTER TABLESPACE name ADD [DATAFILE|MEMORY]
- ALTER TABLESPACE name DROP [DATAFILE|MEMORY]

18.80 ALTER TABLESPACE name RENAME TO

Function

It renames the tablespace.

Syntax

```
<rename tablespace statement> ::=
    ALTER TABLESPACE tablespace_name RENAME TO <new_tablespace_name>
    ;
```

Invocation and Access Rules

ALTER TABLESPACE ON DATABASE privilege required to perform <rename space statement>.

Syntax Rules and Parameters

tablespace_name

It is a name of the old tablespace.

- The built-in tablespace can not be renamed.
- The OFFLINE tablespace can not be renamed.

new_tablespace_name

It is a name of the new tablespace.

Description

Even when the tablespace is renamed, the table or index which was already created in the existing tables pace does not need to be renamed.

Example

The following is an example of renaming the tablespace.

```
gSQL> ALTER TABLESPACE space1 RENAME TO space2;
Tablespace altered.
```

Compatibility

The SQL standard does not define the concepts of the tablespace.

For More Information

Refer to **ALTER TABLESPACE**.

18.81 ALTER USER

Function

It alters the user definition of the database.

Syntax

```
<alter user statement> ::=
      ALTER USER user_identifier <alter user action>
    ¦ ALTER USER PUBLIC <alter schema path>
    ;
<alter user action> ::=
      <alter password>
    ¦ <alter profile>
    ¦ <password expire>
    ¦ <account lock>
    ¦ <alter default tablespace>
    ¦ <alter temporary tablespace>
    ¦ <alter schema path>
<alter password> ::=
    IDENTIFIED BY new_password [ REPLACE old_password ]
<alter profile> ::=
    PROFILE { profile_name ¦ DEFAULT ¦ NULL }
<password expire> ::=
    PASSWORD EXPIRE
<account lock> ::=
    ACCOUNT { LOCK ¦ UNLOCK }
<alter default tablespace> ::=
    DEFAULT TABLESPACE tablespace_name
<alter temporary tablespace> ::=
    TEMPORARY TABLESPACE tablespace_name
<alter schema path> ::=
    SCHEMA PATH ( { schema_name ¦ CURRENT PATH } [, ...] )
```

Invocation and Access Rules

ALTER USER ON DATABASE privilege is required to perform <alter user statement>.
However, <alter password> can be performed without any privilege, when the user and the user identifier are identical.

Syntax Rules and Parameters

user_identifier

It is the username to be altered.

<alter password>

It alters the user's password.

- IDENTIFIED BY new_password
  - The new password is encrypted and stored.
  - The length of the password should be shorter than 128 byte.
  - The password is case sensitive.

- REPLACE old_password
  - It can be omitted when ALTER USER ON DATABASE privilege is given.
  - It can not be omitted when ALTER USER ON DATABASE privilege is not given.
    - The user and the user identifier should be identical.

<alter profile>

It alters the profile for the password management policy.

- PROFILE profile_name
  - It allocates profile_name which is created by a user.
- PROFILE DEFAULT
  - It allocates "DEFAULT" which is the default profile.
- PROFILE NULL
  - It does not allocate the profile.

〈password expire〉

It expires the user's password.

〈account lock〉

- ACCOUNT LOCK
  - It locks the user account.
- ACCOUNT UNLOCK
  - It unlocks the user account.

〈alter default tablespace〉

It alters the user's default tablespace.
The tablespace_name should be a data tablespace.

〈alter temporary tablespace〉

It alters the user's temporary tablespace.
The tablespace_name should be a temporary tablespace.

〈alter index tablespace〉

It alters an index tablespace of the user.

- It specifies INDEX TABLESPACE tablespace_name.
  - If the data tablespace is specified, then it becomes a LOGGING index.
  - If the temporary tablespace is specified, then it becomes a NOLOGGING index.
- INDEX TABLESPACE NULL
  - It does not specify an index tablespace.

〈alter schema path〉

It alters the user's schema access path.
If the schema is not specified in user's SQL statement, the schema access path is determined in the schem a order for the naming resolution of the object.

If the schema name is as same as another schema which is previously listed, it is not applied.

The following is an example of objects existing in a schema when performing *ALTER USER u1 SCHEMA PATH (u1, s2, public);* statement.

Schema name	u1	s2	public
-	t1	-	t1
-	-	t2	-
-	-	-	t3

The object name whose schema name is not specified when the user u1 executes the schema is interpreted by the SCHEMA PATH as follows.

- CREATE statement
  - CREATE TABLE t1 (c1 INTEGER);
    - Error: CREATE TABLE u1.t1 (c1 INTEGER);
  - CREATE TABLE t2 (c1 INTEGER);
    - Execution: CREATE TABLE u1.t2 (c1 INTEGER);

- SELECT statement
  - SELECT * FROM t1;
    - Execution: SELECT * FROM u1.t1;
  - SELECT * FROM t2;
    - Execution: SELECT * FROM s2.t2;
  - SELECT * FROM t3;
    - Execution: SELECT * FROM public.t3;

CURRENT PATH

It is the current user's schema path.

A new schema path can be added using CURRENT PATH maintaining the existing schema path as follows.

- The u1's current schema path
  - (u1, public)
- The statement execution
  - ALTER USER u1 SCHEMA PATH (s1, CURRENT PATH, s2);
- The u1's schema path is altered as follows.
  - (s1, u1, public, s2)

ALTER USER PUBLIC <alter schema path>

It alters the schema path of PUBLIC account.
The schema path of PUBLIC account is included in every user's the schema path.

The initial schema path which is allocated to PUBLIC account is as follows.

- DICTIONARY_SCHEMA
- INFORMATION_SCHEMA
- DEFINITION_SCHEMA
- PERFORMANCE_VIEW_SCHEMA
- FIXED_TABLE_SCHEMA

Description

For more information, refer to the rules for each syntax.

Examples

The following is an example of altering the user's password.

```
gSQL> ALTER USER u1 IDENTIFIED BY new_password;
User altered.
```

The following is an example of allocating the profile to the user.

```
gSQL> ALTER USER u1 PROFILE prof1;
User altered.
gSQL> COMMIT;
Commit complete.
```

The following is an example of dropping the user's profile.

```
gSQL> ALTER USER u1 PROFILE NULL;
User altered.
gSQL> COMMIT;
Commit complete.
```

The following is an example of expiring the user's password.

```
gSQL> ALTER USER u1 PASSWORD EXPIRE;
User altered.
gSQL> COMMIT;
Commit complete.
```

The following is an example of unlocking the user's account.

```
gSQL> ALTER USER u1 ACCOUNT UNLOCK;
User altered.
gSQL> COMMIT;
Commit complete.
```

The following is an example of altering the user's DEFAULT TABLESPACE.

```
gSQL> ALTER USER u1 DEFAULT TABLESPACE mem_data_tbs;
User altered.
```

The following is an example of altering the user's TEMPORARY TABLESPACE.

```
gSQL> ALTER USER u1 TEMPORARY TABLESPACE mem_temp_tbs;
User altered.
```

The following is an example of altering the user's INDEX TABLESPACE.

```
gSQL> ALTER USER u1 INDEX TABLESPACE mem_temp_tbs;
User altered.
```

The following is an example of altering the user's schema path.

```
gSQL> ALTER USER u1 SCHEMA PATH ( s1, CURRENT PATH );
User altered.
```

Compatibility

SQL standard covers the concepts of a user, but it does not define the SQL statements associated with cre ating, altering, dropping a user.

For More Information

Refer to the followings.

- CREATE USER
- DROP USER

18.82 ALTER VIEW

Function

It alters the view definition.

Syntax

```
<alter view statement> ::=
    ALTER VIEW view_name <alter view action>
    ;
<alter view action> ::=
    COMPILE
```

Invocation and Access Rules

One of the following privileges is required to perform <alter view statement>.

- (ALTER or CONTROL TABLE) ON TABLE for the view
- (ALTER TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the view belongs
- ALTER ANY TABLE ON DATABASE

Syntax Rules and Parameters

view_name

It is the view name to be altered.
It can define the schema to which the view belongs, such as schema_name.view_name. If schema_name is omitted, the default schema name of the user performing the statement is used.

COMPILE

It compiles the view again.
COMMENT which is given to the view column is initialized.

Description

When the table or the view which is referenced by the view is altered or dropped, then it affects that vie w.

This information can be retrieved from INFORMATION_SCHEMA.VIEWS.

- IS_COMPILED column
  - TRUE: The view was successfully created.
  - FALSE: The view was created with FORCE option when an error exists.

- IS_AFFECTED column
  - TRUE: The table and the view which was referenced by the view was altered.
  - FALSE: After creation and compilation of a view, the table and the view which was referenced by the view was not altered.

Example

The following is an example of compiling the view which is affected by the table change referenced by th at view.

```
gSQL> SELECT TABLE_NAME, IS_AFFECTED
        FROM INFORMATION_SCHEMA.VIEWS
       WHERE TABLE_SCHEMA = 'PUBLIC'
         AND TABLE_NAME = 'V1';
TABLE_NAME IS_AFFECTED
---------- -----------
V1         TRUE
1 row selected.
gSQL> ALTER VIEW v1 COMPILE;
View altered.
COMMIT;
Commit complete.
gSQL> SELECT TABLE_NAME, IS_AFFECTED
```

```
        FROM INFORMATION_SCHEMA.VIEWS
       WHERE TABLE_SCHEMA = 'PUBLIC'
         AND TABLE_NAME = 'V1';
TABLE_NAME IS_AFFECTED
---------- -----------
V1         FALSE
1 row selected.
```

Compatibility

The SQL standard does not define <alter view statement>.

For More Information

Refer to the followings.

- CREATE VIEW
- DROP VIEW

18.83 ANALYZE SYSTEM

Function

It controls the statistics information of the system.

Syntax

```
<analyze system statement> ::=
    ANALYZE SYSTEM [ <analyze action> ]
    ;
<analyze action> ::=
      COMPUTE STATISTICS
    ¦ DELETE STATISTICS
```

Invocation and Access Rules

ANALYZE ANY ON DATABASE privilege is required to perform <analyze system statement>.

Syntax Rules and Parameters

<analyze action>

When it is omitted, the default value is COMPUTE STATISTICS.

COMPUTE STATISTICS

It builds the following statistics information related to the system.

- CPU_OPS (Operations Per Second)
  - It is the number of operations of which the CPU can process per second.
- NETWORK_IOPS (I/O operations Per Second)
  - It is valid for the cluster.

    - It is the number of the network I/O which can be processed per second.

- BUFFER_MISS_PERCENT
    - It is the probability of the disk buffer miss

DELETE STATISTICS

It deletes the statistics information of the system.

Description

The built statistics information of the system is used to calculate the cost of optimization for the query pr ocess.

Examples

The following is an example of building the statistics information of the system by using the <analyze syst em statement> statement.

```
gSQL> ANALYZE SYSTEM COMPUTE STATISTICS;
analyzed.
```

The following is an example of retrieving the built statistics information of the system.

```
gSQL>
SELECT * FROM DBA_STAT_SYSTEM;
 CPU_OPS NETWORK_IOPS NETWORK_BUFSIZE BUFFER_MISS_PERCENT LAST_ANALYZED
-------- ------------ ---------------- -------------------- ----------------------------
53000412         2914            65536                   99  2017-03-30 16:49:42.200000
1 row selected.
```

Compatibility

The SQL standard does not define the concepts of the statistics information.

For More Information

Refer to **ANALYZE TABLE**.

18.84 ANALYZE TABLE

Function

It controls the statistics information of the table.

Syntax

```
<analyze table statement> ::=
    ANALYZE TABLE table_name
    [ <parallel clause> ]
    [ <analyze action> ]
    ;
<parallel clause> ::=
      NOPARALLEL
    | PARALLEL [thread_count]
<analyze action> ::=
      COMPUTE STATISTICS [ <for_clause> ]
    | ESTIMATE STATISTICS <sample_clause> [ <for_clause> ]
    | DELETE STATISTICS
<sample_clause>
      SAMPLE row_count ROWS
    | SAMPLE percentage PERCENT
<for_clause>
      FOR ALL COLUMNS
    | FOR ALL INDEXED COLUMNS
    | FOR COLUMNS column_name [, ...]
    | FOR ALL INDEXES
    | FOR INDEXES index_name [, ...]
```

Invocation and Access Rules

ANALYZE ANY ON DATABASE privilege is required to perform <analyze table statement>.

Syntax Rules and Parameters

table_name

It is the table name.
It can define the schema to which the table belongs, such as schema_name.table_name. If schema_name is omitted, the default schema name of the user performing the statement is used.

〈parallel clause〉

It specifies the number of threads to be used in a analyzing process.
If it is not specified, the default value is PARALLEL.

- NOPARALLEL
  - It does not analyze in parallel.

- PARALLEL [thread_count]
  - It analyzes in parallel.
  - The minimum value of the thread_count is 0, and the maximum value is 64.
  - If the thread_count value is 0 or it is omitted, then it is determined by the number of CPUs in the system.

〈analyze action〉

When it is omitted, the default value is COMPUTE STATISTICS.

COMPUTE STATISTICS

It builds the following statistics information related to the table through the all inspection.

- The table statistics information
  - Row count
  - The number of pages
- The statistics information of each column
  - The number of different values
  - The number of NULL values
  - The average length of the value
  - The minimum value
  - The maximum value
- The statistics information of an index

- The number of different keys
- The number of pages
- The number of leaf pages
- The tree level
- The clustering factor of the index

The statistics information which is built according to the data type of the column is as follows.

Table 18-22 Statistics information built according to the data type

Data type	NUM_DISTINCT	NUM_NULLS	AVG_LENGTH	MIN/MAX
BOOLEAN	O	O	O	X
NATIVE_SMALLINT	O	O	O	O
NATIVE_INTEGER	O	O	O	O
NATIVE_BIGINT	O	O	O	O
NATIVE_REAL	O	O	O	O
NATIVE_DOUBLE	O	O	O	O
NUMBER	O	O	O	O
NUMERIC	O	O	O	O
FLOAT	O	O	O	O
CHAR(n)	O	O	O	It is built when it is 6 4 bytes or smaller.
VARCHAR(n)	O	O	O	It is built when it is 6 4 bytes or smaller.
LONG VARCHAR	X	X	X	X
BINARY	O	O	O	X
VARBINARY	O	O	O	X
LONG VARBINARY	X	X	X	X
DATE	O	O	O	O
TIME	O	O	O	O
TIMESTAMP	O	O	O	O
INTERVAL	O	O	O	O
ROWID	O	O	O	X

ESTIMATE STATISTICS 〈sample_clause〉

It builds the statistics information of the column and the index by using as many samples as the specified 〈sample_clause〉.

- SAMPLE row_count ROWS
  - It uses as many samples as the specied number of rows.
  - row_count is a positive integer bigger than 0.

- SAMPLE percentage PERCENT
  - It uses as many samples as the specied ratio.
  - The percentage is a positive intege in the range between 1 and 99.

If the number of the sampling rows are smaller than the value of **MIN_SAMPLE_ROW_COUNT** property, t hen it follows the property value.

〈for_clause〉

If it is omitted, it builds the statistics information of all possible columns and indexes.

FOR ALL COLUMNS

It builds the statistics information of all possible columns.
It does not build the statistics information of an index.

FOR ALL INDEXED COLUMNS

It builds the statistics information of all columns included in an index.
It does not build the statistics information of other columns.
It does not build the statistics information of an index.

FOR COLUMNS column_name [, ...]

It builds the statistics information of the listed columns.
It does not build the statistics information of unlisted columns.
It does not build the statistics information of an index.

FOR ALL INDEXES

It builds the statistics information of all indexes.
It does not build the statistics information of columns.

FOR INDEXES index_name [, ...]

It builds the statistics information of the listed indexes.
It does not build the statistics information of unlisted indexes.
It does not build the statistics information of a column.

DELETE STATISTICS

It deletes the statistics information of the table.

Description

The statistics information of the table affects the query optimization, so it is very important information.

The time to build the statistics information is increased in proportion to the data volumes in a table. There fore, when the data volume is big, then it is recommended to build the statistics information by using the sampling or to build only the statistics of major information which affects the query.

- The following is an example of building the statistics information by using the sampling.

```
ANALYZE TABLE lineitem ESTIMATE STATISTICS SAMPLE 10 PERCENT;
```

- The following is an example of building the statistics information only of major columns and the inde x.

```
ANALYZE TABLE lineitem COMPUTE STATISTICS FOR ALL INDEXED COLUMNS;
ANALYZE TABLE lineitem COMPUTE STATISTICS FOR ALL INDEXES;
```

Examples

The following is an example of building the statistics information through the all inspection.

```
gSQL> ANALYZE TABLE orders;
Table analyzed.
```

The following is an example of retrieving the built statistics information of the table.

```
gSQL>
SELECT
       TABLE_NAME
     , NUM_ROWS
  FROM
       DICTIONARY_SCHEMA.USER_TABLES
 WHERE
       TABLE_SCHEMA = 'PUBLIC'
   AND TABLE_NAME   = 'ORDERS'
```

```
;
TABLE_NAME NUM_ROWS
---------- --------
ORDERS      1500000
1 row selected.
gSQL>
SELECT
       TABLE_NAME
     , COLUMN_NAME
     , NUM_DISTINCT
     , NUM_NULLS
     , LOW_VALUE
     , HIGH_VALUE
  FROM
       DICTIONARY_SCHEMA.USER_TAB_COLUMNS
 WHERE
       TABLE_SCHEMA = 'PUBLIC'
   AND TABLE_NAME   = 'ORDERS'
;
TABLE_NAME COLUMN_NAME     NUM_DISTINCT NUM_NULLS LOW_VALUE           HIGH_VALUE
---------- --------------- ------------ --------- ------------------- -------------------
ORDERS     O_ORDERKEY           1500000         0 1                   6000000
ORDERS     O_CUSTKEY              99996         0 1                   149999
ORDERS     O_ORDERSTATUS              3         0 F                   P
ORDERS     O_TOTALPRICE         1464556         0 857.71              555285.16
ORDERS     O_ORDERDATE             2406         0 1992-01-01 00:00:00 1998-08-02 00:00:00
ORDERS     O_ORDERPRIORITY            5         0 1-URGENT            5-LOW
ORDERS     O_CLERK                 1000         0 Clerk#000000001     Clerk#000001000
ORDERS     O_SHIPPRIORITY             1         0 0                   0
ORDERS     O_COMMENT            1482071         0 null                null
9 rows selected.
gSQL>
SELECT
       TABLE_NAME
     , INDEX_NAME
     , DISTINCT_KEYS
  FROM
       DICTIONARY_SCHEMA.USER_INDEXES
 WHERE
       TABLE_SCHEMA = 'PUBLIC'
   AND TABLE_NAME   = 'ORDERS'
```

```
;
TABLE_NAME INDEX_NAME        DISTINCT_KEYS
---------- ----------------- -------------
ORDERS     ORDERS_PK_INDEX         1500000
ORDERS     ORDERS_CUSTKEY_FK         99996
2 rows selected.
```

Compatibility

The SQL standard does not define the concepts of the statistics information.

For More Information

Refer to **ANALYZE SYSTEM**.

18.85 AUDIT POLICY

Function

It activates the audit policy.

Syntax

```
<audit policy statement> ::=
    AUDIT POLICY policy_name
    [ <specified_user_option> ]
    [ <specified_success_option> ]
    ;
<specified_user_option> ::=
      BY user_name [, ...]
    ¦ EXCEPT user_name [, ...]
<specified_success_option> ::=
      WHENEVER SUCCESSFUL
    ¦ WHENEVER NOT SUCCESSFUL
```

Invocation and Access Rules

AUDIT SYSTEM ON DATABASE privilege is required to perform <audit policy statement>.

Syntax Rules and Parameters

policy_name

It is the name of the audit policy object to be activated.
The activated audit policy does not effect on the existing session, and it effects only on the newly created session.

〈specified_user_option〉

It specifies the user to be audited.
If omitted, all users are audited.

BY clause and EXCEPT clause can not be used together for the same audit policy.

- BY user_list: If the user to be audited is specified, then use BY clause.
- EXCEPT user_list: If other users excluding a specific user is to be audited, use EXCEPT clause.

〈specified_success_option〉

- WHENEVER SUCCESSFUL
  - If an action succeeds, then the audit record is created.
- WHENEVER NOT SUCCESSFUL
  - If an action fails, then the audit record is created.
- If omitted, both when an action succeeds and fails, the audit record is created.

Description

Activating the audit policy does not affect the existing session, but it starts to audit the newly created ses sion.

Retrieving Audit Record

The audit record is created when it corresponds to the audit policy, and it can be retrieved through DICTI ONARY_SCHEMA.AUDIT_TRAIL view as follows.

```
SELECT logon_user
     , event_timestamp
     , action_name
     , object_name
     , sql_text
  FROM audit_trail
 WHERE policy_name = 'P1'
;
```

SELECT privilege should be given to an ordinary user to retrieve AUDIT_TRAIL.

```
GRANT SELECT ON DICTIONARY_SCHEMA.AUDTI_TRAIL TO user_name;
```

Retrieving Audit Policy Information

The information about the audit policy object can be retrieved through DICTIONARY_SCHEMA.AUDIT_POLICY_OPTIONS view.

```
SELECT policy_name
     , audit_option
     , object_schema
     , object_name
  FROM audit_policy_options
;
```

The information about whether the audit policy object is activated can be retrieved through DICTIONARY_SCHEMA.AUDIT_POLICY_ENABLED view.

```
SELECT policy_name
     , enabled_opt
     , user_name
     , when_success
     , when_failure
  FROM audit_policy_enabled
;
```

Cautions When Using BY and EXCEPT Clauses

Activate the users group if multiple AUDIT POLICY BY clauses are used for the same audit policy.
In other words, the following two examples have the same meaning.

- Example 1: It activates the p1 audit policy for u1 and u2.

```
AUDIT POLICY p1 BY u1;
AUDIT POLICY p1 BY u2;
```

- Example 2: It activates the p1 audit policy for u1 and u2.

```
AUDIT POLICY p1 BY u1, u2;
```

If multiple AUDIT POLICY EXCEPT clauses are used for the same audit policy, only the last AUDIT POLICY clause is valid.

In other words, the following two examples have different meanings.

- Example 1: Only the last clause is valid, and activates the p1 audit policy excluding u2.

```
AUDIT POLICY p1 EXCEPT u1;
AUDIT POLICY p1 EXCEPT u2;
```

- Example 2: It activates the p1 audit policy excluding u1 and u2.

```
AUDIT POLICY p1 EXCEPT u1, u2;
```

BY and EXCEPT can not be used together for the same policy.

- If the audit policy is activated with BY clause, then only BY clause can be used afterwards.

```
AUDIT POLICY p1 BY u1;
```

    - Error

```
AUDIT POLICY p1 EXCEPT u2;
```

- If the audit policy is activated with EXCEPT clause, then only EXCEPT clause can be used afterwards.

```
AUDIT POLICY p1 EXCEPT u1;
```

    - Error: It corresponds to *by all users*.

```
AUDIT POLICY p1;
```

If a user wants to convert the audit policy activated with BY to EXCEPT or to convert the audit policy activ ated with EXCEPT to BY, the activated audit policy should be deactivated first, then it can be converted.

Deactivate the audit policy with NOAUDIT POLICY statement as follows.

- AUDIT POLICY p1 BY u1, u2;
    - NOAUDIT POLICY p1 BY u1, u2;
- AUDIT POLICY p1;
    - NOAUDIT POLICY p1;
- AUDIT POLICY p1 EXCEPT u1, u2;
    - NOAUDIT POLICY p1;
    - NOAUDIT POLICY statement does not have an EXCEPT option.

WHENEVER clause which is used together with BY clause is accumulated.

The following two examples have the same meaning.

- Example 1: It creates the audit record regardless of success/ failure.

```
AUDIT POLICY p1 BY u1 WHENEVER SUCCESSFUL;
AUDIT POLICY p1 BY u1 WHENEVER NOT SUCCESSFUL;
```

- Example 2: It creates the audit record regardless of success/ failure.

```
AUDIT POLICY p1 BY u1;
```

If WHENEVER clauses are used together with EXCEPT clause, only the last WHENEVER clause is valid.

The following two examples have different meanings.

- Example 1: It creates the audit record when it fails.

```
AUDIT POLICY p1 EXCEPT u1 WHENEVER SUCCESSFUL;
AUDIT POLICY p1 EXCEPT u1 WHENEVER NOT SUCCESSFUL;
```

- Example 2: It creates the audit record regardless of success/ failure.

```
AUDIT POLICY p1 EXCEPT u1;
```

Examples

The following is an example of activating the audit policy for all users.

```
AUDIT POLICY table_pol;
```

The information about activation can be viewed through the following query.

```
SELECT policy_name
     , enabled_opt
     , user_name
  FROM audit_policy_enabled
 WHERE policy_name = 'TABLE_POL';
POLICY_NAME  ENABLED_OPT  USER_NAME
-----------  -----------  ---------
TABLE_POL    BY           ALL USERS
```

The following is an example of activating the audit policy by defining specific users.

```
AUDIT POLICY dml_pol BY u1, u2;
```

The following is an example of activating the audit policy by excluding a specific user.

```
AUDIT POLICY read_seq_pol EXCEPT sys;
```

The following is an example of auditing the failure of SQL statement by a specific user.

```
AUDIT POLICY delete_pol BY u1 WHENEVER NOT SUCCESSFUL;
```

Compatibility

The SQL standard does not have the audit policy.

For More Information

Refer to the followings.

- Managing audit policy object
  - **CREATE AUDIT POLICY**
  - **DROP AUDIT POLICY**
  - **ALTER AUDIT POLICY**
- Activating/ deactivating audit policy
  - **AUDIT POLICY**
  - **NOAUDIT POLICY**
- Viewing audit trail: **AUDIT_TRAIL**
- Clearing audit trail: **ALTER DATABASE CLEAR AUDIT TRAIL**

19.

SQL References (C~G)

19.1 CLOSE cursor_name

Function

It closes a cursor.

Syntax

```
<close statement> ::=
    CLOSE cursor_name
    ;
```

Syntax Rules and Parameters

cursor_name

The cursor should be open.
The cursor should be declared with **DECLARE cursor_name** statement in the session.

Description

The cursor is an object which exists in the session and it does not affect the cursor in a different session.

Example

The following is an example of DECLARE, OPEN, FETCH, and CLOSE the cursor by using gsql (interactive SQL tool).

```
gSQL> DECLARE cur1 CURSOR FOR SELECT id, data FROM t1;
Cursor declared.
gSQL> OPEN cur1;
Cursor is open.
```

```
gSQL> \var v_id   INTEGER
gSQL> \var v_data VARCHAR(128)
gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   1 data_1
1 row fetched.
gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   2 data_2
1 row fetched.
gSQL> FETCH cur1 INTO :v_id, :v_data;
no rows fetched.
gSQL> CLOSE cur1;
Cursor closed.
```

Compatibility

Table 19-1 SQL standard compatibility

Feature ID	Description	Compatibility
B031	Basic dynamic SQL	O

For More Information

Refer to the followings.

- **DECLARE cursor_name**
- **OPEN cursor_name**
- **FETCH cursor_name**

19.2 COMMENT ON name IS

Function

It stores the comments about the object in the dictionary.

Syntax

```
<comment statement> ::=
    COMMENT ON <comment object> IS 'comment string'
    ;
<comment object> ::=
      CLUSTER GROUP group_name
    ¦ CLUSTER MEMBER member_name
    ¦ DATABASE
    ¦ PROFILE profile_name
    ¦ AUTHORIZATION user_name
    ¦ TABLESPACE tablespace_name
    ¦ SCHEMA schema_name
    ¦ TABLE [schema_name].table_name
    ¦ COLUMN [schema_name].table_name.column_name
    ¦ INDEX [schema_name].index_name
    ¦ SEQUENCE [schema_name].sequence_name
    ¦ CONSTRAINT [schema_name].constraint_name
    ¦ PROCEDURE [schema_name].procedure_name
```

Invocation and Access Rules

The altering privileges on each object are required to perform <comment statement> as follows.

- CLUSTER GROUP
  - ALTER SYSTEM ON DATABASE
- CLUSTER MEMBER
  - ALTER SYSTEM ON DATABASE
- DATABASE

    - ALTER DATABASE ON DATABASE
- PROFILE
    - ALTER PROFILE ON DATABASE
- AUDIT POLICY
    - AUDIT SYSTEM ON DATABASE
- AUTHORIZATION (user)
    - ALTER USER ON DATABASE
- AUTHORIZATION (role)
    - ALTER ROLE ON DATABASE
- TABLESPACE
    - ALTER TABLESPACE ON DATABASE
- SCHEMA: One of the following privileges is required.
    - The owner of the schema
    - CONTROL SCHEMA ON SCHEMA for the schema
    - ALTER SCHEMA ON DATABASE
- TABLE: One of the following privileges is required.
    - The owner of the table
    - CONTROL TABLE ON TABLE for the table
    - CONTROL SCHEMA ON SCHEMA for the schema to which the table belongs
    - ALTER ANY TABLE ON DATABASE
- COLUMN: One of the following privileges is required.
    - The owner of the table to which the column belongs
    - CONTROL TABLE ON TABLE for the table to which the column belongs
    - CONTROL SCHEMA ON SCHEMA for the schema of the table to which the column belongs
    - ALTER ANY TABLE ON DATABASE
- INDEX: One of the following privileges is required.
    - The owner of the index
    - CONTROL SCHEMA ON SCHEMA for the schema to which the index belongs
    - ALTER ANY INDEX ON DATABASE
- SEQUENCE
    - The owner of the sequence
    - CONTROL SCHEMA ON SCHEMA for the schema to which the sequence belongs
    - ALTER ANY SEQUENCE ON DATABASE
- CONSTRAINT
    - The owner of that constraint
    - CONTROL SCHEMA ON SCHEMA for the schema to which the constraint belongs
    - ALTER ANY TABLE ON DATABASE
- PROCEDURE
    - The owner of the stored procedure/function
    - CONTROL SCHEMA ON SCHEMA for the schema to which the stored procedure/function belong
      s

- ALTER ANY PROCEDURE ON DATABASE

Syntax Rules and Parameters

〈comment object〉

It is an object in which the comments are to be stored. The comments for the following database objects are stored.

- Cluster object
  - CLUSTER GROUP
  - CLUSTER MEMBER
- Non-schema object
  - DATABASE
  - PROFILE
  - AUDIT POLICY
  - AUTHORIZATION (User or role)
  - TABLESPACE
  - SCHEMA
- Schema object
  - TABLE or VIEW
  - COLUMN
  - INDEX
  - SEQUENCE
  - CONSTRAINT
  - PROCEDURE or FUNCTION

If schema_name for the schema object is not specified, the schema name is determined by **Schema Path** of the user performing the statement.

```
COMMENT ON TABLE test_table IS 'test comment';
→ COMMENT ON TABLE user_default_schema.test_table IS 'test comment';
```

'comment string'

It describes the comments to be stored.
Use the empty string ('') to delete the comments as follows.

```
COMMENT ON TABLE test_table IS '';
```

The length of the comment string can not exceed 1024 bytes.

Description

The information can be retrieved from the COMMENTS column of the following dictionary view per each object type.

- Cluster objects
  - CLUSTER GROUP & CLUSTER MEMBER
    - DICTIONARY_SCHEMA.DBA_CLUSTER_COMMENTS view
- Non-schema objects
  - DATABASE
    - DICTIONARY_SCHEMA.ALL_NONSCHEMA_COMMENTS view
    - DICTIONARY_SCHEMA.DBA_NONSCHEMA_COMMENTS view
  - PROFILE
    - DICTIONARY_SCHEMA.DBA_NONSCHEMA_COMMENTS view
  - AUDIT POLICY
    - DICTIONARY_SCHEMA.AUDIT_POLICIES view
  - USER
    - DICTIONARY_SCHEMA.ALL_NONSCHEMA_COMMENTS view
    - DICTIONARY_SCHEMA.DBA_NONSCHEMA_COMMENTS view
  - TABLESPACE
    - DICTIONARY_SCHEMA.ALL_NONSCHEMA_COMMENTS view
    - DICTIONARY_SCHEMA.DBA_NONSCHEMA_COMMENTS view
  - SCHEMA
    - DICTIONARY_SCHEMA.ALL_NONSCHEMA_COMMENTS view
    - DICTIONARY_SCHEMA.DBA_NONSCHEMA_COMMENTS view
- SQL schema objects
  - TABLE
    - DICTIONARY_SCHEMA.USER_TAB_COMMENTS view
    - DICTIONARY_SCHEMA.ALL_TAB_COMMENTS view
    - DICTIONARY_SCHEMA.DBA_TAB_COMMENTS view
  - VIEW
    - DICTIONARY_SCHEMA.USER_TAB_COMMENTS view
    - DICTIONARY_SCHEMA.ALL_TAB_COMMENTS view
    - DICTIONARY_SCHEMA.DBA_TAB_COMMENTS view
  - COLUMN
    - DICTIONARY_SCHEMA.USER_COL_COMMENTS view

        - DICTIONARY_SCHEMA.ALL_COL_COMMENTS view
        - DICTIONARY_SCHEMA.DBA_COL_COMMENTS view
    - INDEX
        - DICTIONARY_SCHEMA.USER_INDEXES view
        - DICTIONARY_SCHEMA.ALL_INDEXES view
        - DICTIONARY_SCHEMA.DBA_INDEXES view
    - SEQUENCE
        - DICTIONARY_SCHEMA.USER_SEQUENCES view
        - DICTIONARY_SCHEMA.ALL_SEQUENCES view
        - DICTIONARY_SCHEMA.DBA_SEQUENCES view
    - CONSTRAINT
        - DICTIONARY_SCHEMA.USER_CONSTRAINTS view
        - DICTIONARY_SCHEMA.ALL_CONSTRAINTS view
        - DICTIONARY_SCHEMA.DBA_CONSTRAINTS view
    - PROCEDURE & FUNCTION
        - DICTIONARY_SCHEMA.USER_PROCEDURES view
        - DICTIONARY_SCHEMA.ALL_PROCEDURES view
        - DICTIONARY_SCHEMA.DBA_PROCEDURES view

For more information about the detailed description of each view, refer to **DICTIONARY_SCHEMA**.

Examples

The following is an example of creating the comment on the table.

```
gSQL> COMMENT ON TABLE t1 IS 'test comment on table t1';
Comment created.
```

The following is an example of creating the comment on the column.

```
gSQL> COMMENT ON COLUMN t1.id IS 'test comment on column t1.id';
Comment created.
```

The following is an example of creating the comment on the schema.

```
gSQL> COMMENT ON SCHEMA s1 IS 'test comment on schema s1';
Comment created.
```

Compatibility

〈comment statement〉 does not exist in SQL standard.

19.3 COMMIT

Function

It terminates the current transaction and makes all changes permanent.

Syntax

```
<commit statement> ::=
    COMMIT [ WORK ]
       [ [ <commit comment clause> ] [ <commit write clause> ] ¦
         [ <commit force clause> ] [ <commit comment clause> ] ]
    ;
<commit comment clause> ::=
      COMMENT 'comment_string'
<commit write clause> ::=
      WRITE [ WAIT ¦ NOWAIT ]
<commit force clause> ::=
    FORCE 'xid_string'
```

Syntax Rules and Parameters

WORK

It is the reserved word which does not affect the operation.

〈commit comment clause〉

- COMMENT 'comment_string'
  - It specifies the comment to the transaction when committing the transaction.

〈commit write clause〉

It determines whether to wait until the redo logs generated by the commit operation are written on the r edo log file.

- WAIT
  - It waits until the redo logs generated by the commit operation are written to the redo log file, and then the operation is terminated.
- NOWAIT
  - The operation is terminated when the redo logs generated by the commit operation are written to the redo log buffer.
- If it is not specified, it follows the property.

〈commit force clause〉

It is used to manually commit a distributed transaction.

- FORCE 'xid_string'
  - It commits the distributed transaction 'xid_string'.
  - 'xid_string' consists of *'format_id.transaction_id.branch_id'*.

Description

COMMIT statement completes the following statements which were executed in a transaction.

- Data Manipulation Language (DML) statement
  - It is the statement which changes data, such as INSERT, UPDATE, DELETE.
- Data Definition Language (DDL) statement
  - It is the statement which changes the structure and definition of the objects, such as CREATE, DROP, ALTER, TRUNCATE, GRANT, REVOKE.

Exceptionally, the following DDL statements which manage the OS resources or change the DATA TYPE are automatically committed.

- **CREATE TABLESPACE**
- **DROP TABLESPACE**
- **ALTER TABLESPACE**
- ALTER TABLE .. ALTER COLUMN .. SET DATA TYPE: **〈alter column data type clause〉**

When performing COMMIT, the cursor opened by WITHOUT HOLD option is automatically closed. For more information about cursors, refer to the following cursor related statements.

- **DECLARE cursor_name**
- **OPEN cursor_name**

If the transaction violates the DEFERRED constraint, the COMMIT statement fails and the transaction is rol

led back. For more information about DEFERRED constraint, refer to **SET CONSTRAINTS**.

Example

The following is an example of performing COMMIT after executing INSERT statement.

```
gSQL> INSERT INTO t1 VALUES ( 1, 'anonymous' );
1 row created.
gSQL> COMMIT WORK COMMENT 'INSERT T1';
Commit complete.
```

Compatibility

Table 19-2 SQL standard compatibility

Feature ID	Description	Compatibility
T261	Chained transactions	X

For More Information

Refer to the followings.

- **ROLLBACK**
- **SAVEPOINT savepoint_specifier**

19.4 CREATE AUDIT POLICY

Function

It creates an audit policy object.
AUDIT POLICY should be performed to activate the created audit policy object.

Syntax

```
<audit policy definition> ::=
    CREATE AUDIT POLICY policy_name
    { <privilege_audit_clause> ¦  <action_audit_clause> ¦ <privilege_audit_clause>
<action_audit_clause> }
    ;
<privilege_audit_clause> ::=
    PRIVILEGES <database_privilege> [, ...]
<action_audit_clause> ::=
    ACTIONS { <object_action_audit> ¦ <system_action_audit> } [, ...]
<object_action_audit> ::=
      ALL ON [schema_name.]object_name
    ¦ <object_action> ON [schema_name.]object_name
<system_action_audit> ::=
      ALL
    ¦ DDL
    ¦ <system_action>
```

Invocation and Access Rules

AUDIT SYSTEM ON DATABASE privilege is required to perform <audit policy definition>.

Syntax Rules and Parameters

policy_name

It is the name of the audit policy to be created.

〈privilege_audit_clause〉

The privilege audit is auditing when the SQL statement is successfully performed by using the database pr ivilege.
It can audit a specific user performing SQL statement by using the database privilege, and it does not rec ord the privilege audit for SYS user who is the owner of the database.

The following is an example of granting SELECT ANY TABLE privilege to user u1 and activating the audit policy.

```
CREATE AUDIT POLICY p1
       PRIVILEGES SELECT ANY TABLE;
AUDIT POLICY p1;
```

If the user u1 performs the following SQL statement, the privilege audit differently operates.

- SELECT * FROM u1.t1;
  - It does not create the audit record by performing SQL statement with the privilege as an owner o f table u1.t1.
- SELECT * FROM u2.t1;
  - It creates the audit record by performing SQL statement with SELECT ANY TABLE privilege.

〈database_privilege〉 which can be described in the privilege audit can be viewed with the following quer y.

```
SELECT PRIVILEGE_NAME FROM V$AUDITABLE_DB_PRIVILEGES;
```

〈action_audit_clause〉

It audits an action for a specific object and an action for the entire database.

〈object_action_audit〉

ALL ON object_name

It means all actions which can list objects corresponding to object_name.

The following table describes audit actions of which each object type can audit.

Table 19-3 Audit action per object type

Object type	Action
Table	ALTER, COMMENT, DELETE, GRANT, INDEX, INSERT, LOCK, RENAME, SELECT, UPDATE
View	ALTER, COMMENT, GRANT, SELECT
Sequence	ALTER, COMMENT, GRANT, SELECT
Stored Function/Procedure	ALTER, COMMENT, EXECUTE, GRANT

〈object_action〉 ON object_name

Each separate action per a specific object should be listed by specifying ON clause as follows.

```
CREATE AUDIT POLICY p1
       ACTIONS INSERT ON u1.t1
             , DELETE ON u1.t1
             , UPDATE ON u1.t1
;
```

Caution of EXECUTE action

Auditing the success or the failure of stored function or stored procedure is determined only based on whether it is executable at the time of execution.

- WHENEVER NOT SUCCESSFUL creates the audit record when neither the stored function nor procedure is executable.
- WHENEVER SUCCESSFUL creates the audit record even though an error occurs while executing SQL statement within the stored function or the procedure
- If an auditing for the failure of SQL statement within the stored function or the procedure is required, then the audit target should include the corresponding SQL statement.

〈system_action_audit〉

It audits the system action which occurs in the database regardless of a specific object.

- 〈system_action〉

The valid system action can be retrieved by using the following query.

```
SELECT ACTION_NAME FROM V$AUDITABLE_SYSTEM_ACTIONS;
```

- ALL

It means all system actions.

- DDL

It means all Data Definition Language (DDL).

Description

An audit policy object is an object which defines auditing targets.
Perform AUDIT POLICY statement to activate an audit policy.

Though it is possbile to define and activate multiple audit policies, but it is recommended to maintain cert ain numbers of audit policies.
It is also recommended to bind multiple small pieces of policies into a small number of groups.

The information about an option of the created audit policy object can be retrieved through AUDIT_POLI CY_OPTIONS view as follows.

```
SELECT audit_option
     , audit_option_type
     , object_schema
     , object_name
  FROM audit_policy_options
 WHERE policy_name = 'P1'
;
AUDIT_OPTION AUDIT_OPTION_TYPE OBJECT_SCHEMA  OBJECT_NAME
------------ ----------------- -------------- ------------
DELETE         OBJECT ACTION     U1           T1
INSERT         OBJECT ACTION     U1           T1
UPDATE         OBJECT ACTION     U1           T1
```

Creating Audit Record

If an action corresponding to multiple audit policies occurs, then one or more audit records are created.

If similar audit options are listed as follows, then one audit record is created.

- Defining an audit policy

```
CREATE AUDIT POLICY p1
       PRIVILEGES SELECT ANY TABLE
       ACTIONS SELECT;
AUDIT POLICY p1;
```

- Performing an audit action

```
SELECT * FROM other_user.t1;
```

If two different audit options are listed as follows, then two audit records are created.

- Defining an audit policy

```
CREATE AUDIT POLICY p1
       ACTIONS SELECT ON u1.t1
             , SELECT ON u2.t2;
AUDIT POLICY p1;
```

- Performing an audit action

```
SELECT COUNT(*) FROM u1.t1 A, u2.t2 B WHERE A.id = B.id;
```

If multiple audit policies are activated for the same action as follows, then two audit records are created.

- Defining an audit policy

```
CREATE AUDIT POLICY p1
       PRIVILEGES SELECT ANY TABLE;
AUDIT POLICY p1;
CREATE AUDIT POLICY p2
       ACTIONS SELECT;
AUDIT POLICY p2;
```

- Performing an audit action

```
SELECT * FROM other.t1;
```

Examples

The following is an example of defining an audit policy which audits an privilege.

```
CREATE AUDIT POLICY policy_table
       PRIVILEGES CREATE ANY TABLE
                , DROP ANY TABLE
;
```

The following is an example of defining an audit policy which audits an action for an object.

```
CREATE AUDIT POLICY policy_dml
       ACTIONS INSERT ON u1.t1
             , DELETE ON u1.t1
             , UPDATE ON u1.t1
             , ALL    ON u1.t2
;
```

The following is an example of defining an audit policy which audits the system action.

```
CREATE AUDIT POLICY policy_drop
       ACTIONS DROP TABLE, TRUNCATE TABLE
;
```

The following is an example of defining an audit policy which combines all examples above.

```
CREATE AUDIT POLICY policy_group
       PRIVILEGES CREATE ANY TABLE
                , DROP ANY TABLE
       ACTIONS INSERT ON u1.t1
             , DELETE ON u1.t1
             , UPDATE ON u1.t1
             , ALL    ON u1.t2
             , DROP TABLE
             , TRUNCATE TABLE
;
```

Compatibility

The audit policy does not exist in SQL standard.

For More Information

Refer to the followings.

- Managing audit policy object
  - **CREATE AUDIT POLICY**
  - **DROP AUDIT POLICY**
  - **ALTER AUDIT POLICY**

- Activating/ deactivating audit policy
  - **AUDIT POLICY**
  - **NOAUDIT POLICY**

- Retrieving audit trail: **AUDIT_TRAIL**

- Clearing audit trail: **ALTER DATABASE CLEAR AUDIT TRAIL**

19.5 CREATE CLUSTER GROUP

Function

It creates a cluster group which is to participate in a cluster system.

Syntax

```
<cluster group definition> ::=
    CREATE CLUSTER GROUP group_name
        <cluster member definition> [, ...]
    ;
<cluster member definition> ::=
    CLUSTER MEMBER member_name <connection attribute> [<member position>]
<connection attribute> ::=
    HOST 'address' PORT port_no
<member position> ::=
    POSITION DEFAULT
  ¦ POSITION MAX
  ¦ POSITION number
```

Invocation and Access Rules

It can be performed in a cluster system.
ADMINISTRATION ON DATABASE privilege is required to perform <cluster group definition>.

Syntax Rules and Parameters

group_name

It is the name of a cluster group.
An identical cluster group name or a cluster member name should not exist.
The name length should be shorter than 128 bytes.

〈cluster member definition〉

It defines a cluster member to be included in a cluster group.
A cluster group can include maximum 32 cluster members.
A cluster group which is created first in a cluster system can define only one cluster member, and should i nclude itself as a cluster member.

member_name

It is the name of a cluster member.
The name of a cluster member should be same as the name of the member which was defines when crea ting the database of that member.
An identical cluster group name or a cluster member name should not exist.
The name length should be shorter than 128 bytes.

The start-up phase of the cluster member should be the GLOBAL OPEN phase.

〈connection attribute〉

It defines the connection information for the communication between the cluster members.
〈connection attribute〉 should be as same as the HOST and PORT which were defined when the database of that cluster member was created.
The combination of HOST and PORT should be unique in the cluster system.

- HOST 'address' uses the host name or IPv4 address. If the host name is used, then it uses the first IPv 4 address of the system.
- PORT port_no should be in the range between 1024 ~ 49151.

〈member position〉

It assigns the position number of the cluster member.

- POSITION DEFAULT
  - The system automatically assigns the position number.
- POSITION MAX
  - It assigns the new member position number even when an empty position number exists.
  - It assigns the value bigger than the biggest member position.
- POSITION number
  - It assigns the position number corresponding to the number
  - The position number should be unique in the cluster system.
  - The position number should be an empty position number, and it should be same or smaller than

the biggest position number.
- If it is omitted, the default value is POSITION DEFAULT.

The member_position information of the cluster member can be retrieved through DBA_CLUSTER view.

```
SELECT member_name, member_id, member_position FROM dba_cluster;
```

If the following position numbers are being used,

- G1N1: 0
- G1N2: 1
- G2N2: 3
- G3N2: 5

The following values are assigned according to each option.

- POSITION DEFAULT
  - It assigns 2 which is an empty value.
- POSITION MAX
  - It assigns 6 which is a value of a new position number.
- POSITION 3
  - It is duplicated, so it is an error.
- POSITION 4
  - It assigns 4 which is a position number.

Description

<cluster group definition> statement does not rebalance the shard of tables.

Perform the following statements to rebalance the shard to an added cluster group.

- **ALTER DATABASE REBALANCE**
- **ALTER TABLE name REBALANCE**

Examples

The following is an example of creating a cluster group which consists of two cluster members.

```
gSQL>
CREATE CLUSTER GROUP g1
    CLUSTER MEMBER g1n1 HOST '192.168.0.11' PORT 10110
;
Cluster Group created.
gSQL>
ALTER CLUSTER GROUP g1
    ADD CLUSTER MEMBER g1n2 HOST '192.168.0.12' PORT 10120
;
Cluster Group altered.
gSQL>
CREATE CLUSTER GROUP g2
    CLUSTER MEMBER g2n1 HOST '192.168.0.21' PORT 10210,
    CLUSTER MEMBER g2n2 HOST '192.168.0.22' PORT 10220
;
Cluster Group created.
```

Compatibility

The SQL standard does not define the concepts of the cluster.

For More Information

Refer to the followings.

- **DROP CLUSTER GROUP**
- **ALTER CLUSTER GROUP name ADD MEMBER**

19.6 CREATE CLUSTER LOCATION

Function

It creates the connection information of a cluster member.

Syntax

```
<cluster location definition> ::=
    CREATE CLUSTER LOCATION member_name
    <cluster connection attribute>
    ;
<cluster connection attribute> ::
       HOST 'address' PORT port_no
```

Invocation and Access Rules

It can be performed in a cluster system.

ADMINISTRATION ON DATABASE privilege is required to perform <cluster location definition>.

Syntax Rules and Parameters

member_name

It is the name of a cluster member.
The same cluster member name should not exist in the registered cluster location information.
The length of the name should be shorter than 128 bytes.

<cluster connection attribute>

It defines the connection information for the communication between the cluster members.
The combination of HOST and PORT should be unique in the cluster system.

- HOST 'address' uses the host name or IPv4 address. If the host name is used, then it uses the first IPv 4 address of the system.
- PORT port_no should be in the range between 1024 ~ 49151.

Description

Generally, the information of the cluster location is automatically created by using the connection inform ation provided when creating the cluster group or adding the cluster member. The created information is deleted together when deleting the cluster member and the cluster group.

If the information of the cluster location is modified, then the connection information can be modified by using **ALTER CLUSTER LOCATION** without deleting or recreating the cluster member.

Examples

```
gSQL>
CREATE CLUSTER LOCATION g1n2
    HOST '192.168.0.12' PORT 10120,
;
Created
```

Compatibility

The SQL standard does not define the concepts of the cluster.

For More Information

Refer to **DROP CLUSTER LOCATION** .

19.7 CREATE DISK DATA TABLESPACE

Function

It defines the disk data tablespace.

Syntax

```
<disk data tablespace statement> ::=
    CREATE DISK [ DATA ] TABLESPACE tablespace_name
        DATAFILE <disk datafile clause> [, ...]
        [ <data tablespace management clause> [, ...] ]
<disk datafile clause> ::=
     'filename'
        [ SIZE <size clause> ¦ REUSE ¦ SIZE <size clause> REUSE ]
        [ <autoextend clause> ]
        [ AT <domain_name> ]
<autoextend clause>
    AUTOEXTEND { ON [ <next size clause> ] [ <max size clause> ] ¦ OFF }
<next size clause>
    NEXT <size clause>
<max size clause>
    MAXSIZE { <size clause> ¦ UNLIMITED }
<size clause> ::=
    integer [ K ¦ M ¦ G ¦ T ]
<data tablespace management clause> ::=
      { ONLINE ¦ OFFLINE }
    ¦ EXTSIZE <size clause>
```

Invocation and Access Rules

CREATE TABLESPACE ON DATABASE privilege is required to perform <disk data tablespace definition>.

The user who performed the statement has CREATE OBJECT ON TABLESPACE privilege for the created ta
blespace.

The following privileges are required to create an object on the created tablespace.

- CREATE OBJECT ON TABLESPACE for the tablespace
- USAGE TABLESPACE ON DATABASE

Syntax Rules and Parameters

tablespace_name

It is the name of the tablespace to be created.
The length of the tablespace name should be shorter than 128 bytes.

〈disk datafile clause〉

- 'filename'
  - It is the name of the file which stores and manages the data.
  - It is the space to store tables and index pages which were created in the disk tablespace.
  - filename is either a new file or an existing file.
  - filename length should be shorter than 1024 bytes.

- SIZE 〈size clause〉
  - If it is a new file, then it specifies the initial size by using SIZE clause.
  - If a file exists, then an error occurs.
  - The file size can be in the range between 1 M ~ maximum 30 G.

- REUSE
  - If it is an existing file, then it uses REUSE clause.
  - If a file does not exist, then it creates a new file.
  - The size of the newly created files is determined by USER_DATA_TABLESPACE_SIZE property.

- SIZE 〈size clause〉 REUSE
  - If both SIZE clause and REUSE clause are specified, then it is operated according to the existence of filename as follows.
    - If it is a new filename, then it specifies the initial file size by using SIZE clause.
    - If it is an existing filename, then it adjust the size to the value in SIZE clause.

〈autoextend clause〉

It sets the automatic expand property to ON or OFF. If it is set to ON, then it can specify the automatic ex panded size and the maximum size of the data file.

〈next size clause〉

It specifies the size to be extended when the data file in use does not have available space.

〈max size clause〉

It specifies the maximum expanded size of the data file.

〈size clause〉

It specifies the file size in byte. (If it is omitted, the default unit is bytes.)

- K: Kilobytes
- M: Megabytes
- G: Gigabytes
- T: Terabytes

〈domain_name〉

It is the name of member or the group to performs the statement.
If it is omitted, then it is performed for all groups.

ONLINE | OFFLINE

It determines whether to ONLINE/ OFFLINE the tablespace.

- If it is set to ONLINE, then the tablespace is available as soon as it is created.
- If it is set to OFFLINE, then it is not available, It is available only after it is switched to ONLINE.

EXTSIZE 〈size clause〉

It specifies the extent size of the tablespace.

- The extent size is specified in byte, and one of the six (64 K, 128 K, 256 K, 512 K, 1 M, 2 M) is selecte d.
- If the extent size is set specified between 64 K ~ 128 K, then it is set to 128 K, and if it is specifies big ger than 2 M, then it is set to 2 M.

Description

The data tablespace is an object which provides a physical storage to store SQL schema objects such as a table, and index (LOGGING).

Examples

The following is an example of creating the disk data tablespace.

```
gSQL> CREATE DISK TABLESPACE space1 DATAFILE 'test_file_1.dbf' SIZE 10M REUSE;
Tablespace created.
```

The following is an example of creating the tablespace which consists of multiple data files.

```
gSQL> CREATE DISK TABLESPACE space1
             DATAFILE 'test_file_3_1.dbf' SIZE 10M REUSE,
                      'test_file_3_2.dbf' SIZE 10M REUSE;
Tablespace created.
```

Compatibility

The SQL standard does not define the concepts of the tablespace.

For More Information

Refer to the followings.

- DROP TABLESPACE
- ALTER TABLESPACE
- ALTER DATABASE DATAFILE AUTOEXTEND

19.8 CREATE GLOBAL TEMPORARY TABLE

Function

It creates a new global temporary table.

Syntax

```
<global temporary table definition> ::=
    CREATE GLOBAL TEMPORARY TABLE table_name
        ( <table element> [, ...] )
        [ <table commit action clause> ]
        [ TABLESPACE tablespace_name ]
    ;
<global temporary table definition: AS query expression> ::=
    CREATE GLOBAL TEMPORARY TABLE table_name
        [ TABLESPACE tablespace_name ]
        AS <query expression> [ WITH [ NO ] DATA ]
    ;
<table commit action clause> ::=
    ON COMMIT { PRESERVE ¦ DELETE } ROWS
```

The definition of <table element> is as same as that of <table_definition>.
For more information, refer to **CREATE TABLE**.

Invocation and Access Rules

A user should satisfy the following conditions to perform <global temporary table definition> statement.

- Table creation privilege
  - Refer to the access privilege in **CREATE TABLE**.
- SELECT access privilege
  - Refer to the access privilege in **SELECT**.

Syntax Rules and Parameters

table_name

It is the table name to be created.
For more information, refer to **table_name**.

other syntax

For more information about other syntaxes, refer to the syntax in **CREATE TABLE** and in **CREATE TABLE AS SELECT** statement.

Description

GLOBAL TEMPORARY TABLE is used to store the data which is maintained while a transaction or a session is performed.
It is used for the purpose as same as that of the variable of which a developer temporarily stores the mid-data of the operation when developing an application.

The global temporary table has the following features.

- The definition of the global temporary table can be viewed in every session.
- The physical segment is not allocated when defining the global temporary table, but the segment subordinated to that session is allocated when it is inserted for the first time.
- The data of the global temporary table can be viewed in a session or a transaction which was inserted.
- The tablespace to store the data of the global temporary table is determined as follows.

Whether to specify tablespace	The tablespace in which the table is created
It specifies the tablespace.	It is created in the specified tablespace.
It does not specify the tablespace.	It is created in the default temporary tablespace of the current session user.

- The index for the global temporary table is subordinate to the same session of the corresponding table, and the time duration is as same as that of the table.
- It can define the view for the global temporary table.
- It can not specify <table sharding strategy> statement which describe the cluster-related features of the table for the global temporary table, nor <table global secondary index clause> statement.
- It can not specify <table attribute clause> which describes the physical attributes of the table for the global temporary table, nor <index attribute clause> which describes the physical attributes of the inde

x.

- <index attribute clause> which describes the physical attributes of the index can not be specified for t he index which is created based on the global temporary table.
- If a transaction is terminated by <table commit action clause>, it can determine how to process the re maining data.

Table commit action	Description
ON COMMIT PRESERVE ROWS	It maintains the data remained in a table even after COMMIT or ROLLBACK.
ON COMMIT DELETE ROWS (defa ult)	It deletes all data remained in a table at the time of COMMIT or ROLLBACK (TR UNCATE).

- It supports most of DDLs for an ordinary table. (Including ALTER and TRUNCATE)
  - It does not support CLUSTER-related statement (SHARD and global secondary index-related state ment).
  - DDL for the global temporary table which is currently used in its own session or in another sessio n causes an error.
  - DDL is available after removing all segments which is used as TRUNCATE TABLE or COMMIT in all sessions in use.
- It supports all DMLs and select statements for an ordinary table.
- All alteration for the global temporary table (DML) do not leave the redo log.
- All alteration for the global temporary table (DML) leaves the undo log, the location of the undo log i s determined according to TEMP_UNDO_ENABLED property.

TEMP_UNDO_ENABLED value	Description
TRUE	The undo log is recorded in the default temporary tablespace of the database sy stem.
FALSE	The undo log is recorded in the undo tablespace of the database system.

- TRUNCATE command for the global temporary table truncates only the segment of the correspondin g session.
- If the session is terminated, all segments are TRUNCATEd and then returned.

Examples

The following is an example of performing CREATE GLOBAL TEMPORARY TABLE statement.

```
gSQL> CREATE  GLOBAL TEMPORARY TABLE SESSION_TABLE1(
        COL1    CHAR(10)
       ,COL2    VARCHAR2(20)
       ,COL3    NUMBER(10)
```

```
)   ON  COMMIT  DELETE ROWS;
Table created.
```

The following is an example of performing CREATE GLOBAL TEMPORARY TABLE ... AS SELECT statemen t.

```
gSQL> CREATE  GLOBAL TEMPORARY TABLE SESSION_TABLE2
    ON  COMMIT  PRESERVE ROWS
    AS  SELECT  *
          FROM  EMPLOYEES;
Table created.
```

Compatibility

CREATE GLOBAL TEMPORARY TABLE and CREATE GLOBAL TEMPORARY TABLE AS SELECT statement fo llows the definition of SQL standard <table definition>. However, the following is an extension of SQL sta ndard.

- SQL standard require parentheses outside SELECT clause, but it is optional in GOLDILOCKS.
- SQL standard require WITH [NO] DATA clause, but it is optional in GOLDILOCKS.
- The concepts of tablespace in GOLDILOCKS is an extended concept, and it is not supported in SQL st andard.

Table 19-4 SQL standard compatibility

Feature ID	Description	Compatibility
T171	LIKE clause in table definition	X
T172	AS subquery clause in table definition	O
F531	Temporary tables	X
S051	Create table of type	X
S043	Enhanced reference types	X
S081	Subtables	X
T173	Extended LIKE clause in table definition	X
T180	System-versioned tables	X
F692	Extended collation support	X
T174	Identity columns	O
T175	Generated columns	X
S071	SQL paths in function and type name resolution	X
F321	User authorization	O
T322	Extended roles	X
F762	CURRENT_CATALOG	O

Feature ID	Description	Compatibility
F763	CURRENT_SCHEMA	O

For More Information

Refer to the followings.

- CREATE TABLE
- CREATE TABLE AS SELECT

19.9 CREATE IMMUTABLE TABLE

Function

It creates a new immutable table.

Syntax

```
<immutable table definition> ::=
    CREATE IMMUTABLE TABLE table_name
        ( <table element> [, ...] )
        [ <table sharding strategy> ]
        [ <table attribute clause> [...] ]
        [ TABLESPACE tablespace_name ]
        [ <table global secondary index clause> ]
    ;
<immutable table definition: AS query expression> ::=
    CREATE IMMUTABLE TABLE table_name
        [ ( column_name [, ...] ) ]
        [ <table sharding strategy> ]
        [ <table attribute clause> [, ...] ]
        [ TABLESPACE tablespace_name ]
        [ <table global secondary index clause> ]
        AS <query expression> [ WITH [ NO ] DATA ]
    ;
```

The definition for <table element>, <table sharding strategy>, <table attribute clause> and <table global secondary index clause> are as same as that of <table_definition>. For more information, refer to **CREATE TABLE**.

Invocation and Access Rules

The user should satisfy the following conditions to perform <immutable table definition>.

- The privilege to create the table
  - Refer to the access rules for **CREATE TABLE** statement.
- The privilege for SELECT to access
  - Refer to the access rules for **SELECT** statement.

Syntax Rules and Parameters

table_name

It is the table name to be created and it should be unique in the schema.
It can define the schema to which the table belongs, such as schema_name.table_name. If schema_name is omitted, the default schema name of the user performing the statement is used.
The length of the table name should be shorter than 128 byte.

Other Syntax

For other syntaxes, refer to the syntaxes for **CREATE TABLE** and **CREATE TABLE AS SELECT**.

Description

An immutable table is used when it is required to prevent the record stored in the table from being altered or deleted and to prevent the table from being dropped.

An immutable table can be dropped when a user, a schema, a tablespace and a cluster group are dropped.

The following SQL statements are not allowed for an immutable table.

- UPDATE
- DELETE
- ALTER TABLE RENAME/DROP COLUMN
- ALTER TABLE SET UNUSED COLUMN
- ALTER TABLE DROP UNUSED COLUMNS
- ALTER TABLE RENAME TO
- TRUNCATE TABLE
- DROP TABLE

The following SQL statements are allowed for an immutable table.

- INSERT
- SELECT
- SELECT .. FOR UPDATE
- CREATE/ALTER/DROP INDEX
- ALTER TABLE ADD/ALTER COLUMN
- ALTER TABLE ADD/ALTER/RENAME/DROP CONSTRAINT
- ALTER TABLE for physical property changes
- ALTER TABLE ADD/DROP SUPPLEMENTAL LOG
- LOCK TABLE
- ALTER TABLE .. ADD/ALTER/DROP GLOBAL SECONDARY INDEX
- ALTER DATABASE MOVE SHARD
- ALTER TABLE .. MOVE SHARD
- ALTER TABLE .. SPLIT SHARD
- ALTER TABLE .. MERGE SHARD
- DROP TABLESPACE
- DROP SCHEMA
- DROP USER
- DROP CLUSTER GROUP

Examples

The following is an example of executing CREATE IMMUTABLE TABLE statement.

```
gSQL> CREATE IMMUTABLE TABLE t1
(
    id INTEGER PRIMARY KEY,
    name VARCHAR(128),
    addr VARCHAR(128)
);
Table created.
```

The following is an example of executing CREATE IMMUTABLE TABLE ... AS SELECT statement.

```
gSQL> CREATE IMMUTABLE TABLE T2
       AS SELECT *
             FROM T1;
```

```
Table created.
```

Compatibility

The SQL standard does not cover CREATE IMMUTABLE TABLE and CREATE IMMUTABLE TABLE AS SELEC T statements.

For More Information

Refer to the followings.

- CREATE TABLE
- CREATE TABLE AS SELECT

19.10 CREATE INDEX

Function

It creates an index.

Syntax

```
<index definition> ::=
    CREATE [ UNIQUE ] INDEX index_name
        ON table_name ( <index column element> [, ...] )
        [ <index attributes> [...] ]
        [ TABLESPACE tablespace_name ]
    ;
<index column element> ::=
    column_name [ ASC ¦ DESC ] [ NULLS FIRST ¦ NULLS LAST ]
<index attributes> ::=
      <physical attribute clause>
    ¦ STORAGE ( <segment attr clause> [...] )
    ¦ <parallel clause>
<physical attribute clause> ::=
      PCTFREE integer
    ¦ INITRANS integer
    ¦ MAXTRANS integer
<segment attr clause> ::=
      INITIAL <size_clause>
    ¦ NEXT <size_clause>
    ¦ MINSIZE <size_clause>
    ¦ MAXSIZE <size_clause>
<size clause> ::=
      integer [ K ¦ M ¦ G ¦ T ]
<parallel clause> ::=
      NOPARALLEL
    ¦ PARALLEL [ integer ]
```

Invocation and Access Rules

The user should satisfy the following conditions to perform <index definition>.

- One of the following privileges is required to create an index on the table.
  - (INDEX or CONTROL TABLE ON) TABLE for that table
  - CONTROL SCHEMA ON SCHEMA for the schema to which the table belongs
  - CREATE ANY INDEX ON DATABASE

- One of the following privileges is required for the schema on which the index is to be created.
  - (CREATE INDEX or CONTROL SCHEMA) ON SCHEMA for the schema
  - CREATE ANY INDEX ON DATABASE

- One of the following privileges is required for the tablespace on which the index is to be created.
  - CREATE OBJECT ON TABLESPACE for that tablespace
  - USAGE TABLESPACE ON DATABASE

- The owner of the index is determined as follow.
  - The owner of the schema to which the index belongs.
  - If the schema to which the index belongs is PUBLIC, then it is the user who executed the stateme nt.

Unique indexes in a cluster system should include all sharding keys.

Syntax Rules and Parameters

UNIQUE

It does not allow duplicate values for the columns of the index.

index_name

It is the index name to be created and it should be a unique name within the schema.
If the schema name is omitted, the index is created in the schema to which the referring table belongs.
The length of the index name should be shorter than 128 bytes.

table_name

It is the table name which creates the index.
The schema to which a table belongs, such as schema_name.table_name, can be defined. If schema_name is omitted, the default schema name of the user performing the statement is used.

column_name

It is the column name to be used as an index key.
One or more columns should be defined, and maximum 32 columns can be used as an index key.

The following constraints can occur depending on the implementation.

- If the column data type included in an index is LONG CHARACTER VARYING, LONG BINARY VARYING, an index can not be created.
- An index is created only when the sum of the column precisions is less than 1200 bytes.

ASC | DESC

It specifies the sort order of a column.

- ASC: It is sorted in ascending order.
- DESC: It is sorted in descending order.
- If not specified, the default value is ASC.

NULLS FIRST | NULLS LAST

It specifies the sort order of the NULL value.

- NULLS FIRST: It precedes the non-NULL values.
- NULLS LAST: It is behind the non-NULL values.
- If not specified, the default value is NULLS LAST.

〈physical attribute clause〉

It defines the physical attribute of the index.

- PCTFREE integer
  - Definition
    - It is the reserved space for adjusting the page split frequency caused by the key inserted in the page.
    - It applies only to the index bottom-up build.

- It can use the value from 0 to 99.
- If it is omitted, the default value is the value set in DEFAULT_INDEX_PCTFREE property.

- INITRANS integer
  - Definition
    - It specifies the initial number of transactions which can simultaneously access the page.
    - If the number of users who access the index is small, INITRANS is set to low, and if the numb er of users who simultaneously access the index is big, INITRANS is set to high.
    - If necessary, it is automatically increased to the specified MAXTRANS.
  - It can use the value from 1 to 32.
  - If it is omitted, the default value is 4.

- MAXTRANS integer
  - Definition
    - It specifies the maximum number of transactions which can simultaneously access the page.
  - It can use the value from 1 to 32.
  - If it is omitted, the default value is 8.

〈segment attr clause〉

It specifies the information for the index storage space.

- INITIAL integer
  - Definition
    - It specifies the size of physical storage space which is initially allocated when creating the ind ex.
    - This size is aligned to the EXTENT size of the TABLESPACE to which the table belongs. (e.g. If the EXT size is 8192 bytes, 'INITIAL 100' is actually operated as 8192 bytes.)
    - The size (aligned to the EXTENT size of TABLESPACE) should be equal to or bigger than MIN EXTENTS, or it should be equal to or less than MAXEXTENTS.
  - The minimum value is 1, and the maximum value depends on the system environment.
  - If it is omitted, the default value is one EXTENT size of TABLESPACE to which the table belongs.

- NEXT integer
  - Definition
    - It specifies the physical space size to be allocated when adding the space to the table.
    - This size is aligned to the EXTENT size of the TABLESPACE to which the table belongs. (e.g. If the EXT size is 8192 bytes, 'NEXT 100' is actually operated as 8192 bytes.)
    - NEXT operates as follows, depending on the remaining space size of the index available curr ently. (Obtained by subtracting the amount of currently used space from the MAXEXTENTS si ze)

    - If the remaining space size is 0, then it can not extend the space.
    - If the remaining space size is bigger than 0, but smaller than NEXT, then it allocates the

space as big as the remaining space.
- If the remaining space size is bigger than NEXT, then it allocates the space as big as the NEXT.
  - The minimum value is 1 and the maximum value depends on the system environment.
  - If it is omitted, the default value is one EXTENT size of TABLESPACE to which the index belongs.

- MINSIZE integer
  - Definition
    - It is the minimum space size of the index.
    - The value should be equal to or smaller than MAXSIZE.
  - This size is aligned to the EXTENT size of the TABLESPACE to which the index belongs.
  - The minimum value is 1 and the maximum value depends on the system environment.
  - If it is smaller than the size of two EXTENT, it is specified to the size of two EXTENT.
  - If it is omitted, the default value is the size of two EXTENT.

- MAXSIZE integer
  - Definition
    - It is the maximum space size of the index.
    - The value should be equal to or bigger than MINSIZE.
  - This size is aligned to the EXTENT size of the TABLESPACE to which the index belongs.
  - The minimum value is 1 and the maximum value depends on the system environment.
  - If it is omitted, the default value is 32 terabytes (35,184,372,088,832).
  - Even though the value is set to over 32 terabytes, it is adjusted and set to 32 terabytes.

〈size clause〉

It specifies the file size in bytes. (If the unit is omitted, the default value is bytes.)

- K: Kilobytes
- M: Megabytes
- G: Gigabytes
- T: Terabytes

NOPARALLEL | PARALLEL [integer]

It specifies the number of threads to be used when building an index.

- NOPARALLEL
  - It does not build an index in parallel.
- PARALLEL [integer]
  - It builds an index in parallel.
  - If an integer is omitted or set as 0, then it follows INDEX_BUILD_PARALLEL_FACTOR property.
  - The minimum value of an integer is 0 and the maximum value is 16.

    - If the property value is 0, then the system determines the optimal value.
- If it is omitted, the default value is PARALLEL.

TABLESPACE tablespace_name

It specifies the name of the tablespace in which the index is to be stored.

- When it specifies tablespace_name
    - if tablespace_name is data tablespace, then a LOGGING index is created.
    - if tablespace_name is temporary tablespace or nologging tablespace, then a NOLOGGING index i s created.

- When it omits TABLESPACE clause
    - If INDEX TABLESPACE tablespace_name of USER is specified
        - The defined tablespace is used.
    - If INDEX TABLESPACE of USER is NULL
        - The index of the DISK table uses the user's default data tablespace.
        - The index of the MEMORY table uses the user's default temporary tablespace.

Description

LOGGING index and NOLOGGING index have the following trade-offs.

- LOGGING index
    - Advantage: It does not separately build an index because the index is automatically restored by u sing the log when starting up the system.
    - Disadvantage: A disk I/O occur because the changes on the index is recorded on the log when alt ering the row.
- NOLOGGING index
    - Advantage: A disk I/O does not occur for the changes on the index when altering the row.
    - Disadvantage: It automatically rebuilds the index when starting up the system because the log inf ormation of the index does not exist.

Examples

The following is an example of creating the unique index.

```
gSQL> CREATE UNIQUE INDEX idx_t1_id ON t1( id );
Index created.
```

The following is an example of creating an index for multiple columns.

```
gSQL> CREATE INDEX idx_t1_id_name ON t1( id, name );
Index created.
```

The following is an example of specifying the sort order of the index column.

```
gSQL> CREATE INDEX idx_t1_dept_id ON t1( dept_id DESC );
Index created.
```

The following is an example of specifying the sort order of NULL value of the index column.

```
gSQL> CREATE INDEX idx_t1_name ON t1( name NULLS FIRST );
Index created.
```

The following is an example of setting information about the space in which index is stored.

```
gSQL> CREATE INDEX idx_t1_id ON t1( id )
             STORAGE ( INITIAL 10M NEXT 1M MINSIZE 10M MAXSIZE 100M );
Index created.
```

The following is an example of creating the redo logging for the index.

```
gSQL> CREATE INDEX idx_t1_id ON t1( id );
Index created.
```

The following is an example of creating the index with parallel option.

```
gSQL> CREATE INDEX idx_t1_name ON t1( name ) PARALLEL;
Index created.
```

The following is an example of specifying the tablespace when creating an index.

```
gSQL> CREATE INDEX idx_t1_name ON t1( name ) TABLESPACE mem_temp_tbs;
Index created.
```

Compatibility

The SQL standard does not cover the concepts of the index.

For more information

Refer to **DROP INDEX**.

19.11 CREATE MEMORY DATA TABLESPACE

Function

It defines a memory data tablespace.

Syntax

```
<memory data tablespace statement> ::=
    CREATE [ MEMORY ] [ DATA ] TABLESPACE tablespace_name
        DATAFILE <memory datafile clause> [, ...]
        [ <data tablespace management clause> [, ...] ]
<memory datafile clause> ::=
     'filename'
        [ SIZE <size clause> ¦ REUSE ¦ SIZE <size clause> REUSE ]
        [ AT <domain_name> ]
<size clause> ::=
    integer [ K ¦ M ¦ G ¦ T ]
<data tablespace management clause> ::=
      { ONLINE ¦ OFFLINE }
    ¦ EXTSIZE <size clause>
```

Invocation and Access Rules

CREATE TABLESPACE ON DATABASE privilege is required to perform <memory data tablespace definition>
.

The user who performed the statement has CREATE OBJECT ON TABLESPACE privilege on the created ta
blespace.

One of the following privileges is required to create the objects on the created tablespace.

- CREATE OBJECT ON TABLESPACE for that tablespace
- USAGE TABLESPACE ON DATABASE

Syntax Rules and Parameters

[MEMORY] [DATA]

It is a memory tablespace to store the permanent objects such as tables, indexes, etc.
The reserved words, MEMORY and DATA, can be omitted.

tablespace_name

It is the tablespace name to be created.
The length of the tablespace name should be shorter than 128 bytes.

〈memory datafile clause〉

- 'filename'
  - It is the file name to store and manage the data.
  - It is the space to store the checkpoint image for the memory data.
  - filename can be either a new file or an already existing file.
  - The length of the filename should be shorter than 1024 bytes.

- SIZE 〈size clause〉
  - The initial size is assigned for a new file by using the SIZE clause.
  - An error occurs if the file already exists.
  - The file size can be specified between minimum 1M and maximum 30G.

- REUSE
  - If the file already exists, REUSE clause is used.
  - If the file does not exist, a new file is created.
  - The newly created file size is
    - determined by USER_DATA_TABLESPACE_SIZE property in case of the data tablespace.
    - determined by USER_TEMP_TABLESPACE_SIZE property in case of the temporary tablespace.

- SIZE 〈size clause〉 REUSE
  - If both SIZE clause and REUSE clause are specified, it is operated as follows according to the pres ence of the filename.
    - For the new filename, the initial file size is assigned by using the SIZE clause.
    - For the existing filename, the size is adjusted to the value of SIZE clause by using the existing file.

〈size clause〉

It specifies the file size in bytes. (If it is omitted, the default unit is bytes.)

- K: Kilobytes
- M: Megabytes
- G: Gigabytes
- T: Terabytes

〈domain name〉

It is a name of a member or a group for which the statement is performed.
If it is omitted, it is performed for all groups.

ONLINE | OFFLINE

It sets ONLINE or OFFLINE of the tablespace.

- ONLINE is the state which a tablespace can be used as soon as it is created.
- OFFLINE is the state which a tablespace is unable to be used, it can be used after switching to ONLIN E state.

EXTSIZE 〈size clause〉

It specifies extent size of the tablespace.

- The extent size is specified in bytes, and one of the six (64 K, 128 K, 256 K, 512 K, 1 M, 2 M) is select ed.
- If the extent size is defined as a value between 64 K ~ 128 K, 128 K is set. If the extent size is defined as 2 M or bigger, 2 M is set.

Description

The data tablespace is an object which provides the physical space to store the SQL schema object such a s a table, an index (LOGGING).

Examples

The following is an example of creating a memory data tablespace.

```
gSQL> CREATE TABLESPACE space1 DATAFILE 'test_file_1.dbf' SIZE 10M REUSE;
Tablespace created.
```

The following is an example of creating a tablespace which consists of multiple data files.

```
gSQL> CREATE TABLESPACE space1
             DATAFILE 'test_file_3_1.dbf' SIZE 10M REUSE,
                      'test_file_3_2.dbf' SIZE 10M REUSE;
Tablespace created.
```

Compatibility

The SQL standard does not cover the concepts of the tablespace.

For More Information

Refer to the followings.

- DROP TABLESPACE
- ALTER TABLESPACE

19.12 CREATE MEMORY TEMPORARY TABLESPACE

Function

It defines a memory temporary tablespace.

Syntax

```
<memory temporary tablespace statement> ::=
    CREATE [ MEMORY ] TEMPORARY TABLESPACE tablespace_name
        MEMORY <memory clause> [, ...]
        <temporary tablespace management clause>
<memory clause>
     'memory_name' { SIZE <size clause> } [ AT <domain_name> ]
<temporary tablespace management clause> ::=
    EXTSIZE <size clause>
```

Invocation and Access Rules

CREATE TABLESPACE ON DATABASE privilege is required to perform <memory temporary tablespace definition>.

The user who performed the statement has CREATE OBJECT ON TABLESPACE privilege on the created tablespace.

One of the following privileges is required to create the objects on the created tablespace.

- CREATE OBJECT ON TABLESPACE for the tablespace.
- USAGE TABLESPACE ON DATABASE

Syntax Rules and Parameters

[MEMORY] TEMPORARY

It is a memory temporary tablespace to store the no logging indexes or the temporary objects such as inte rmediate results which are generated during the query processing.
The reserved word, MEMORY, can be omitted.

tablespace_name

It is the tablespace name to be created.
The length of the tablespace name should be shorter than 128 bytes.

〈memory clause〉

- 'memory_name'
  - It is a memory name to store the temporary data.
  - memory_name should be guaranteed to be unique within the tablespace.
  - The length of the memory_name should be shorter than 1024 bytes.
- SIZE 〈size clause〉
  - It specifies the initial size.
  - It can be specified between minimum 1M and maximum 30G.

〈size clause〉

It specifies the size of shared memory space in bytes.(If it is omitted, the default unit is bytes.)
The image is not managed as a file in case of the temporary memory data.

- K: Kilobytes
- M: Megabytes
- G: Gigabytes
- T: Terabytes

〈domain name〉

It is a name of a member or a group for which the statement is performed.
If it is omitted, it is performed for all groups.

EXTSIZE <size clause>

It specifies extent size of the tablespace.

- The extent size is specified in bytes, and one of the six (64 K, 128 K, 256 K, 512 K, 1 M, 2 M) is select ed.
- If the extent size is defined as a value between 64 K ~ 128 K, 128 K is set. If the extent size is defined as 2 M or bigger, 2 M is set.

Description

The temporary tablespace is an object which provides the physical space to store the SQL schema object s uch as an index (NOLOGGING), and to store the intermediate results for sorting, hashing during the quer y processing.

Examples

The following is an example of creating a temporary tablespace.

```
gSQL> CREATE TEMPORARY TABLESPACE temp_space1 MEMORY 'test_memory_1' SIZE 10M;
Tablespace created.
```

The following is an example of creating a temporary tablespace which includes multiple memory spaces.

```
gSQL> CREATE TEMPORARY TABLESPACE temp_space1
              MEMORY 'test_memory_3_1' SIZE 10M,
                     'test_memory_3_2' SIZE 10M;
Tablespace created.
```

Compatibility

The SQL standard does not cover the concepts of the tablespace.

For More Information

Refer to the followings.

- DROP TABLESPACE
- ALTER TABLESPACE

19.13 CREATE PROFILE

Function

It is the statement which creates the profile, and it sets the password management method.
When a profile is allocated to a user, the user's password is managed in the way defined in the profile.

Syntax

```
<profile definition> ::=
    CREATE PROFILE profile_name LIMIT
    { <password_parameters>, ...}
    ;
<password parameters> ::=
      FAILED_LOGIN_ATTEMPTS { integer ¦ UNLIMITED ¦ DEFAULT }
    ¦ PASSWORD_LOCK_TIME  { password_parameter_number_interval ¦ UNLIMITED ¦ DEFAULT }
    ¦ PASSWORD_LIFE_TIME  { password_parameter_number_interval ¦ UNLIMITED ¦ DEFAULT }
    ¦ PASSWORD_GRACE_TIME { password_parameter_number_interval ¦ UNLIMITED ¦ DEFAULT }
    ¦ PASSWORD_REUSE_MAX  { integer ¦ UNLIMITED ¦ DEFAULT }
    ¦ PASSWORD_REUSE_TIME { password_parameter_number_interval ¦ UNLIMITED ¦ DEFAULT }
    ¦ PASSWORD_VERIFY_FUNCTION { <verify_policy> ¦ NULL ¦ DEFAULT }
<verify_policy> ::=
      KISA_VERIFY_FUNCTION
    ¦ ORA12C_VERIFY_FUNCTION
    ¦ ORA12C_STRONG_VERIFY_FUNCTION
    ¦ VERIFY_FUNCTION_11G
    ¦ VERIFY_FUNCTION
<password_parameter_number_interval> ::=
   integer
 ¦ integer / integer
```

Invocation and Access Rules

CREATE PROFILE ON DATABASE privilege is required to perform <profile definition>.

Syntax Rules and Parameters

profile_name

It specifies the profile name to be created.

password_parameters

It sets the parameters for password management.

- The following parameters are to be set.
  - FAILED_LOGIN_ATTEMPTS
  - PASSWORD_LOCK_TIME
  - PASSWORD_LIFE_TIME
  - PASSWORD_GRACE_TIME
  - PASSWORD_REUSE_MAX
  - PASSWORD_REUSE_TIME
  - PASSWORD_VERIFY_FUNCTION

The omitted parameters follows the "DEFAULT" profile policy.

FAILED_LOGIN_ATTEMPTS

It sets the number of consecutive login attempts allowed to fail.
If the failed attempts exceed the specified number, the account is locked.

- FAILED_LOGIN_ATTEMPTS integer
  - The value range should be a positive integer bigger than 0.
- FAILED_LOGIN_ATTEMPTS UNLIMITED
  - Account lockout which is due to a login failure does not occur.
- FAILED_LOGIN_ATTEMPTS DEFAULT
  - It follows the "DEFAULT" profile policy.

PASSWORD_LOCK_TIME

It sets the period (days) which the account is locked after consecutive login failure.

- PASSWORD_LOCK_TIME constant_expression
  - It is the lock duration (day).
  - The default unit is a day.
  - Hours(n/24), minutes (n/1440), seconds (n/86400) can be specified for testing.

    - The value range is from one second (1/86400) to 100,000 days.
- PASSWORD_LOCK_TIME UNLIMITED
    - If an account lockout occurs, the lock is not release until performing ALTER USER user_name AC COUNT UNLOCK statement.
- PASSWORD_LOCK_TIME DEFAULT
    - It follows the "DEFAULT" profile policy.

PASSWORD_LIFE_TIME

It sets the life time of the password (day).

- PASSWORD_LIFE_TIME constant_expression
    - It is the life time of the password (day).
    - The default unit is a day.
    - Hours(n/24), minutes (n/1440), seconds (n/86400) can be specified for testing.
    - The value range is from one second (1/86400) to 100,000 days.
- PASSWORD_LIFE_TIME UNLIMITED
    - The password does not have the expiration date.
- PASSWORD_LIFE_TIME DEFAULT
    - It follows the "DEFAULT" profile policy.

PASSWORD_GRACE_TIME

It sets the grace period of password expiration when logging in after PASSWORD_LIFE_TIME.

- PASSWORD_GRACE_TIME constant_expression
    - The grace period for password expiration (day)
    - The default unit is a day.
    - Hours (n/24), minutes (n/1440), seconds (n/86400) can be specified for testing.
    - The value range is from one second (1/86400) to 100,000 days.
- PASSWORD_GRACE_TIME UNLIMITED
    - It continues to defer the password expiration.
- PASSWORD_GRACE_TIME DEFAULT
    - It follows the "DEFAULT" profile policy.

PASSWORD_GRACE_TIME starts at first login trial after the password life time. If the password is not alte red during the grace period, the password expires.

PASSWORD_REUSE_MAX

It sets the number of the recent passwords which can not be reused when the user wants to reuse the old password.

PASSWORD_REUSE_MAX should be used together with PASSWORD_REUSE_TIME.

- PASSWORD_REUSE_MAX integer
  - The value range should be a positive integer bigger than 0.
- PASSWORD_REUSE_MAX UNLIMITED
  - If PASSWORD_REUSE_TIME is UNLIMITED, all old passwords can be reused.
  - If PASSWORD_REUSE_TIME is not UNLIMITED, any old password can not be reused.
- PASSWORD_REUSE_MAX DEFAULT
  - It follows the "DEFAULT" profile policy.

PASSWORD_REUSE_TIME

It sets the duration which the password can not be reused when the user wants to reuse the old password.

PASSWORD_REUSE_TIME should be used together with PASSWORD_REUSE_MAX.

- PASSWORD_REUSE_TIME constant_expression
  - It is the duration which the password can not be reused. (day)
  - The default unit is a day.
  - Hours(n/24), minutes (n/1440), seconds (n/86400) can be specified for testing.
  - The value range is from one second (1/86400) to 100,000 days.
- PASSWORD_REUSE_TIME UNLIMITED
  - If PASSWORD_REUSE_MAX is UNLIMITED, all old passwords can be reused.
  - If PASSWORD_REUSE_MAX is not UNLIMITED, any old password can not be reused.
- PASSWORD_REUSE_TIME DEFAULT
  - It follows the "DEFAULT" profile policy.

PASSWORD_VERIFY_FUNCTION

It sets the password complexity verification methods.

- PASSWORD_VERIFY_FUNCTION null
  - The password complexity verification is not performed.
- PASSWORD_VERIFY_FUNCTION DEFAULT
  - It follows the "DEFAULT" profile policy.
- PASSWORD_VERIFY_FUNCTION <verify policy>
  - The password complexity verification methods can be specified as follows.

- KISA_VERIFY_FUNCTION
- ORA12C_VERIFY_FUNCTION
- ORA12C_STRONG_VERIFY_FUNCTION
- VERIFY_FUNCTION_11G
- VERIFY_FUNCTION

KISA_VERIFY_FUNCTION

It is the password verification method of KISA (Korea Internet & Security Agency).

- 8 or more letters
- One or more characters
- One or more numbers
- One or more special characters

ORA12C_VERIFY_FUNCTION

It is the password verification method of Oracle, ORA 12C VERIFY_FUNCTION.

- 8 or more letters
- 1 or more characters
- 1 or more numbers
- The database name should not be included.
- The username or the reversed username should not be included.
- *goldilocks* should not be included.
- *oracle* should not be included.
- The following simple passwords are not allowed.
  - welcome1, database1, account1, user1234, password1, oracle123, computer1, abcdefg1, change_on_intall
- At least 3 characters of the new password should be different from the old password.

ORA12C_STRONG_VERIFY_FUNCTION

It is the password verification method of Oracle, ORA12C_STRONG_VERIFY_FUNCTION.

- 9 or more letters
- 2 or more uppercases
- 2 or more lowercases
- 2 or more numbers
- 2 or more special characters
- At least 4 characters of the new password should be different from the old password.

VERIFY_FUNCTION_11G

It is the password verification method of Oracle, VERIFY_FUNCTION_11G.

- 8 or more letters
- 1 or more characters
- 1 or more numbers
- The username should not be included.
- At least 3 characters of the new password should be different from the old password.

VERIFY_FUNCTION

It is the password verification method of Oracle, VERIFY_FUNCTION.

- It should not be as same as the username.
- 4 or more letters
- 1 or more characters
- 1 or more numbers
- 1 or more special characters
- The following simple passwords are not allowed.
  - welcome, database, account, user, password, oracle, computer, abcd
- At least 3 characters of the new password should be different from the old password.

Description

Account Lockout

The followings are the parameters affecting the account lockout.

- FAILED_LOGIN_ATTEMPTS
- PASSWORD_LOCK_TIME

For example, when a user and a profile are created as follows

```
CREATE PROFILE prof LIMIT
    FAILED_LOGIN_ATTEMPTS 4
    PASSWORD_LOCK_TIME 30;
ALTER USER u1 PROFILE prof;
```

If the user u1 fails to log in more than four times, the account is locked for 30 days. Then the account loc kout is released after 30 days.

If PASSWORD_LOCK_TIME is UNLIMITED, the account lockout should be explicitly released by using ALTER USER statement.

```
ALTER USER user1 ACCOUNT UNLOCK;
```

Password Expiration

The followings are the parameters affecting the password expiration.

- PASSWORD_LIFE_TIME
- PASSWORD_GRACE_TIME

The password is expired in the following order.

1. The password is set.
   - The time of the password expiration is set to the time elapsed as much as PASSWORD_LIFE_TIME since when a password is altered.
   - The password expiration status is OPEN, and it allows a normal login.

2. When a user logs in after the password expiration
   - The login succeeds but the password expiration status becomes EXPIRED (GRACE) and the following warnings occur.
     - ERR-28000(16310): The password will expire in n days
     - ERR-28000(16311): The password will expire soon
     - The SQL standard does not cover the concepts of password expiration.
     - 28000 is the authentication warning or SQL standard status code of an error. 16310, 16311 are the GOLDILOCKS error code.
   - The time of the password expiration is reset to the time elapsed as much as PASSWORD_GRACE_TIME since when a user logged in.

3. When a user logs in after the grace time
   - The password expiration status becomes EXPIRED, the user can not log in, and the following error occurs.
     - ERR-28000(16312): The password has expired
     - The SQL standard does not cover the concepts of password expiration.
     - 28000 is the authentication warning or SQL standard status code of an error. 16312 is the GOLDILOCKS error code.
     - GOLDILOCKS internal error code of 16312 should be used to control password reentering using program.

Table 19-5 Password expiration status transition

Step	Point of time	login	Status
1	The password is changed.	Success	OPEN

Step	Point of time	login	Status
2	PASSWORD_LIFE_TIME is elapsed.	Success with warning	EXPIRED(GRACE)
3	PASSWORD_GRACE_TIME is elapsed.	Error	EXPIRED

The following is another example.

```
CREATE PROFILE prof LIMIT
   PASSWORD_LIFE_TIME 90
   PASSWORD_GRACE_TIME 3;
ALTER USER u1 PROFILE prof;
```

The example above describes that the user u1 succeeds in login after 90 days of password expiration. Ho wever, the user receives a warning that the password expires in three days.

If the password is not changed within three days, the password expires.
Once the password expires, it reminds that the new password should be entered when logging in, and th e account access is denied.

Password Reusability

The followings are the parameters affecting the password reusability.

- PASSWORD_REUSE_MAX
- PASSWORD_REUSE_TIME

The password reusability of the two parameters above are determined according to the following table.

Table 19-6 Conditions for the password reusability

PASSWORD_REUSE_MAX	PASSWORD_REUSE_TIME	Reusable condition
value	value	It can be reused when both conditions of PASSWORD_R EUSE_TIME and PASSWORD_REUSE_MAX are satisfied.
value	UNLIMITED	It can not be reused.
UNLIMITED	value	It can not be reused.
UNLIMITED	UNLIMITED	It can always be reused.

If a profile is created as follows,

```
CREATE PROFILE prof LIMIT
   PASSWORD_REUSE_MAX 5
   PASSWORD_REUSE_TIME 3;
```

the password can not be reused if it is the five recent passwords or the password changed within three d

ays.

The following is an example of the user u1's password change history. If the current password is P # _000 007 and the current date is 2015-08-08, the reusability of existing passwords is as follows.

Table 19-7 Examples of password reusability

Password	Password_date	Password reusability
P#_000001	2015-08-01	It can be reused.
P#_000002	2015-08-02	It can be reused.
P#_000003	2015-08-03	It violates REUSE_MAX.
P#_000004	2015-08-04	It violates REUSE_MAX.
P#_000005	2015-08-05	It violates REUSE_MAX, REUSE_TIME.
P#_000006	2015-08-06	It violates REUSE_MAX, REUSE_TIME.
P#_000007	2015-08-07	It violates REUSE_MAX, REUSE_TIME.

The password change history accumulated for checking the password reusability can be deleted by using the following statement.

```
ALTER DATABASE CLEAR PASSWORD HISTORY;
```

DEFAULT profile

When creating the database, the following "DEFAULT" profile is automatically created. The password par ameters of the "DEFAULT" profile are as follows.

Table 19-8 Configuration of DEFAULT profile

Parameter	Value
FAILED_LOGIN_ATTEMPTS	10
PASSWORD_LOCK_TIME	1
PASSWORD_LIFE_TIME	180
PASSWORD_GRACE_TIME	7
PASSWORD_REUSE_MAX	UNLIMITED
PASSWORD_REUSE_TIME	UNLIMITED
PASSWORD_VERIFY_FUNCTION	NULL

The default values of "DEFAULT" profile have the following characteristics.

- Account lockout
  - The account is locked for one day (PASSWORD_LOCK_TIME) after ten consecutive login failure (F AILED_LOGIN_ATTEMPTS).
- Password expiration
  - The password expires after the grace period of seven days (PASSWORD_GRACE_TIME) after 180

days (PASSWORD_LIFE_TIME) are exceeded.

- Password reusability
  - The old password can be reused.
- Password complexity verification
  - It is not performed.

DEFAULT profile can not be dropped but it can be altered as follows.

```
ALTER PROFILE DEFAULT LIMIT ...
```

Examples

The following is an example of creating the profile to control the account lockout. The account is locked f or three days after three consecutive login failures.

```
gSQL> CREATE PROFILE prof1 LIMIT
        FAILED_LOGIN_ATTEMPTS 3
        PASSWORD_LOCK_TIME 3;
Profile created.
gSQL> COMMIT;
Commit complete.
```

The following is an example of creating the profile to control the password expiration. The life time of th e password is 90 days, and the grace time of the password is seven days.

```
gSQL> CREATE PROFILE prof1 LIMIT
        PASSWORD_LIFE_TIME 90
        PASSWORD_GRACE_TIME 7;
Profile created.
gSQL> COMMIT;
Commit complete.
```

The following is an example of creating the profile to control whether the password is reusable. The follo wing example does not verify the old password when changing the password.

```
gSQL> CREATE PROFILE prof1 LIMIT
        PASSWORD_REUSE_MAX  DEFAULT
        PASSWORD_REUSE_TIME DEFAULT;
Profile created.
gSQL> COMMIT;
```

```
Commit complete.
```

The following is an example of creating the profile to control the password complexity check.

```
gSQL> CREATE PROFILE prof1 LIMIT
        PASSWORD_VERIFY_FUNCTION KISA_VERIFY_FUNCTION;
Profile created.
gSQL> COMMIT;
Commit complete.
```

The following is an example of creating the profile by setting all parameter.

```
gSQL> CREATE PROFILE prof1 LIMIT
        FAILED_LOGIN_ATTEMPTS 3
        PASSWORD_LOCK_TIME 3
        PASSWORD_LIFE_TIME 90
        PASSWORD_GRACE_TIME 7
        PASSWORD_REUSE_MAX  DEFAULT
        PASSWORD_REUSE_TIME DEFAULT
        PASSWORD_VERIFY_FUNCTION KISA_VERIFY_FUNCTION;
Profile created.
gSQL> COMMIT;
Commit complete.
```

Compatibility

The SQL standard does not cover the concepts of the profile.

For More Information

Refer to the followings.

- DROP PROFILE
- ALTER PROFILE
- CREATE USER
- ALTER USER
- ALTER DATABASE CLEAR PASSWORD HISTORY

19.14 CREATE SCHEMA

Function

It defines the schema.

Syntax

```
<schema definition> ::=
    CREATE SCHEMA <schema name clause>
        [ <schema element> [...] ]
    ;
<schema name clause> ::=
      schema_name
    ¦ AUTHORIZATION user_identifier
    ¦ schema_name AUTHORIZATION user_identifier
<schema element> ::=
      <table definition>
    ¦ <view definition>
    ¦ <index definition>
    ¦ <sequence generator definition>
    ¦ <grant privilege statement>
    ¦ <comment statement>
```

Invocation and Access Rules

The user should satisfy the following conditions to perform <schema definition>.

- CREATE SCHEMA ON DATABASE privilege is required to create the schema.
- If <schema element> exists, the privilege to perform each <schema element> is required.
  For more information about the access privilege, refer to *invocation and access rules* in the following statements.
  - **CREATE TABLE**
  - **CREATE VIEW**
  - **CREATE INDEX**

    - **CREATE SEQUENCE**
    - **GRANT privileges TO**
    - **COMMENT ON name IS**

- The user who is user_identifier has the following privileges for the created schema.
    - The owner of the created schema, which is schema_name
    - The owner of the object which is created by <schema element> clause

- An appropriate privilege for the schema is required to create an object because a separate privilege o n the created schema is not granted.
  For more information about schema privilege types, refer to **<schema privilege>** of GRANT privileges TO statement.
  For more information about usage example, refer to **Examples** of CREATE USER statement.

Syntax Rules and Parameters

schema_name

It is the schema name to be created.
An identical schema name should not exist in the database.
The length of the schema name should be shorter than 128 bytes.

AUTHORIZATION user_identifier

If the schema name is omitted, a schema with the same name as the user_identifier is created.
If the AUTHORIZATION is not specified, the user_identifier of the user performing the statement is used.

schema_name AUTHORIZATION user_identifier

It specifies the schema name and schema owner to be created.
The owner can not be a role or PUBLIC.

<schema element>

It defines the objects to be created in the schema along with the schema creation.
The schema_element is executed in the listed order, and they are separated by a white space without a c omma (,).
The object can not be defined in a schema which has different name from the schema to be created.

- The <grant privilege statement> can be specified only for the following privileges.
  - <schema privilege>
  - <table privilege>
  - <sequence privilege>

- The <comment statement> can be specified only for the following objects.
  - SCHEMA schema_name
  - TABLE [schema_name].table_name
  - COLUMN [schema_name].table_name.column_name
  - INDEX [schema_name].index_name
  - SEQUENCE [schema_name].sequence_name
  - CONSTRAINT [schema_name].constraint_name

Description

A schema is an object which logically classifies SQL schema objects such as table, view, index, sequence a nd constraint.

In GOLDILOCKS, the relationship between the user and the schema is 1 : N. In other words, the schema o wned by a user may not exist, or the user may own multiple schemas.

The SQL standard does not explicitly define the relationship of the non-schema objects such as user, sche ma, database, but each DBMS defines the relationship between the non-schema objects as a different co ncept. Refer to the following note.

The relationship between user and schema in other DBMS

- Oracle
  - User : schema = 1 : 1

- DB2
  - User : schema = 1 : N

- Postgres
  - User : schema = 1 : N

- MySQL
  - Database : schema = 1 : 1
  - User is a subordinate object of a database (schema).

Examples

The following is an example of creating a schema.

```
gSQL> CREATE SCHEMA s1;
Schema created.
```

The following is an example of creating a schema and assigning the schema owner.

```
gSQL> CREATE SCHEMA s1 AUTHORIZATION test;
Schema created.
```

The following is an example of creating a schema together with the objects which belong to the schema.

```
gSQL> CREATE SCHEMA s1
              CREATE TABLE t1 ( id INTEGER, name VARCHAR(128) )
              CREATE INDEX idx_t1_id ON t1 ( id )
              COMMENT ON TABLE t1 IS 'comment on s1.t1'
;
Schema created.
```

Compatibility

Table 19-9 SQL standard compatibility

Feature ID	Description	Compatibility
S071	SQL paths in function and type name resolution	X
F461	Named character sets	X
F171	Multiple schemas per user	O
T332	Extended roles	X

For More Information

Refer to the followings.

- DROP SCHEMA
- CREATE USER
- CREATE TABLE
- CREATE VIEW
- CREATE INDEX
- CREATE SEQUENCE
- GRANT privileges TO
- COMMENT ON name IS

19.15 CREATE SEQUENCE

Function

It creates a sequence.

Syntax

```
<sequence generator definition> ::=
    CREATE SEQUENCE [schema_name.] sequence_name
        [ <sequence generator option> [, ...] ]
    ;
<sequence generator option> ::=
      <sequence generator start with option>
    ¦ <basic sequence generator option>
<sequence generator start with option> ::=
    START WITH integer
<basic sequence generator option> ::=
      <sequence generator increment by option>
    ¦ <sequence generator maxvalue option>
    ¦ <sequence generator minvalue option>
    ¦ <sequence generator cycle option>
    ¦ <sequence generator cache option>
<sequence generator increment by option> ::=
    INCREMENT BY integer
<sequence generator maxvalue option> ::=
      MAXVALUE integer
    ¦ (NO MAXVALUE ¦ NOMAXVALUE)
<sequence generator minvalue option> ::=
      MINVALUE integer
    ¦ (NO MINVALUE ¦ NOMINVALUE)
<sequence generator cycle option> ::=
      CYCLE
    ¦ (NO CYCLE ¦ NOCYCLE)
<sequence generator cache option> ::=
      CACHE integer
    ¦ (NO CACHE ¦ NOCACHE)
```

Invocation and Access Rules

The user should have one of the following privileges to perform 〈sequence generator definition〉 statement.

- (CREATE SEQUENCE or CONTROL SCHEMA) ON SCHEMA for the schema to which the sequence belongs
- CREATE ANY SEQUENCE ON DATABASE

The sequence owner is determined as follows.

- The owner of the schema to which the sequence belongs
- If the schema to which the sequence belongs is PUBLIC, then it is the user who executed the statement.

The sequence owner has USAGE ON SEQUENCE WITH GRANT OPTION privilege.

One of the following privileges is required to use the created sequence.

- USAGE ON SEQUENCE for the sequence
- (USAGE SEQUENCE or CONTROL SCHEMA) ON SCHEMA for the schema to which the sequence belongs
- USAGE ANY SEQUENCE ON DATABASE

Syntax Rules and Parameters

sequence_name

It is the sequence name to be created, and it should be a unique name within the schema.
The schema to which the sequence belongs, such as schema_name.sequence_name, can be defined. If schema_name is omitted, the default schema name of the user performing the statement is used.
The length of the sequence name should be shorter than 128 bytes.

〈sequence generator option〉

If any of 〈sequence generator option〉 is not used, the following two statements have the same meaning.

- CREATE SEQUENCE test_seq;
- CREATE SEQUENCE test_seq START WITH 1 INCREMENT BY 1 NO MINVALUE NO MAXVALUE NO CYCLE CACHE 20;

〈sequence generator start with option〉

It defines the first sequence number to be generated.
Depending on the ascending or the descending order, it has the following features.

- Ascending sequence (INCREMENT BY a positive number)
  - It is used when starting the sequence with bigger sequence value than the minimum value.
  - If START WITH clause is omitted, the default value is the minimum value (MINVALUE value).
- Descending sequence (INCREMENT BY a negative number)
  - It is used when starting the sequence with smaller sequence value than the maximum value.
  - If START WITH clause is omitted, the default value is the maximum value (MAXVALUE value).

〈sequence generator increment by option〉

It defines the interval of sequence numbers.
The constraints and features are as follows.

- A positive number or a negative number is allowed, but 0 is not allowed.
- The absolute value of the interval should be smaller than the difference between MINVALUE and MA XVALUE.
- The ascending sequence is generated if it is a positive number, and the descending sequence is gener ated if it is a negative number.
- If INCREMENT BY clause is omitted, the default is a positive number 1.

〈sequence generator maxvalue option〉

It defines the maximum value of which the sequence can generate.

- MAXVALUE integer
  - The maximum value range is between the minimum (-9,223,372,036,854,775,808) and the max imum (+9,223,372,036,854,775,807) of the 64 bit integer.
  - It should be equal to or bigger than START WITH value, and bigger than MINVALUE value.
- NO MAXVALUE | NOMAXVALUE
  - The maximum value is defined as follows.
    - If it is an ascending sequence, it is the maximum value (+9,223,372,036,854,775,807) of the 64 bit integer.
    - If it is a descending sequence, it is -1.
  - NO MAXVALUE(SQL standard) and NOMAXVALUE are the reserved words with the same meani ng, so either of them can be used.
- If MAXVALUE and NO MAXVALUE are not specified, the default value is NO MAXVALUE.

〈sequence generator minvalue option〉

It defines the minimum value of which the sequence can generate.

- MINVALUE integer
  - The minimum value range is between the minimum (-9,223,372,036,854,775,808) and the maximum (+9,223,372,036,854,775,807) of the 64 bit integer.
  - It should be equal to or smaller than START WITH value, and smaller than MAXVALUE.
- NO MINVALUE | NOMINVALUE
  - The minimum value is defined as follows.
    - If it is an ascending sequence, it is 1.
    - If it is a descending sequence, it is the minimum value (-9,223,372,036,854,775,808) of the 64 bit integer.
  - NO MINVALUE(SQL standard) and NOMINVALUE are the reserved words with the same meaning, so either of them can be used.
- If MINVALUE and NO MINVALUE are not specified, the default value is NO MINVALUE.

〈sequence generator cycle option〉

It specifies whether to continue generating a value when the sequence value becomes the maximum value or the minimum value.

- CYCLE
  - When the ascending sequence becomes the maximum value, it generates the value again from the minimum value.
  - When the descending sequence becomes the minimum value, it generates the value again from the maximum value.
- NO CYCLE | NOCYCLE
  - When the sequence value becomes the maximum value or the minimum value, it does not generate a sequence value.
  - NO CYCLE(SQL standard) and NOCYCLE are the reserved words with the same meaning, so either of them can be used.
- If CYCLE and NO CYCLE are not specified, the default value is NO CYCLE.

〈sequence generator cache option〉

For quick access of a sequence, it defines the number of the sequence values to be preloaded in memory. When restarting database, the sequence values loaded in memory are lost, and it starts from the value since being loaded.

- CACHE integer
  - CACHE value should be equal to or bigger than 2.

    - CACHE value should not be bigger than CYCLE length, if CYCLE exists.
        - CYCLE length: CEIL(MAXVALUE - MINVALUE) / ABS(INCREMENT)
- NO CACHE | NOCACHE
    - The sequence values are not preloaded in memory.
- If CACHE and NO CACHE are not specified, the default value is CACHE 20.

Description

The sequence values of the created sequence objects are used by using **NEXTVAL** and **CURRVAL** function s.

The sequence value does not have a transaction property. The sequence value maintains the most recent value, even when an error occurs in the SQL statement in which the sequence function is used or when e xplicit ROLLBACK is performed.

CURRVAL function returns NEXTVAL value from the most recent call by a session.
Therefore, using this feature, the sequence value obtained by NEXTVAL can still be usable in the other SQ L statements. However, when the session does not call NEXTVAL, using CURRVAL function generates an error.

Examples

The object seq1 without the defined sequence options is an ascending sequence of the same meanings a s the object seq2 in the following example.

```
gSQL> CREATE SEQUENCE seq1;
Sequence created.
gSQL> CREATE SEQUENCE seq2 START WITH 1 INCREMENT BY 1 NO MINVALUE NO MAXVALUE NO CYCLE CACHE
20;
Sequence created.
```

The following is an example of a sequence which generates an odd value.

```
gSQL> CREATE SEQUENCE seq1 START WITH 1 INCREMENT BY 2;
Sequence created.
```

The following is an example of generating sequence which repeatedly generates an even number starting from 0 to 1000.

```
gSQL> CREATE SEQUENCE seq1 START WITH 0 MINVALUE 0 MAXVALUE 1000 INCREMENT BY 2 CYCLE;
Sequence created.
```

The following is an example of generating a descending sequence starting from -1.

```
gSQL> CREATE SEQUENCE seq1 INCREMENT BY -1;
Sequence created.
```

Compatibility

The SQL standard does not define <sequence generator cache option> clause.

Table 19-10 SQL standard compatibility

Feature ID	Description	Compatibility
T176	Sequence generator support	O

For More Information

Refer to the followings.

- DROP SEQUENCE
- ALTER SEQUENCE
- NEXTVAL
- CURRVAL

19.16 CREATE SYNONYM

Function

It creates a synonym. A synonym is an alternative name for a table, view, sequence, or another synonym, and it can be used in the following statements.

- DML: SELECT, INSERT, UPDATE, DELETE, LOCK TABLE, CALL
- DDL: GRANT, REVOKE, COMMENT

Syntax

```
<table definition> ::=
    CREATE [OR REPLACE] [PUBLIC] SYNONYM [schema_name.]synonym_name
    FOR [schema_name.]object_name
    ;
```

Invocation and Access Rules

The user should satisfy the following conditions to perform <synonym definition> statement.

- CREATE PUBLIC SYNONYM ON DATABASE privilege is required to create public synonym by explicitly specifying PUBLIC.

- The owner of public synonym is PUBLIC. The user who created the synonym does not have any privile ge.

- One of the following privileges is required to create the private synonym.
  - (CREATE SYNONYM or CONTROL SCHEMA) ON SCHEMA for that schema
  - CREATE ANY SYNONYM ON DATABASE

- The private synonym owner is determined as follows.
  - The owner of the schema to which the private synonym belongs
  - If the schema to which the private synonym belongs is PUBLIC, then it is the user who executed t he statement.

- If the user does not have privilege on the base object, the user is not allowed to execute the stateme

nt using its synonym even when the user created the synonym.

- Be cautious when allowing the privilege on the synonym because it means allowing privilege on the b ase object to which the synonym indicates.

Syntax Rules and Parameters

[OR REPLACE]

It replaces the existing synonym If the synonym already exists.

[PUBLIC]

It is specified when creating public synonym.
If it is omitted, private synonym is created.

synonym_name

It is the synonym name to be created, and it should be a unique name within the schema.
The schema to which the synonym belongs, such as schema_name.synonym_name, can be defined. If sch ema_name is omitted, default schema name of the user performing the statement is used.
The length of the synonym name should be shorter than 128 bytes.
Public synonym is a non-schema object. Therefore, a schema name can not be specified when creating p ublic synonym by explicitly specifying PUBLIC.

object_name

The schema to which the object belongs, such as schema_name.object_name, can be defined. If schema _name is omitted, default schema name of the user performing the statement is used.

The object types which can specify the object_name are as follows.

- Table
- View
- Sequence
- Another synonym

Existence of the target object, cycle check and privilege check are performed when executing the statem ent using the synonym.

Description

A synonym is an alternative name for a table, view, sequence, or another synonym.

If synonym is created, the applications does not need to be modified even when the base object is changed instead only the synonyms should be redefined. Therefore, it is convenient.
Also, the database security is improved by hiding the objects' real names and their schemas, and the usability is enhanced by changing the object's long name to a shorter name.

The synonym is literally an alternative name so creating the synonym does not mean that the synonym can be used to access the object. The proper privilege is required to access the object.

When executing the statement using the synonym, the object access procedure is as follows.

1. Find a table of the corresponding name.
2. If the table does not exist, find the private synonym of the corresponding name.
3. If the private synonym does not exist, find the public synonym of the corresponding name.

```
gSQL> CREATE PUBLIC SYNONYM syn1 FOR u1.t1;
Synonym created.
gSQL> CREATE PUBLIC SYNONYM syn2 FOR syn1;
Synonym created.
gSQL> SELECT * FROM syn2;
```

The example above is the object access procedure in SELECT statement.

1. It searched for the table syn2, but it does not exist.
2. It searched for the private synonym syn2, but it does not exist.
3. It searched for the public synonym syn2, and it exists.
    I. It searched for the table syn1, but it does not exist.
    II. It searched for the private synonym syn1, but it does not exist.
    III. It searched for the public synonym syn2, and it exists.
        i. It searched for the table u1.t1, and it exists.

Examples

The following is an example of creating a private synonym.

```
gSQL> CREATE SYNONYM MyEmp FOR branch.Employee;
Synonym created.
```

```
gSQL> SELECT * FROM MyEmp;
```

The following is an example of creating a public synonym.

```
gSQL> CREATE PUBLIC SYNONYM MainEmp FOR main.Employee;
Synonym created.
gSQL> SELECT * FROM MainEmp;
```

Compatibility

The SQL standard does not define the CREATE SYNONYM statement.

For More Information

Refer to **DROP SYNONYM**.

19.17 CREATE TABLE

Function

It defines a table.

Syntax

```
<table definition> ::=
    CREATE TABLE table_name
        ( <table element> [, ...] )
        [ <table sharding strategy> ]
        [ <table attribute clause> [...] ]
        [ TABLESPACE tablespace_name ]
        [ <table global secondary index clause> ]
    ;
<table element> ::=
      <column definition>
    ¦ <table constraint definition>
<column definition> ::=
    column_name <data type>
        [ <default clause> ¦ <identity column specification> ]
        [ <column constraint definition> ]
<data type> ::=
      <character string type>
    ¦ <binary string type>
    ¦ <numeric type>
    ¦ <boolean type>
    ¦ <datetime type>
    ¦ <interval type>
<character string type> ::=
      CHARACTER [ ( integer [ <character length units> ] ) ]
    ¦ CHAR [ ( integer [ <character length units> ] ) ]
    ¦ CHARACTER VARYING ( integer [ <character length units> ] )
    ¦ CHAR VARYING ( integer [ <character length units> ] )
    ¦ VARCHAR ( integer [ <character length units> ] )
    ¦ CHARACTER LONG VARYING
```

```
    ¦ LONG VARCHAR

<character length units> ::=
      CHARACTERS
    ¦ CHAR
    ¦ OCTETS
    ¦ BYTE
<binary string type> ::=
      BINARY [ ( length ) ]
    ¦ BINARY VARYING ( length )
    ¦ VARBINARY ( length )
    ¦ LONG BINARY VARYING
    ¦ LONG VARBINARY
<numeric type> ::=
      <exact numeric type>
    ¦ <approximate numeric type>
    ¦ <native numeric type>
<exact numeric type> ::=
      NUMERIC [ ( precision [, scale ] ) ]
    ¦ SMALLINT
    ¦ INTEGER
    ¦ INT
    ¦ BIGINT
<approximate numeric type> ::=
    ¦ FLOAT [ ( precision ) ]
    ¦ REAL
    ¦ DOUBLE PRECISION
<native numeric type> ::=
      NATIVE_SMALLINT
    ¦ NATIVE_INTEGER
    ¦ NATIVE_BIGINT
    ¦ NATIVE_REAL
    ¦ NATIVE_DOUBLE
<boolean type> ::=
    BOOLEAN
<datetime type> ::=
      DATE
    ¦ TIME [ ( time_precision ) ] [ WITH TIME ZONE ¦ WITHOUT TIME ZONE ]
    ¦ TIMESTAMP [ ( timestamp_precision ) ] [ WITH TIME ZONE ¦ WITHOUT TIME ZONE ]
<interval type> ::=
    INTERVAL <interval qualifier>
```

```
<interval qualifier> ::=
      <non-second primary datetime field> [ ( interval_leading_field_precision ) ]
          TO { <non-second primary datetime field> ¦ SECOND [ (
interval_fractional_seconds_precision ) ] }
    ¦ <non-second primary datetime field> [ ( interval_leading_field_precision ) ]
    ¦ SECOND [ ( interval_leading_field_precision [, interval_fractional_seconds_precision ] )
]
<non-second primary datetime field> ::=
      YEAR
    ¦ MONTH
    ¦ DAY
    ¦ HOUR
    ¦ MINUTE
<default clause> ::=
    DEFAULT <default option>
<default option> ::=
      constant
    ¦ NULL
    ¦ expression
<identity column specification> ::=
    GENERATED { ALWAYS ¦ BY DEFAULT } AS IDENTITY
    [ ( <common sequence generator option> [, ...] ) ]
<common sequence generator option> ::=
      START WITH integer_constant
    ¦ <basic sequence generator option>
<basic sequence generator option> ::=
      INCREMENT BY integer_constant
    ¦ { MAXVALUE integer_constant ¦ NO MAXVALUE }
    ¦ { MINVALUE integer_constant ¦ NO MINVALUE }
    ¦ { CYCLE ¦ NO CYCLE }
    ¦ { CACHE integer_constant ¦ NO CACHE }
<column constraint definition> ::=
    [ CONSTRAINT constraint_name ] <column constraint> [ <constraint characteristics> ]
<column constraint> ::=
      NOT NULL
    ¦ { UNIQUE ¦ PRIMARY KEY } [ <index name clause> [ <index attributes> ] [ TABLESPACE
index_tablespace_name ] ]
<index name clause> ::=
    INDEX index_name
<index attributes> ::=
      <index physical attribute clause>
```

```
    ¦ STORAGE ( <segment attr clause> [...] )
<table constraint definition> ::=
    [ CONSTRAINT constraint_name ] <table constraint> [ <constraint characteristics> ]
<table constraint> ::=
      <unique constraint definition> [ <index name clause> [ <index attributes> ] [ TABLESPACE
index_tablespace_name ] ]
<unique constraint definition> ::=
    { UNIQUE ¦ PRIMARY KEY } ( <key column element> [, ...] )
<key column element> ::=
    column_name [ ASC ¦ DESC ] [ NULLS FIRST ¦ NULLS LAST ]
<table sharding strategy> ::=
      <cloned strategy>
    ¦ <hash sharding strategy>
    ¦ <range sharding strategy>
    ¦ <list sharding strategy>
<cloned strategy> ::=
    CLONED [ <clone placement> ]
<clone placement> ::=
      AT CLUSTER WIDE
    ¦ AT CLUSTER GROUP group_list
<hash sharding strategy> ::=
    SHARDING BY [HASH] ( column_list )
    [ <hash shard count> ]
    [ <hash shard placement> ]
<hash shard count> ::=
    SHARD COUNT integer
<hash shard placement> ::=
      AT CLUSTER WIDE
    ¦ AT CLUSTER GROUP group_list
<range sharding strategy> ::=
    SHARDING BY RANGE ( column_list )
    { <cluster-wide range shard placement> ¦ <group-specific range shard placement> }
<cluster-wide range shard placement> ::=
    AT CLUSTER WIDE
    <range shard definition> [, ...]
<group-specific range shard placement> ::=
    <group-specific range shard definition> [, ...]
<group-specific range shard definition> ::=
    <range shard definition> AT CLUSTER GROUP group_name
<range shard definition> ::=
    SHARD range_name VALUES LESS THAN ( <range value clause> )
```

```
<range value clause> ::=
    <range value> [, ...]
<range value> ::=
      constant
    ¦ MAXVALUE
<list sharding strategy> ::=
    SHARDING BY LIST ( column_name )
    { <cluster-wide list shard placement> ¦ <group-specific list shard placement> }
<cluster-wide list shard placement> ::=
    AT CLUSTER WIDE
    <list shard definition> [, ...]
<group-specific list shard placement> ::=
    <group-specific list shard definition> [, ...]
<group-specific list shard definition> ::=
    <list shard definition> AT CLUSTER GROUP group_name
<list shard definition> ::=
      SHARD shard_name VALUES IN ( <list value clause> )
<list value clause> ::=
    <list value> [, ...]
<list value> ::=
      constant
    ¦ NULL
    ¦ DEFAULT
<table attribute clause> ::=
      [ <table physical attribute clause> ]
    ¦ [ STORAGE ( <segment attr clause> [...] ) ]
<table physical attribute clause> ::=
      PCTFREE integer
    ¦ PCTUSED integer
    ¦ INITRANS integer
    ¦ MAXTRANS integer
<index physical attribute clause> ::=
      PCTFREE integer
    ¦ INITRANS integer
    ¦ MAXTRANS integer
<segment attr clause> ::=
      INITIAL <size_clause>
    ¦ NEXT <size_clause>
    ¦ MINSIZE <size_clause>
    ¦ MAXSIZE <size_clause>
<size clause> ::=
```

```
      integer [ K ¦ M ¦ G ¦ T ]
<constraint characteristics> ::=
      [ NOT ] DEFERRABLE [ <constraint check time> ]
    ¦ <constraint check time> [ [ NOT ] DEFERRABLE ]
<constraint check time> ::=
      INITIALLY DEFERRED
    ¦ INITIALLY IMMEDIATE
<table global secondary index clause> ::=
      WITH GLOBAL SECONDARY INDEX [ <index attributes> [...] ] [ TABLESPACE tablespace_name ]
    ¦  WITHOUT GLOBAL SECONDARY INDEX
```

Invocation and Access Rules

The differences between the stand-alone database and the cluster database are as follows.

- Stand-alone
  - It can not define <table sharding strategy>.
  - It can not define <table global secondary index clause>.
- Cluster
  - Constraints of PRIMARY KEY, UNIQUE should include all sharding keys.
  - It can not define the deferrable constraints.

The user should satisfy the following conditions to perform <table definition> statement.

- One of the following privileges is required for the schema in which a table is to be created.
  - (CREATE TABLE or CONTROL SCHEMA) ON SCHEMA for that schema
  - CREATE ANY TABLE ON DATABASE

- One of the following privileges is required for the tablespace in which a table is to be created.
  - CREATE OBJECT ON TABLESPACE for that tablespace
  - USAGE TABLESPACE ON DATABASE

- One of the following privileges is required for the schema in which the constraints are to be created when the constraints exists which were created together.
  - (ADD CONSTRAINT or CONTROL SCHEMA) ON SCHEMA for that schema
  - ALTER ANY TABLE ON DATABASE

- One of the following privileges is required for the tablespace in which the index is to be created whe n the key constraint is created together.
  - CREATE OBJECT ON TABLESPACE for that tablespace
  - USAGE TABLESPACE ON DATABASE

- The owner of the table is determined as follows.
  - The owner of the schema to which the table belongs
  - If the schema to which the table belongs is PUBLIC, then it is the user who executed the statement.

- The table owner has the following privileges for the created table.
  - The privilege for the table
    - SELECT ON TABLE WITH GRANT OPTION
    - INSERT ON TABLE WITH GRANT OPTION
    - UPDATE ON TABLE WITH GRANT OPTION
    - DELETE ON TABLE WITH GRANT OPTION
    - TRIGGER ON TABLE WITH GRANT OPTION
    - REFERENCES ON TABLE WITH GRANT OPTION
    - LOCK ON TABLE WITH GRANT OPTION
    - INDEX ON TABLE WITH GRANT OPTION
    - ALTER ON TABLE WITH GRANT OPTION
  - The privilege for all columns in the table.
    - SELECT(columns) ON TABLE WITH GRANT OPTION
    - INSERT(columns) ON TABLE WITH GRANT OPTION
    - UPDATE(columns) ON TABLE WITH GRANT OPTION
    - REFERENCES(columns) ON TABLE WITH GRANT OPTION
  - The privilege for the constraint which was generated together
    - The owner of that constraint
    - The owner of the index which was generated together with the constraint

〈table sharding strategy〉 statement can be used in a cluster system.

Syntax Rules and Parameters

table_name

It is the table name to be created and it should be a unique name within the schema.
The schema to which the table belongs, such as schema_name.table_name, can be defined. If schema_name is omitted, The default schema name of the user performing the statement is used.
The length of the table name should be shorter than 128 bytes.

〈column definition〉

It defines the columns which configure the table.
The table should include one or more column definitions.
It can specify the column data type, default value, automatically generated value, and constraints.

column_name

It is name of the column which configure a table and each column should have a unique name within the table.
The length of the column name should be shorter than 128 bytes.

〈data type〉

- 〈character string type〉
- 〈binary string type〉
- 〈numeric type〉
- 〈exact numeric type〉
- 〈approximate numeric type〉
- 〈boolean type〉
- 〈datetime type〉
- 〈interval type〉
- 〈interval qualifier〉
- 〈non-second primary datetime field〉

It defines the data type of the column.
When defining the column including automatically generated values(〈identity column specification〉), its data type should be one of SMALLINT, INTEGER or BIGINT.
For more information about data types, refer to **Data Type**.

〈character length units〉

It specifies the length unit of a single character for the character type.

- CHARACTERS/ CHAR assigns the maximum bytes of a single character as the length of a single chara cter. Therefore, the length of multi-bytes single character such as Hangul is 1.
- OCTETS/ BYTE assigns one byte for the length of a single character. Therefore, the length of multi-by tes single character such as Hangul is multi-bytes.
- If it is omitted, it follows CHAR_LENGTH UNITS property which is used when creating the database.

SQL standard define CHARACTERS as a default value.

The default value of the char length unit in other DBMS are as follows.

- Oracle, DB2: OCTETS
- MS-SQL, MySQL, PostgreSQL: CHARACTERS

[<default clause> | <identity column specification>]

It specifies the default value of a column.
<default clause> and <identity column specification> can not be used together.
When both of them are omitted, the default value is NULL.

<default clause>

The DEFAULT clause defines the default value to be used when DEFAULT is specified in INSERT, UPDATE statements or the corresponding column name is omitted.

- When DEFAULT clause is used
  - e.g. CREATE TABLE t1 (id INTEGER, name VARCHAR(32) DEFAULT 'anonymous');
  - When the column is omitted
    - INSERT INTO t1(id) VALUES (1);
    - INSERT INTO t1(id) SELECT id FROM other_table;
  - When the DEFAULT is specified
    - INSERT INTO t1 DEFAULT VALUES;
    - INSERT INTO t1 VALUES (2, DEFAULT);
    - UPDATE t1 SET name = DEFAULT;

The data type of DEFAULT expression should be compatible with the data type of the column.
If the data type is not compatible or the expression is not valid, an error occurs.

```
--# result: error
CREATE TABLE t1 ( c1 INTEGER DEFAULT 1 / 0 );
ERR-22012(12122): divisor is equal to zero
--# result: success
CREATE TABLE t1 ( c1 INTEGER DEFAULT 1 / 1 );
Table created.
```

DEFAULT expression can use any built-in functions but it can not use the followings.

- Logical operators (AND, OR, NOT), comparison operators (=,>, ...)
- Stored function

- Column name
- Subquery expression

〈identity column specification〉

It defines a column which has automatically generated values.

The table can have only one identity column.
The identity column becomes *not nullable* column even though NOT NULL constraint is not specified.

〈identity column specification〉 clause can not be specified together with DEFAULT clause.
〈identity column specification〉 clause, like as DEFAULT clause, specifies DEFAULT in INSERT, UPDATE stat ements or it defines the default value to be used when the column name is omitted.

The generation method is defined as follows.

- GENERATED BY DEFAULT AS IDENTITY
  A user defined value is applied if defined, but the value is automatically generated, like as DEFAULT cl ause, when the default value should be used.
  - CREATE TABLE t1 (id INTEGER GENERATED BY DEFAULT AS IDENTITY, name VARCHAR(32));
  - (O) INSERT INTO t1 VALUES (12345, 'GOLDILOCKS');
    - It inserts the user defined value (12345).
  - (O) INSERT INTO t1(name) VALUES ('GOLDILOCKS');
    - It inserts the automatically generated value in an id column.
  - (O) INSERT INTO t1(id, name) SELECT other_id, other_name FROM other_table;
    - It inserts the user defined value.
  - (O) INSERT INTO t1(name) SELECT other_name FROM other_table;
    - It inserts the automatically generated value in an id column.
  - (O) UPDATE t1 SET id = 10000 WHERE id = 12345;
    - It inserts the user defined value
  - (O) UPDATE t1 SET id = DEFAULT WHERE id = 12345;
    - It inserts the automatically generated value in an id column.

- GENERATED ALWAYS AS IDENTITY
  A user can not define the value and the default value should be generated like as DEFAULT clause.
  - CREATE TABLE t1 (id INTEGER GENERATED ALWAYS AS IDENTITY, name VARCHAR(32));
  - (X) INSERT INTO t1 VALUES (12345, 'GOLDILOCKS');
    - Error, a user can not define the value.
  - (O) INSERT INTO t1(name) VALUES ('GOLDILOCKS');
    - It inserts the automatically generated value in an id column.
  - (X) INSERT INTO t1(id, name) SELECT other_id, other_name FROM other_table;
    - Error, a user can not define the value.

- (O) INSERT INTO t1(name) SELECT other_name FROM other_table;
  - It inserts the automatically generated value in an id column.
- (X) UPDATE t1 SET id = 10000 WHERE id = 12345;
  - Error, a user can not define the value.
- (O) UPDATE t1 SET id = DEFAULT WHERE id = 12345;
  - It inserts the automatically generated value in an id column.

For more information about <common sequence generator option> and <basic sequence generator option>, which is an option to create an identity column, refer to **CREATE SEQUENCE**.

<column constraint definition>

It defines the following constraints for a column.

- NOT NULL constraints
- UNIQUE constraints
- PRIMARY KEY constraints

constraint_name

It is the constraint name and it can be omitted.

If the constraint_name is omitted, it is automatically set as follows.
If the automatically generated name is duplicated, the constraint_name should be explicitly specified.

- NOT NULL constraints
  - "table_name" + "_" + "NOT_NULL" + "_" + "column_name"
- UNIQUE constraints
  - "table_name" + "_" + "UNIQUE" + "_" + "column_name"
- PRIMARY KEY constraints
  - "table_name" + "_" + "PRIMARY_KEY"

The length of the constraint name should be shorter than 128 bytes.

NOT NULL Constraint

NULL is not allowed for the column value.

UNIQUE Constraint

The identical value is not allowed for the column value, but NULL is allowed.

PRIMARY KEY Constraint

NULL or the identical value is not allowed as the column value. A single PRIMARY KEY constraint can be defined on a single table.

〈index name clause〉

It defines the index name to be created when defining UNIQUE constraint and PRIMARY KEY constraint.

- INDEX index_name
  - It defines the index name for the constraint.
  - It can not be used together with a schema name and it is created in the same schema where the constraint is created.

When defining UNIQUE constraint and PRIMARY KEY constraint, if INDEX clause is omitted, an index whi ch satisfies the constraints is automatically created.
"constraint_name" + "INDEX" is added to the name of index which is automatically generated.

- 〈index attributes〉
  - It specifies the physical attributes of the index to be created.
  - For more information, refer to **CREATE INDEX**.
- TABLESPACE index_tablespace_name
  - It specifies the tablespace where the index is to be created.
  - For more information, refer to **CREATE INDEX**.

〈table constraint definition〉

〈unique constraint definition〉

- When defining a table, the constraints can be classified in two ways depending on the location in the statement.
  - When defining a column, column constraints can be specified by using 〈column constraint defini tion〉.
  - On the other hand, the table constraint definition clause, 〈table constraint definition〉, can be sp ecified separately from the column definition and the constraints on one or more columns can be specified.

The table constraint definition has the following syntactic difference compared to the column constraint definition.

- NOT NULL constraint
  - It can not be specified by using the table constraint definition.
- It should explicitly specify the column unlike the column constraint definition.
  - UNIQUE constraint <unique constraint definition>
    - UNIQUE (column_name [, ...])
  - PRIMARY KEY CONSTRAINT <unique constraint definition>
    - PRIMARY KEY (column_name [, ...])

key column element

It specifies the column to be a target of the key.

- column name
  - It is name of the column which creates the key.
- ASC | DESC
  - ASC: It is sorted in an ascending order.
  - DESC: It is sorted in a descending order.
  - If not specified, the default value is ASC.
- NULLS FIRST | NULLS LAST
  - NULLS FIRST: It is located before non-NULL values.
  - NULLS LAST: It is located after non-NULL values.
  - If not specified, the default value is NULLS LAST.

<table sharding strategy>

It defines the sharding strategy of a table.
It can be defined as one of the four following strategies.

- <cloned strategy>
- <hash sharding strategy>
- <range sharding strategy>
- <list sharding strategy>

If it is omitted, it is determined by **DEFAULT_SHARDING** property value.

- If DEFAULT_SHARDING value is 0
  - <cloned strategy>
- If DEFAULT_SHARDING value is 1
  - <hash sharding strategy>

〈cloned strategy〉

It clones all data in a table.

〈clone placement〉

It defines the placement strategy of a clone.

- AT CLUSTER WIDE
  - It places clones in all cluster members of all cluster groups in a cluster system.
  - A clone can be relocated by using **ALTER TABLE name REBALANCE** statement when adding a cluster group and a cluster member.
- AT CLUSTER GROUP group_list
  - It places clones in all cluster members of a specified cluster groups.
  - A clone can be relocated by using **ALTER TABLE name REBALANCE** statement when adding a cluster member in a specified cluster group.
  - Adding a cluster group does not affect the relocation of the clone.
- When it is omitted, the default value is AT CLUSTER WIDE.

〈hash sharding strategy〉

It shards the table data according to the hash value of the sharding key.

SHARDING BY [HASH] (column_list)

It defines a sharding key for a hash sharding.

- It can list maximum 32 columns.
- It can not use a duplicate column.
- It can not use a LONG VARCHAR type column or a LONG VARBINARY type column.

〈hash shard count〉

It defines the number of the hash shards to be sharded.
The number of shards can be defined from 1 to 512.
If it is omitted, the default value is 24.

〈hash shard placement〉

It defines the placement strategy of a hash shard.

- AT CLUSTER WIDE
  - It places shards in all cluster members of all cluster groups in a cluster system.
  - A shard can be relocated by using **ALTER TABLE name REBALANCE** statement when adding a cluster group and a cluster member.
- AT CLUSTER GROUP group_list
  - It places hash shards in all cluster members of a specified cluster groups.
  - The number of group_list should be equal to or smaller than the value of <hash shard count>.
  - Unlike a range shard and a list shard, the cluster group on which the specific hash shard is to be located can not be specified, but the system automatically determines a cluster group on which the shard is to be located.
  - A shard can be relocated by using **ALTER TABLE name REBALANCE** statement when adding a cluster member in a specified cluster group.
  - Adding a cluster group does not affect the relocation of the hash shard.
- When it is omitted, the default value is AT CLUSTER WIDE.

<range sharding strategy>

It shards the table data according to the range value of the sharding key.

SHARDING BY RANGE (column_list)

It defines a sharding key for the range sharding.

- It can list maximum 32 columns.
- It can not use a duplicate column.
- It can not use a LONG VARCHAR type column or a LONG VARBINARY type column.

<cluster-wide range shard placement>

It automatically places range shards in all cluster groups of a cluster system.
AT CLUSTER WIDE statement is described before describing <range shard definition>.
Shards can be relocated by using **ALTER TABLE name REBALANCE** statement when adding a cluster group and a cluster member.

- Create a range sharded table.
- Place six shards in the existing cluster groups (g1, g2, g3).

```
CREATE TABLE t1
(
   id   INTEGER,
   name VARCHAR(32)
)
```

```
SHARDING BY RANGE (id)
    AT CLUSTER WIDE
    SHARD s1 VALUES LESS THAN ( 200000 ),
    SHARD s2 VALUES LESS THAN ( 400000 ),
    SHARD s3 VALUES LESS THAN ( 500000 ),
    SHARD s4 VALUES LESS THAN ( 600000 ),
    SHARD s5 VALUES LESS THAN ( 800000 ),
    SHARD s6 VALUES LESS THAN ( MAXVALUE )
;
```

- Add a cluster group.

```
CREATE CLUSTER GROUP g4
       CLUSTER MEMBER g4n1 HOST '192.168.0.41' PORT 10401
;
```

- Relocate the range shard.
- Place six shards in the cluster groups (g1, g2, g3, g4) including the added g4.

```
ALTER TABLE t1 REBALANCE;
```

〈group-specific range shard placement〉

It places range shards in a specified cluster group.
It describes AT CLUSTER GROUP group_name statement which places that shard together with 〈range shard definition〉.
Shards can be automatically relocated by using **ALTER TABLE name REBALANCE** statement when adding a cluster member to a specified cluster group.
Adding a cluster group does not affect the relocation of the range shard.

- Create a range sharded table.
- Place each range shard in a specified cluster group.

```
CREATE TABLE t1
(
   id   INTEGER,
   name VARCHAR(32)
)
SHARDING BY RANGE (id)
    SHARD s1 VALUES LESS THAN ( 200000 )   AT CLUSTER GROUP g1,
    SHARD s2 VALUES LESS THAN ( 400000 )   AT CLUSTER GROUP g2,
```

```
    SHARD s3 VALUES LESS THAN ( 500000 )   AT CLUSTER GROUP g3,
    SHARD s4 VALUES LESS THAN ( 600000 )   AT CLUSTER GROUP g2,
    SHARD s5 VALUES LESS THAN ( 800000 )   AT CLUSTER GROUP g3,
    SHARD s6 VALUES LESS THAN ( MAXVALUE ) AT CLUSTER GROUP g1
;
```

- Add a cluster group.

```
CREATE CLUSTER GROUP g4
       CLUSTER MEMBER g4n1 HOST '192.168.0.41' PORT 10401
;
```

- Relocate a range shard.
- The shard is not placed in a newly created cluster group g4.

```
ALTER TABLE t1 REBALANCE;
```

<range shard definition>

SHARD range_name should be unique in a table.

It can define maximum 512 of <range shard definition>.

The listed <range shard definition> is sorted in an order of <range value clause>, and it should use each di
fferent <range value clause>.

The <range shard definition> whose all values are define as MAXVALUE is a MAX shard.
MAX shard should exist, and it should be a single one.

- It should include a MAX shard.

```
gSQL>
CREATE TABLE t1
(
   id INTEGER,
   name VARCHAR(32)
)
SHARDING BY RANGE (id)
   AT CLUSTER WIDE
   SHARD s1 VALUES LESS THAN ( 100000 ),
   SHARD s2 VALUES LESS THAN ( 200000 ),
   SHARD s3 VALUES LESS THAN ( MAXVALUE )
```

```
;
Table created.
```

- If it does not include a MAX shard, then an error occurs.

```
gSQL>
CREATE TABLE t1
(
   id INTEGER,
   name VARCHAR(32)
)
SHARDING BY RANGE (id)
   AT CLUSTER WIDE
   SHARD s1 VALUES LESS THAN ( 100000 ),
   SHARD s2 VALUES LESS THAN ( 200000 ),
   SHARD s3 VALUES LESS THAN ( 300000 )
;
ERR-42000(16377): MAX shard not defined :
   SHARD s3 VALUES LESS THAN ( 300000 )
   *
ERROR at line 10:
```

〈range value clause〉

〈range value〉 should be a constant or a MAXVALUE (a maximum value).
NULL can not be used as 〈range value〉.

- (O) SHARD s1 VALUES LESS THAN (1)
- (O) SHARD s2 VALUES LESS THAN (1 + 1)
- (O) SHARD s3 VALUES LESS THAN (MAXVALUE)
- (X) SHARD s4 VALUES LESS THAN (SYSDATE)
- (X) SHARD s5 VALUES LESS THAN (NULL)

MAXVALUE is always bigger than any other value, and it includes null.

If there are multiple sharding keys, only a MAXVALUE can be specified after the MAXVALUE.

- (O) SHARD s1 VALUES LESS THAN (100, MAXVALUE)
- (X) SHARD s2 VALUES LESS THAN (MAXVALUE, 100)
- (O) SHARD s3 VALUES LESS THAN (MAXVALUE, MAXVALUE)

If a sharding key is defined by using multiple columns, a MAX shard which is listed with MAXVALUE for it

s all values as like the SHARD s3 below should exist.

```
CREATE TABLE t1
(
   id INTEGER,
   name VARCHAR(32)
)
SHARDING BY RANGE (id, name)
   AT CLUSTER WIDE
   SHARD s1 VALUES LESS THAN ( 100000, MAXVALUE ),
   SHARD s2 VALUES LESS THAN ( 200000, 20000 ),
   SHARD s3 VALUES LESS THAN ( MAXVALUE, MAXVALUE )
;
```

〈list sharding strategy〉

It shards the table data according to the listed value of the sharding key.

SHARDING BY LIST (column_name)

It defines a sharding key for a list sharding.

- It can use only one column.
- It can not use a LONG VARCHAR type column or a LONG VARBINARY type column.

〈cluster-wide list shard placement〉

It automatically places list shards in all cluster groups of a cluster system.
AT CLUSTER WIDE statement is described before describing 〈range shard definition〉.
Shards can be relocated by using **ALTER TABLE name REBALANCE** statement when adding a cluster grou p and a cluster member.

- Create a list sharded table.
- Place five shards in the existing cluster groups (g1, g2, g3).

```
CREATE TABLE city
(
   id   INTEGER,
   name VARCHAR(32)
)
SHARDING BY LIST (name)
```

```
    AT CLUSTER WIDE
    SHARD s1 VALUES IN ( 'SEOUL' ),
    SHARD s2 VALUES IN ( 'PUSAN', 'ULSAN', 'DAEGU' ),
    SHARD s3 VALUES IN ( 'DAEJEON', 'GWANGJU' ),
    SHARD s4 VALUES IN ( 'ANSAN', 'GOYANG' ),
    SHARD s5 VALUES IN ( DEFAULT )
;
```

- Add a cluster group.

```
CREATE CLUSTER GROUP g4
       CLUSTER MEMBER g4n1 HOST '192.168.0.41' PORT 10401
;
```

- Relocate a list shard.
- Place five shards in the cluster groups (g1, g2, g3, g4) including the added g4.

```
ALTER TABLE t1 REBALANCE;
```

<group-specific list shard placement>

It places list shards in a specified cluster group.
It describes AT CLUSTER GROUP group_name statement which places that shard together with <list shard definition>.
Shards can be automatically relocated by using **ALTER TABLE name REBALANCE** statement when adding a cluster member to a specified cluster group.
Adding a cluster group does not affect the relocation of the list shard.

- Create a list sharded table.
- Place each list shard in a specified cluster group.

```
CREATE TABLE city
(
   id   INTEGER,
   name VARCHAR(32)
)
SHARDING BY LIST (name)
    SHARD s1 VALUES IN ( 'SEOUL' )                  AT CLUSTER GROUP g1,
    SHARD s2 VALUES IN ( 'PUSAN', 'ULSAN', 'DAEGU' ) AT CLUSTER GROUP g2,
    SHARD s3 VALUES IN ( 'DAEJEON', 'GWANGJU' )     AT CLUSTER GROUP g3,
    SHARD s4 VALUES IN ( 'ANSAN', 'GOYANG' )        AT CLUSTER GROUP g2,
```

```
    SHARD s5 VALUES IN ( DEFAULT )                  AT CLUSTER GROUP g1
;
```

- Add a cluster group.

```
CREATE CLUSTER GROUP g4
       CLUSTER MEMBER g4n1 HOST '192.168.0.41' PORT 10401
;
```

- Relocate a list shard.
- The shard is not placed in a newly added cluster group g4.

```
ALTER TABLE t1 REBALANCE;
```

〈list shard definition〉

LIST list_name should be unique in a table.

It can define maximum 512 of 〈list shard definition〉.
All 〈list value〉 of the listed 〈list shard definition〉 should be different each other.

DFFAULT are other values which is not the listed 〈list value〉.
DFFAULT can not be defined together with another value.
A shard including DEFAULT is a DEFAULT shard.

MAX shard should exist, and it should be a single one.

- It should include a DEFAULT shard.

```
gSQL>
CREATE TABLE t1
(
   category INTEGER,
   name     VARCHAR(32)
)
SHARDING BY LIST (category)
   AT CLUSTER WIDE
   SHARD s1 VALUES IN ( 1, 3, 5, 7 ),
   SHARD s2 VALUES IN ( 2, 4, 6, 8 ),
   SHARD s3 VALUES IN ( DEFAULT )
;
Table created.
```

- If it does not include a DEFAULT shard, then an error occurs.

```
gSQL>
CREATE TABLE t1
(
   category INTEGER,
   name     VARCHAR(32)
)
SHARDING BY LIST (category)
   AT CLUSTER WIDE
   SHARD s1 VALUES IN ( 1, 3, 5, 7 ),
   SHARD s2 VALUES IN ( 2, 4, 6, 8 ),
   SHARD s3 VALUES IN ( 9, 10 )
;
ERR-42000(16385): DEFAULT shard not defined :
   SHARD s3 VALUES IN ( 9, 10 )
   *
ERROR at line 10:
```

〈list value clause〉

〈list value〉 should be a constant.
NULL or DEFAULT can be used as 〈list value〉.

- (O) SHARD s1 VALUES IN (1, 1 + 1, 3, 4)
- (O) SHARD s2 VALUES IN (5, 6, 7, NULL)
- (O) SHARD s3 VALUES IN (DEFAULT)
- (X) SHARD s4 VALUES IN (DEFAULT, 8, 9, 10)
- (X) SHARD s5 VALUES IN (current_timestamp, systimestamp)
- (X) SHARD s6 VALUES IN (c1, c2)

〈table physical attribute clause〉

It defines the physical attribute information of the table.

- PCTFREE integer
  - Definition
    - The reserved space in case the row size is increased when altering or updating a row in a pag e.
    - The initial input is input in other space except for this reserved space.
    - If PCTFREE is not enough ROW MIGRATION occurs when altering or updating the data.

    - It can use of the value from 0 to 99.
    - If it is omitted, the default value is 10.

- PCTUSED integer
    - Definition
        - It is the minimum percentage which can be used for the row data and overhead before addi ng a new row to a page.
        - It can be input only on this page when the value got smaller than PCTUSED due to alteration or deletion of the existing data.
    - It can use of the value from 0 to 99.
    - If it is omitted, the default value is 40.

- INITRANS integer
    - Definition
        - It specifies the initial number of transactions which can simultaneously access the page.
        - If the number of users who access the index is small, INITRANS is set to low, and if the numb er of users who simultaneously access the index is big, INITRANS is set to high.
        - If necessary, it is automatically increased to the specified MAXTRANS.
    - It can use the value from 1 to 32.
    - If it is omitted, the default value is 4.

- MAXTRANS integer
    - Definition
        - It specifies the maximum number of transactions which can simultaneously access the page.
    - It can use the value from 1 to 32.
    - If it is omitted, the default value is 8.

〈index physical attribute clause〉

It defines the physical attributes of the index.

- PCTFREE integer
    - Definition
        - It is a reserved space for adjusting the frequency of page splits caused by the key insertion in the page.
        - It is applied only to the index bottom-up build.
    - It can use the value from 0 to 99.
    - If it is omitted, the value set in DEFAULT_INDEX_PCTFREE property is used.
- INITRANS integer
    - It is as same as INITRANS in 〈table physical attribute clause〉.
- MAXTRANS integer
    - It is as same as MAXTRANS in 〈table physical attribute clause〉.

〈segment attr clause〉

It describes the information about the space in which the table is stored.

- INITIAL integer
  - Definition
    - It specifies the size of physical storage space which is initially allocated when creating the tabl e.
    - The size is aligned to the EXTENT size of TABLESPACE to which the table belongs. (e.g. If the EXT size is 8192 bytes, 'INITIAL 100' actually operates as 8192 bytes.)
    - The size (aligned to the size of TABLESPACE EXTENT) should be equal to or bigger than the s ize of MINEXTENTS, and it should be equal to or smaller than the size of the MAXEXTENTS.
  - The minimum value is 1, and the maximum value depends on the system environment.
  - If it is omitted, the default value is a single EXTENT size of TABLESPACE where the table belongs.

- NEXT integer
  - Definition
    - It specifies the physical space size to be allocated when adding the space to the table.
    - The size is aligned to EXTENT size of TABLESPACE to which the table belongs. (e.g. If the EX T size is 8192 bytes, 'INITIAL 100' actually operates as 8192 bytes.)
    - NEXT is operated as follows according to the size of the remaining space of the currently avai lable table. (The size of subtracting the currently used space from the size of MAXEXTENTS.)

    - If the remaining space size is 0, then it can not extend the space.
    - If the remaining space size is bigger than 0, but smaller than NEXT, then it allocates the space as big as the remaining space.
    - If the remaining space size is bigger than NEXT, then it allocates the space as big as the NEXT.
  - The minimum value is 1, and the maximum value depends on the system environment.
  - If it is omitted, the default value is a single EXTENT size of TABLESPACE to which the table belon gs.

- MINSIZE integer
  - Definition
    - It is the minimum space size which the table should maintain.
    - It should be equal to or smaller than the value of MAXSIZE.
  - It is aligned to the EXTENT size of TABLESPACE to which the table belongs.
  - The minimum value is 1, and the maximum value depends on the system environment.
  - If it is smaller than the size of two EXTENTs, the size of two EXTENTs is allocated.
  - If it is omitted, the default value is the size of two EXTENTs.

- MAXSIZE integer
  - Definition
    - It is the maximum space size which can be allocated to a table.
    - It should be equal to or bigger than the value of MINSIZE.

- This size is aligned to the EXTENT size of TABLESPACE to which the table belongs.
- The minimum value is 1, and the maximum value depends on the system environment.
- If it is omitted, the default value is 32 terabytes (35,184,372,088,832).
- Even though the value is set to over 32 terabytes, it is adjusted and set to 32 terabytes.

〈size clause〉

It specifies the file size in bytes. (If it is omitted, the default unit is bytes.)

- K: Kilobytes
- M: Megabytes
- G: Gigabytes
- T: Terabytes

TABLESPACE tablespace_name

It specifies the tablespace name in which a table is to be stored.
If TABLESPACE clause is omitted, the default tablespace_name of the user performing the statement is us ed.

TABLESPACE index_tablespace_name

It specifies the tablespace name in which an index is to be stored.
If TABLESPACE clause is omitted, then it uses the index tablespace of the user.
If the index tablespace of the user is NULL, then the DISK table uses the user's data tablespace and the M EMORY table uses the user's default temporary tablespace.

〈constraint characteristics〉

It defines characteristics of the constraint.
When defining constraints, the following characteristics can be set.

- DEFERRABLE | NOT DEFERRABLE
- 〈constraint check time〉

If 〈constraint characteristics〉 is omitted, it is set to NOT DEFERRABLE INITIALLY IMMEDIATE.

DEFERRABLE | NOT DEFERRABLE

It sets whether the constraints checking is deferrable so that the constraints can be checked when executi ng COMMIT without checking when executing DML statements.

The checking time of the deferrable constraint is controlled by **SET CONSTRAINTS**.

- NOT DEFERRABLE
  - The checking point is not deferrable, and the constraints are checked when executing INSERT/DE LETE/UPDATE statements.
- DEFERRABLE
  - The checking point can be controlled by **SET CONSTRAINTS** statement.
  - SET CONSTRAINTS constraint_name IMMEDIATE
    - The constraint is checked when executing DML statement.
  - SET CONSTRAINTS constraint_name DEFERRED
    - The constraint is checked when executing COMMIT statement.
- If not specified, the default value is determined in accordance with the 〈constraint check time〉.
  - If INITIALLY IMMEDIATE is specified, the constraint check time is NOT DEFERRABLE.
  - If INITIALLY DEFERRED is specified, the constraint check time is DEFERRABLE.
  - If 〈constraint check time〉 is not specified, the constraint check time is NOT DEFERRABLE.

〈constraint check time〉

If the constraints are DEFERRABLE, it sets an initial value for the checking time.

- INITIALLY IMMEDIATE
  - The constraint is checked when executing the DML statements.
- INITIALLY DEFERRED
  - The constraint is checked when executing the COMMIT statements.
  - It can not be used together with NOT DEFERRABLE.
- If not specified, the default value is INITIALLY IMMEDIATE.

For more information about the deferrable constraints, refer to **SET CONSTRAINTS**.

〈table global secondary index clause〉

It defines a global secondary index of the table.

- WITH GLOBAL SECONDARY INDEX [〈index attributes〉 [...]] [TABLESPACE tablespace_name]
  - It creates a global secondary index when creating a table in a cluster system environment.
  - 〈index attribute〉
    - It sets an index attribute of a global secondary index.
  - TABLESPACE tablespace_name

        - It specifies a tablespace to create a global secondary index.
- WITHOUT GLOBAL SECONDARY INDEX
    - It does not create a global secondary index when creating a table in a cluster system environment.

Description

Constraint Characteristics

GOLDILOCKS automatically creates an index to check the uniqueness when generating key constraints.

The following columns do not allow NULL value.

- A column including NOT NULL constraint
- A column which is included in primary key constraints
- An identity column

Cluster Table

A table manages data using one of the following sharding strategies in a cluster environment.

- Cloned table
    - It duplicates all table data and place them in a cluster system.
- Hash sharded table
    - It divides the table data into several shards based on the hash value of a sharding key, then place s them in the cluster system.
- Range sharded table
    - It divides the table data into several shards based on the range value of a sharding key, then plac es them in the cluster system.
- List sharded table
    - It divides the table data into several shards based on the list value of a sharding key, then places t hem in the cluster system.

The sharding strategy is determined considering the followings when creating a table. Tables operated in a cluster system are specified as a code table and a fact table based on its features.

- Code table
    - It is a table such as a product list or a provider list whose data is relatively small and is not often al tered
    - This table is referenced often together with a fact table.
- Fact table
    - It is a table such as a transaction history or a call history whose data is relatively big and is often al tered.

- Its data is big so required to be sharded.

<cloned strategy> is appropriate for a code table, and an appropriate <table sharding strategy> should be determined according to the table access pattern in case of a fact table.

Examples

The following is an example of generating an ordinary table.

```
gSQL> CREATE TABLE region
(
    r_regionkey    INTEGER
  , r_name         CHAR(25)
  , r_comment      VARCHAR(152)
);
Table created.
```

The following is an example of specifying the constraints on the columns when creating a table.

```
gSQL> CREATE TABLE supplier
(
    s_suppkey      INTEGER PRIMARY KEY
  , s_name         CHAR(25) NOT NULL
  , s_address      VARCHAR(40)
  , s_nationkey    INTEGER
  , s_phone        CHAR(15)
  , s_acctbal      NUMERIC(12,2)
  , s_comment      VARCHAR(101)
);
Table created.
```

The following is an example of specifying the constraint which includes multiple columns when creating a table.

```
gSQL> CREATE TABLE partsupp
(
    ps_partkey     INTEGER
  , ps_suppkey     INTEGER
  , ps_availqty    INTEGER
  , ps_supplycost NUMERIC(12,2)
```

```
  , ps_comment    VARCHAR(199)
  , CONSTRAINT ps_unique_key UNIQUE(ps_partkey, ps_suppkey)
);
Table created.
```

The following is an example of specifying whether constraints are deferrable when creating a table.

```
gSQL> CREATE TABLE t1
(
    id     NUMBER        PRIMARY KEY
                         NOT DEFERRABLE INITIALLY IMMEDIATE
  , name   VARCHAR(128)  CONSTRAINT t1_nn NOT NULL
                         DEFERRABLE INITIALLY IMMEDIATE
  , addr   VARCHAR(1024)
  , CONSTRAINT t1_uk UNIQUE ( id, name )
                     DEFERRABLE INITIALLY DEFERRED
);
Table created.
gSQL> COMMIT;
Commit complete.
```

The following is an example of specifying a column including automatically generated values and the def ault value when creating a table.

```
CREATE TABLE customer
(
    c_custkey     INTEGER   GENERATED BY DEFAULT AS IDENTITY
  , c_name        VARCHAR(25)
  , c_address     VARCHAR(40) DEFAULT 'N/A'
  , c_nationkey   INTEGER
  , c_phone       CHAR(15)
  , c_acctbal     NUMERIC(12,2)
  , c_mktsegment  CHAR(10)
  , c_comment     VARCHAR(117)
);
Table created.
```

The following is an example of specifying the tablespace which is to be stored when creating a table.

```
gSQL> CREATE TABLE lineitem
(
    l_orderkey      INTEGER
```

```
  , l_partkey       INTEGER
  , l_suppkey       INTEGER
  , l_linenumber    INTEGER
  , l_quantity      NUMERIC(12,2)
  , l_extendedprice NUMERIC(12,2)
  , l_discount      NUMERIC(12,2)
  , l_tax           NUMERIC(12,2)
  , l_returnflag    CHAR(1)
  , l_linestatus    CHAR(1)
  , l_shipdate      DATE
  , l_commitdate    DATE
  , l_receiptdate   DATE
  , l_shipinstruct  CHAR(25)
  , l_shipmode      CHAR(10)
  , l_comment       VARCHAR(44)
  , PRIMARY KEY (l_orderkey, l_linenumber) INDEX lineitem_pk_idx TABLESPACE mem_temp_tbs
) TABLESPACE mem_data_tbs;
Table created.
```

The following is an example of defining a cluster-wide cloned table. The table data is copied and placed a ll over the cluster system.

```
gSQL>
CREATE TABLE region
(
    r_regionkey   INTEGER
  , r_name        CHAR(25)
  , r_comment     VARCHAR(152)
)
CLONED
AT CLUSTER WIDE
;
Table created.
```

The following is an example of defining a group-specific cloned table. The table data is copied and placed in a cluster group g1 and g2 specified by a user.

```
gSQL>
CREATE TABLE region
(
    r_regionkey   INTEGER
  , r_name        CHAR(25)
```

```
  , r_comment     VARCHAR(152)
)
CLONED
AT CLUSTER GROUP g1, g2
;
Table created.
```

The following is an example of defining a cluster-wide hash sharded table. The table data is divided into 2 4 shards based on the hash value of a ps_partkey column, and each shard is automatically placed all over the cluster system.

```
gSQL>
CREATE TABLE partsupp
(
    ps_partkey    INTEGER
  , ps_suppkey    INTEGER
  , ps_availqty   INTEGER
  , ps_supplycost NUMERIC(12,2)
  , ps_comment    VARCHAR(199)
)
SHARDING BY HASH ( ps_partkey )
SHARD COUNT 24
AT CLUSTER WIDE
;
Table created.
```

The following is an example of defining a group-specific hash sharded table. The table data is divided int o 24 shards based on the hash value of a ps_partkey column, and each shard is automatically placed in a specified cluster group g2 and g3.

```
gSQL>
CREATE TABLE partsupp
(
    ps_partkey    INTEGER
  , ps_suppkey    INTEGER
  , ps_availqty   INTEGER
  , ps_supplycost NUMERIC(12,2)
  , ps_comment    VARCHAR(199)
)
SHARDING BY HASH ( ps_partkey )
SHARD COUNT 24
AT CLUSTER GROUP g2, g3
```

```
;
Table created.
```

The following is an example of defining a cluster-wide range sharded table. The table data is divided into 24 shards based on the range value of D_ID column, and each shard is automatically placed all over the cl uster system.

```
gSQL>
CREATE TABLE DISTRICT (
    D_ID        INTEGER,
    D_W_ID      INTEGER,
    D_NAME      VARCHAR(10),
    D_STREET_1  VARCHAR(20),
    D_STREET_2  VARCHAR(20),
    D_CITY      VARCHAR(20),
    D_STATE     CHAR(2),
    D_ZIP       CHAR(9),
    D_TAX       NUMERIC(4,4),
    D_YTD       NUMERIC(15,2),
    D_NEXT_O_ID INTEGER,
    PRIMARY KEY (D_W_ID, D_ID) INDEX DISTRICT_PK_IDX
)
    SHARDING BY RANGE (D_ID)
    AT CLUSTER WIDE
    SHARD s1 VALUES LESS THAN ( 100 ),
    SHARD s2 VALUES LESS THAN ( 200 ),
    SHARD s3 VALUES LESS THAN ( 300 ),
    SHARD s4 VALUES LESS THAN ( 400 ),
    SHARD s5 VALUES LESS THAN ( 500 ),
    SHARD s6 VALUES LESS THAN ( 600 ),
    SHARD s7 VALUES LESS THAN ( 700 ),
    SHARD s8 VALUES LESS THAN ( MAXVALUE );
;
Table created.
```

The following is an example of defining a group-specific range sharded table. The table data is divided int o three range values based on the range value of a NO_D_ID column, and a shard s1 is placed in a cluster group g1, a shard s2 is placed in a cluster group g2, a shard s3 is placed in a cluster group g3.

```
gSQL>
CREATE TABLE NEW_ORDER
(
```

```
    NO_O_ID INTEGER,
    NO_D_ID INTEGER,
    NO_W_ID INTEGER,
    PRIMARY KEY(NO_W_ID, NO_D_ID, NO_O_ID) INDEX NEW_ORDER_PK_IDX
)
    SHARDING BY RANGE (NO_D_ID)
    SHARD s1 VALUES LESS THAN ( 5 )        AT CLUSTER GROUP g1,
    SHARD s2 VALUES LESS THAN ( 8 )        AT CLUSTER GROUP g2,
    SHARD s3 VALUES LESS THAN ( MAXVALUE ) AT CLUSTER GROUP g3
;
Table created.
```

The following is an example of defining a cluster-wide list sharded table. A list shard is divided into five b ased on a city column, and each shard is automatically placed all over the cluster system.

```
gSQL>
CREATE TABLE t1
(
    id   INTEGER
  , name VARCHAR(32)
  , city VARCHAR(128)
)
   SHARDING BY LIST (city)
      AT CLUSTER WIDE
      SHARD s1 VALUES IN ( 'seoul' ),
      SHARD s2 VALUES IN ( 'busan', 'ulsan' ),
      SHARD s3 VALUES IN ( 'suwon', 'ansan', 'osan' ),
      SHARD s4 VALUES IN ( 'goyang', 'paju', 'guri' ),
      SHARD s5 VALUES IN ( DEFAULT )
;
Table created.
```

The following is an example of defining a group-specific list sharded table. A list shard is divided into five based on a city column, and each shard is placed in a specified cluster group.

```
gSQL>
CREATE TABLE t1
(
    id   INTEGER
  , name VARCHAR(32)
  , city VARCHAR(128)
)
```

```
   SHARDING BY LIST (city)
      SHARD s1 VALUES IN ( 'seoul' )                 AT CLUSTER GROUP g1,
      SHARD s2 VALUES IN ( 'busan', 'ulsan' )        AT CLUSTER GROUP g2,
      SHARD s3 VALUES IN ( 'suwon', 'ansan', 'osan' ) AT CLUSTER GROUP g1,
      SHARD s4 VALUES IN ( 'goyang', 'paju', 'guri' ) AT CLUSTER GROUP g2,
      SHARD s5 VALUES IN ( DEFAULT )                 AT CLUSTER GROUP g3
;
Table created.
```

A table T1 is created without a global secondary index.

```
gSQL> CREATE TABLE T1 ( I1 INTEGER, I1 CHAR(32) )  WITHOUT GLOBAL SECONDARY INDEX;
Table created.
```

A global secondary index is created after creating a table T1.

```
gSQL> CREATE TABLE T1 ( I1 INTEGER, I1 CHAR(32) )  WITH GLOBAL SECONDARY INDEX;
Table created.
```

A global secondary index of table T1 is created in a tablespace USER_DATA_TBS as a logging index after creating a table T1.

```
gSQL> CREATE TABLE T1 ( I1 INTEGER, I1 CHAR(32) )
      WITH GLOBAL SECONDARY INDEX
      TABLESPACE USER_DATA_TBS;
Table created.
```

A global secondary index of table T1 is created in a tablespace USER_TEMP_TBS as a nologging index aft er creating a table T1.

```
gSQL> CREATE TABLE T1 ( I1 INTEGER, I1 CHAR(32) )
      WITH GLOBAL SECONDARY INDEX
      TABLESPACE USER_TEMP_TBS;
Table created.
```

Compatibility

The SQL standard does not define the following clauses.

- The physical concepts of TABLESPACE clause and <physical attribute clause> clause.
- The SQL standard does not allow an operation in DEFAULT clause.

Table 19-11 SQL standard compatibility

Feature ID	Description	Compatibility
T171	LIKE clause in table definition	X
F531	Temporary tables	X
S051	Create table of type	X
S043	Enhanced reference types	X
S081	Subtables	X
T173	Extended LIKE clause in table definition	X
T180	System-versioned tables	X
F692	Extended collation support	X
T174	Identity columns	O
T175	Generated columns	X
S071	SQL paths in function and type name resolution	X
F321	User authorization	O
T322	Extended roles	X
F762	CURRENT_CATALOG	O
F763	CURRENT_SCHEMA	O

For More Information

Refer to the followings.

- DROP TABLE
- ALTER TABLE
- CREATE TABLESPACE
- CREATE SCHEMA
- CREATE INDEX
- CREATE SEQUENCE
- SET CONSTRAINTS
- ALTER TABLE name ADD GLOBAL SECONDARY INDEX

19.18 CREATE TABLE AS SELECT

Function

It creates a new table from the query result.

Syntax

```
<table definition: AS query expression> ::=
    CREATE TABLE table_name
        [ ( column_name [, ...] ) ]
        [ <table sharding strategy> ]
        [ <table attribute clause> [, ...] ]
        [ TABLESPACE tablespace_name ]
        [ <table global secondary index clause> ]
        AS <query expression> [ WITH [ NO ] DATA ]
    ;
<table sharding strategy> ::=
      <cloned strategy>
    ¦ <hash sharding strategy>
    ¦ <range sharding strategy>
    ¦ <list sharding strategy>
<cloned strategy> ::=
    CLONED [ <clone placement> ]
<clone placement> ::=
      AT CLUSTER WIDE
    ¦ AT CLUSTER GROUP group_list
<hash sharding strategy> ::=
    SHARDING BY [HASH] ( column_list )
    [ <hash shard count> ]
    [ <hash shard placement> ]
<hash shard count> ::=
    SHARD COUNT integer
<hash shard placement> ::=
      AT CLUSTER WIDE
    ¦ AT CLUSTER GROUP group_list
<range sharding strategy> ::=
```

```
    SHARDING BY RANGE ( column_list )
    { <cluster-wide range shard placement> ¦ <group-specific range shard placement> }
<cluster-wide range shard placement> ::=
    AT CLUSTER WIDE
    <range shard definition> [, ...]
<group-specific range shard placement> ::=
    <group-specific range shard definition> [, ...]
<group-specific range shard definition> ::=
    <range shard definition> AT CLUSTER GROUP group_name
<range shard definition> ::=
    SHARD range_name VALUES LESS THAN ( <range value clause> )
<range value clause> ::=
    <range value> [, ...]
<range value> ::=
      constant
    ¦ MAXVALUE
<list sharding strategy> ::=
    SHARDING BY LIST ( column_name )
    { <cluster-wide list shard placement> ¦ <group-specific list shard placement> }
<cluster-wide list shard placement> ::=
    AT CLUSTER WIDE
    <list shard definition> [, ...]
<group-specific list shard placement> ::=
    <group-specific list shard definition> [, ...]
<group-specific list shard definition> ::=
    <list shard definition> AT CLUSTER GROUP group_name
<list shard definition> ::=
      SHARD shard_name VALUES IN ( <list value clause> )
<list value clause> ::=
    <list value> [, ...]
<list value> ::=
      constant
    ¦ NULL
    ¦ DEFAULT
<table attribute clause> ::=
      [ <table physical attribute clause> ]
    ¦ [ STORAGE ( <segment attr clause> [...] ) ]
<table physical attribute clause> ::=
      PCTFREE integer
    ¦ PCTUSED integer
    ¦ INITRANS integer
```

```
    ¦ MAXTRANS integer
<index physical attribute clause> ::=
      PCTFREE integer
    ¦ INITRANS integer
    ¦ MAXTRANS integer
<segment attr clause> ::=
      INITIAL <size_clause>
    ¦ NEXT <size_clause>
    ¦ MINSIZE <size_clause>
    ¦ MAXSIZE <size_clause>
<size clause> ::=
      integer [ K ¦ M ¦ G ¦ T ]
<table global secondary index clause> ::=
      WITH GLOBAL SECONDARY INDEX [ <index attributes> [...] ] [ TABLESPACE tablespace_name ]
    ¦  WITHOUT GLOBAL SECONDARY INDEX
```

Invocation and Access Rules

A user should satisfy the following conditions to perform <table definition:AS query expression>statement.

- Table creation privilege
  - Refer to the access privilege in **CREATE TABLE**.
- SELECT access privilege
  - Refer to the access privilege in **SELECT**.

Syntax Rules and Parameters

table_name

It is the table name to be created.
For more information, refer to **table_name**.

column_name_list

It is the name of columns which configure the table, and it should be unique names within the table.
The number of columns should be as same as the number of result columns in SELECT clause.
If not specified, the column names of SELECT clause in <query expression> are used.

However, if there is a expression (function, operation, subquery, etc.) instead of a column in SELECT clau se, the alias or column name should be specified.

The length of the column name should be shorter than 128 bytes.

WITH [NO] DATA

If WITH DATA is specified, the result of SELECT clause is inserted to the table to be created.
If WITH NO DATA is specified, the result of SELECT clause is not inserted to the table to be created.
If not specified, it is operated as same as when WITH DATA is specified.

Other Syntax

For more information about other syntaxes, refer to the syntax in **CREATE TABLE** statement.

Description

When executing CREATE TABLE AS SELECT, if a column including a NOT NULL constraint is specified in S ELECT list, the NOT NULL constraint is also created in the new table.
However, if the NOT NULL constraint is deferrable, then NOT NULL constraint is not created in the new ta ble.

On the other hand, the NOT NULL constraint is not created in the new table if NOT NULL constraint was n ot explicitly created but there is NOT NULL property such as primary key column or identity column.

Examples

The following is an example of executing CREATE TABLE AS SELECT statement.

```
gSQL> CREATE TABLE recent_orders
                AS SELECT order_id, order_item, order_date
                   FROM orders
                   WHERE order_date >= '2015-03-03'
Table created.
```

The following is an example of specifying the column name.

```
gSQL> CREATE TABLE recent_orders ( order_id, order_item, order_date )
                AS SELECT order_id, order_item, order_date
                   FROM orders
                   WHERE order_date >= '2015-03-03'
Table created.
```

The following is an example of a function in SELECT list.

```
gSQL> CREATE TABLE recent_orders ( order_id, order_item, order_date )
                AS SELECT order_date, COUNT(*)
                   FROM orders
                   WHERE order_date >= '2015-03-03'
                   GROUP BY order_date;
Table created.
```

The following is an example of the statement including WITH DATA.

```
gSQL>CREATE TABLE orders
(
    order_id   NUMBER
  , order_item VARCHAR(128)
  , order_date DATE
);
gSQL> COMMIT;
gSQL> INSERT INTO orders VALUES ( 1, 'Pen', '2010-01-01' );
gSQL> INSERT INTO orders VALUES ( 2, 'Book', '2015-03-03' );
gSQL> COMMIT;
gSQL> CREATE TABLE recent_orders
                AS SELECT order_id, order_item, order_date
                   FROM orders
                   WHERE order_date >= '2015-03-03'
      WITH DATA;
Table created.
gSQL> SELECT COUNT(*) FROM recent_orders;
COUNT(*)
--------
       1
1 row selected.
```

The following is an example of the statement including WITH NO DATA.

```
gSQL>CREATE TABLE orders
(
    order_id   NUMBER
  , order_item VARCHAR(128)
  , order_date DATE
);
gSQL> COMMIT;
gSQL> INSERT INTO orders VALUES ( 1, 'Pen', '2010-01-01' );
gSQL> INSERT INTO orders VALUES ( 2, 'Book', '2015-03-03' );
gSQL> COMMIT;
gSQL> CREATE TABLE recent_orders
                AS SELECT order_id, order_item, order_date
                   FROM orders
                   WHERE order_date >= '2015-03-03'
      WITH NO DATA;
Table created.
gSQL> SELECT COUNT(*) FROM recent_orders;
COUNT(*)
--------
       0
1 row selected.
```

Compatibility

CREATE TABLE AS SELECT statement follows SQL standard. However, the following is an extension of SQ L standard.

- SQL standard require parentheses outside SELECT clause, but it is optional in GOLDILOCKS.
- SQL standard require WITH [NO] DATA clause, but it is optional in GOLDILOCKS.
- The concepts of tablespace in GOLDILOCKS is an extended concept, and it is not supported in SQL st andard.

Table 19-12 SQL standard compatibility

Feature ID	Description	Compatibility
T172	AS subquery clause in table definition	O

For More Information

Refer to the followings.

- CREATE TABLE
- SELECT

19.19 CREATE TABLESPACE

Function

It creates a tablespace.

Syntax

```
<create tablespace statement> ::=
      <memory data tablespace statement>
    | <memory temporary tablespace statement>
    ;
```

Invocation and Access Rules

CREATE TABLESPACE ON DATABASE privilege is required to perform <create tablespace statement>.

The user who performed the statement has CREATE OBJECT ON TABLESPACE privilege on the created tablespace.

One of the following privileges are required to create the objects on the created tablespace.

- CREATE OBJECT ON TABLESPACE for that tablespace
- USAGE TABLESPACE ON DATABASE

Syntax Rules and Parameters

<memory data tablespace statement>

- [MEMORY] TEMPORARY

It is a memory temporary tablespace to store the no logging indexes or the temporary objects such as intermediate results which are generated during the query processing.
The reserved word MEMORY can be omitted.

〈memory data tablespace clause〉

It defines a memory data tablespace.
For more information, refer to CREATE MEMORY DATA TABLESPACE.

〈memory temporary tablespace definition〉

It defines a memory temporary tablespace.
For more information, refer to CREATE MEMORY TEMPORARY TABLESPACE.

Description

For more information, refer to the description of each detailed statement.

Example

For more information, refer to usage example of each detailed statement.

Compatibility

The SQL standard does not cover the concepts of the tablespace.

For More Information

Refer to the followings.

- DROP TABLESPACE
- ALTER TABLESPACE

19.20 CREATE USER

Function

It defines a database user.

Syntax

```
<user definition> ::=
    CREATE USER user_identifier IDENTIFIED BY password
    [ PROFILE { profile_name ¦ DEFAULT ¦ NULL } ]
    [ PASSWORD EXPIRE ]
    [ ACCOUNT { LOCK ¦ UNLOCK } ]
    [ DEFAULT TABLESPACE tablespace_name ]
    [ TEMPORARY TABLESPACE tablespace_name ]
    [ INDEX TABLESPACE { tablespace_name ¦ NULL } ]
    [ <schema clause> ]
    ;
<schema clause> ::=
      WITH SCHEMA [schema_name]
    ¦ WITHOUT SCHEMA
```

Invocation and Access Rules

CREATE USER ON DATABASE privilege is required to perform <user definition>.

The created user, user_identifier, has the privilege, which is the owner the schema created by using <schema clause>.

A separate privilege is not granted to the created user_identifier.
The appropriate privileges should be granted to user_identifier to access and perform SQL statements.

Syntax Rules and Parameters

user_identifier

It is the username to be created.
The identical username (user identifier) or role (role name) should not exist.
The length of user_identifier should be shorter than 128 bytes.

password

It is the user's password to be created. It is encrypted and stored.
The length of password should be shorter than 128 bytes.
The password is case sensitive.
The password should start with an alphabetic character, and it can include alphabetic characters, number s, underscore (_), and $.
The other special characters should be enclosed in double quotes (").

PROFILE { profile_name | DEFAULT | NULL }

The profile for password management policy is assigned.

- PROFILE profile_name
  - It allocates the profile_name which is created by a user.
- PROFILE DEFAULT
  - It allocates the default profile "DEFAULT".
- PROFILE NULL
  - It does not allocate the profile.

If PROFILE clause is omitted, it is as same as PROFILE NULL, and the profile is not applied.
For more information about the password management policy, refer to **CREATE PROFILE**.

PASSWORD EXPIRE

It expires the user's password.
It is used when a user attempts to change the password by force before login.

ACCOUNT { LOCK | UNLOCK }

- ACCOUNT LOCK
  - It locks the user account.
- ACCOUNT UNLOCK
  - It unlocks the user account.

DEFAULT TABLESPACE tablespace_name

It specifies the default TABLESPACE to store objects such as the table created by the user, the indexes (wi th NOLOGGING option).
If DEFAULT TABLESPACE clause is omitted, default data tablespace (MEM_DATA_TBS) is specified, which was defined when creating DATABASE.

TEMPORARY TABLESPACE tablespace_name

It specifies the TABLESPACE which stores the temporary tables created by a user, indexes (NO LOGGING), and the intermediate results generated by the query processing.
If TEMPORARY TABLESPACE clause is omitted, default temporary tablespace (MEM_TEMP_TBS) is specifi ed, which was defined when creating DATABASE.

INDEX TABLESPACE { tablespace_name | NULL }

It specifies the default TABLESPACE which stores the index objects created by a user.

- Specifying INDEX TABLESPACE tablespace_name
  - If it specifies the data tablespace, then it becomes the LOGGING index.
  - If it specifies the temporary tablespace, then it becomes the NOLOGGING index.

- INDEX TABLESPACE NULL
  - It does not specify the index tablespace.

If INDEX TABLESPACE clause is omitted, then it is INDEX TABLESPACE NULL.

〈schema clause〉

It creates a schema which a user uses by default.
The schema name should be unique in the database.

- WITH SCHEMA [schema_name]
  - If schema_name is not assigned, the schema is created whose name is as same as user_identifier.

    - SCHEMA PATH of the user is set as follows.
        - schema_name, PUBLIC
- WITHOUT SCHEMA
    - It does not create the schema which is to be owned by a user.
    - SCHEMA PATH of the user is set as PUBLIC.

If <schema clause> is not specified, the default value is WITH SCHEMA and the schema is created whose name is as same as user_identifier.
The schema to be owned by the user can be additionally created by using **CREATE SCHEMA** statement.

Description

A user is an authorization object which consists of a set of privileges.

When <user definition> statement is executed for the first time, a user without any privilege is created, an d the appropriate privileges should be granted as follows.

In GOLDILOCKS, the relationship between user and schema is 1 : N.
In other words, a user does not own any schema, or a user can own multiple schemas.

The SQL standard does not explicitly define the relationship of the non-schema objects such as a user, a s chema, or a database. Each DBMS defines the relationship of non-schema objects in different concept as follows.

The relationship between user and schema in other DBMS.

- Oracle
    - User : schema = 1 : 1.

- DB2
    - It is as same as the OS user.
    - The separate SQL statements which creates or deletes a user do not exist.

- Postgres
    - User : schema = 1 : N.

- MySQL
  - Database : schema = 1 : 1.
  - User is a subordinate object of database (schema).

Examples

At least the following privileges should be granted to create a user, and for the created user to create the objects, manipulate data.

The following is an example of creating a user and granting privileges.

- Create a user.

```
gSQL> CREATE USER u1 IDENTIFIED BY u1_password
             DEFAULT   TABLESPACE mem_data_tbs
             TEMPORARY TABLESPACE mem_temp_tbs
             INDEX TABLESPACE NULL;
User created.
gSQL> COMMIT;
Commit complete.
```

- Grant database privileges.

```
gSQL> GRANT CREATE SESSION ON DATABASE TO u1;
Grant succeeded.
COMMIT;
Commit complete.
```

- Grant schema privileges.

```
GRANT CREATE TABLE, CREATE VIEW, CREATE INDEX, CREATE SEQUENCE, ADD CONSTRAINT
      ON SCHEMA u1 TO u1;
Grant succeeded.
COMMIT;
Commit complete.
```

- Grant tablespace privileges.

```
GRANT CREATE OBJECT ON TABLESPACE mem_data_tbs TO u1;
Grant succeeded.
```

```
GRANT CREATE OBJECT ON TABLESPACE mem_temp_tbs TO u1;
Grant succeeded.
COMMIT;
Commit complete.
```

The following is an example of creating objects by the user.

- It needs CREATE SESSION ON DATABASE.

```
gSQL> \connect u1 u1_password
```

- It needs CREATE TABLE ON SCHEMA u1.
- It needs CREATE OBJECT ON TABLESPACE mem_data_tbs.

```
gSQL> CREATE TABLE u1.t1 ( c1 INTEGER, c2 INTEGER ) TABLESPACE mem_data_tbs;
Table created.
gSQL> COMMIT;
```

- It needs CREATE INDEX ON SCHEMA u1.
- It needs CREATE OBJECT ON TABLESPACE mem_temp_tbs.

```
gSQL> CREATE INDEX u1.idx ON t1 (c2) TABLESPACE mem_temp_tbs;
Index created.
gSQL> COMMIT;
```

- It needs ADD CONSTRAINT ON SCHEMA u1.

```
gSQL> ALTER TABLE t1 ADD CONSTRAINT u1.t1_pk PRIMARY KEY (c1) ;
Table altered.
gSQL> COMMIT;
```

- It needs CREATE SEQUENCE ON SCHEMA u1.

```
gSQL> CREATE SEQUENCE u1.seq;
Sequence created.
gSQL> COMMIT;
gSQL> INSERT INTO u1.t1 VALUES ( u1.seq.NEXTVAL, u1.seq.NEXTVAL );
1 row created
gSQL> COMMIT;
```

Compatibility

The SQL standard covers the concepts of the user, but it does not define the SQL statements associated with the creation and deletion of user.

For More Information

Refer to the followings.

- DROP USER
- ALTER USER
- CREATE SCHEMA

19.21 CREATE VIEW

Function

It defines a view.

Syntax

```
<view definition> ::=
    CREATE [ OR REPLACE ] [ FORCE ¦ NO FORCE ]
        VIEW view_name [ ( column_name [, ...] ) ]
        AS <query expression>
    ;
```

Invocation and Access Rules

The user should satisfy the following conditions to perform <view definition> statement.

- One of the following privileges is required to create a view.
  - (CREATE VIEW or CONTROL SCHEMA) ON SCHEMA for the schema to which the view belongs
  - CREATE ANY VIEW ON DATABASE

- When using OR REPLACE clause, if a view already exists, then one of the following privileges is required to remove the existing view.
  - The owner of that view
  - CONTROL TABLE ON TABLE for that view.
  - (DROP VIEW or CONTROL SCHEMA) ON SCHEMA for the schema to which the view belongs
  - DROP ANY VIEW ON DATABASE

- One of the following privileges is required for all tables used in <Query expression>.
  - SELECT(columns) ON TABLE for all columns used in the statement among table columns
  - (SELECT or CONTROL TABLE) ON TABLE for the table
  - (SELECT TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - SELECT ANY TABLE ON DATABASE

- The owner of the created view is determined as follows.

    - The owner of the schema to which the view belongs.
    - If the schema to which the view belongs is PUBLIC, then it is the user who executed the statemen t.

- The owner of the view has the following privileges for the created view.
    - SELECT ON TABLE WITH GRANT OPTION
    - INSERT ON TABLE WITH GRANT OPTION
    - UPDATE ON TABLE WITH GRANT OPTION
    - DELETE ON TABLE WITH GRANT OPTION
    - TRIGGER ON TABLE
    - LOCK ON TABLE WITH GRANT OPTION
    - ALTER ON TABLE WITH GRANT OPTION

Syntax Rules and Parameters

[OR REPLACE]

It replaces the existing view when a view already exists.

[FORCE | NO FORCE]

- FORCE
    - A view is created regardless of the validity of <query expression>.
- NO FORCE
    - A view is created when <Query expression> is valid.
- The default value is NO FORCE.

view_name

It is the view name to be created, and it should be a unique name within the schema.
The schema to which the view belongs, such as schema_name.view_name, can be defined. If schema_na me is omitted, the default schema name of the user performing the statement is used.
The length of the view name must be shorter than 128 bytes.

[(column_name [, ...])]

It defines a column name which will configure the view.
Each column name should be unique within the view.

The number of columns should be as same as the number of result columns in SELECT clause.

If the list of column names is omitted, the column names of SELECT clause in 〈query expression〉 are used.

AS 〈query expression〉

It is the **SELECT** query which will create a view.

〈query expression〉 can not include the following variables.

- Host parameter
- SQL parameter
- Dynamic parameter
- Embedded variable
- SEQUENCE object

Description

A view is the object which gave a name to the query, and it is used in the similar way of a table.

When executing a query including the view, that view is interpreted as the query included in the view defi nition. If the table referenced by the view is altered, as the following example, the asterisk (*) included in the view definition is automatically interpreted based on the altered table information.

```
gSQL> CREATE VIEW v1 AS SELECT * FROM t1;
gSQL> COMMIT;
gSQL> SELECT * FROM v1;
ID NAME
-- ---------
 1 leekmo
 2 mkkim
 3 egonspace
3 rows selected.
gSQL> ALTER TABLE t1 ADD COLUMN ( dept_id  INTEGER,  addr  VARCHAR(1024) );
gSQL> COMMIT;
gSQL> select * from v1;
ID NAME      DEPT_ID ADDR
-- --------- ------- ----
 1 leekmo       null null
 2 mkkim        null null
 3 egonspace    null null
```

```
3 rows selected.
```

The view is affected if the view is created on the query including errors by using FORCE option, the tables referred by the view, or views are altered or deleted.

This information can be retrieved from INFORMATION_SCHEMA.VIEWS information.

- IS_COMPILED column
  - TRUE: The view is normally created.
  - FALSE: The view is created having an error by using FORCE option.
- IS_AFFECTED column
  - TRUE : The referenced tables by a view or view is altered.
  - FALSE: After a view is created and compiled, the referenced tables or views are not altered.

The maximum number of creating views and the maximum number of columns to be created within a vie w is not limited. Therefore, they can be created as many as the storage space is available.

Examples

The following is an example of creating a view.

```
gSQL> CREATE VIEW v1 AS SELECT * FROM t1 WHERE dept_id = 101;
View created.
```

The following is an example of defining the column names when defining a view.

```
gSQL> CREATE VIEW v1 ( v_id, v_name )
          AS SELECT id, name FROM t1 WHERE dept_id = 101;
View created.
```

The following is an example of removing the existing view and creating a new view by using REPLACE op tion.

```
gSQL> CREATE OR REPLACE VIEW v1(id, name)
             AS SELECT id, name FROM t1;
View created.
```

The following is an example of forcing to create a view by using FORCE option even when the referenced object by the view does not exist.

```
gSQL> CREATE FORCE VIEW v1
          AS SELECT * FROM t1 WHERE dept_id = 101;
ERR-01000(16243): Warning: View created with compilation errors
ERR-42000(16040): table or view does not exist :
    AS SELECT * FROM t1 WHERE dept_id = 101
                      *
ERROR at line 2:
View created.
```

Compatibility

The SQL standard does not define the following clauses.

- [OR REPLACE] clause
- [FORCE | NO FORCE] clause

Table 19-13 SQL standard compatibility

Feature ID	Description	Compatibility
T131	Recursive query	O
F751	View CHECK enhancements	X
S043	Enhanced reference types	X
T111	Updatable joins, unions, and columns	X
F852	Top-level <order by clause> in views	O
F864	Top-level <result offset clause> in views	O
F859	Top-level <fetch first clause> in views	O
S081	Subtables	X

For More Information

Refer to the followings.

- DROP VIEW
- ALTER VIEW
- SELECT

19.22 DECLARE cursor_name

Function

It declares a cursor.

Syntax

```
<declare cursor> ::=
    DECLARE cursor_name <cursor properties> { FOR ¦ IS } <cursor specification>
    ;
<cursor properties> ::=
      [ <cursor sensitivity> ] [ <cursor scrollability>] ] CURSOR [ <cursor holdability> ]
    ¦ [ <odbc cursor type] CURSOR [ <cursor holdability> ]
<cursor sensitivity> ::=
      INSENSITIVE
    ¦ SENSITIVE
    ¦ ASENSITIVE
<cursor scrollability> ::=
      NO SCROLL
    ¦ SCROLL
<cursor holdability> ::=
      WITH HOLD
    ¦ WITHOUT HOLD
<odbc cursor type> ::=
      STATIC
    ¦ KEYSET
<cursor specification> ::=
      statement_name
    ¦ <cursor query>  [ <updatability clause> ]
<cursor query> ::=
      <select statement>
    ¦ <insert returning query statement>
    ¦ <update returning query statement>
    ¦ <delete returning query statement>
<updatability clause> ::=
      FOR READ ONLY
```

```
    ¦ FOR UPDATE [ OF <column name list> ] [ <lock wait mode> ]
<lock wait mode> ::=
    ¦ WAIT
    ¦ WAIT second
    ¦ NOWAIT
```

Invocation and Access Rules

The dynamic cursor which uses statement_name can be used in an embedded SQL.

An appropriate access privilege is required depending on <cursor query> types.
For more information about the access privileges, refer to the followings.

- The access privilege of a **SELECT** statement
- The access privilege of a **SELECT .. FOR UPDATE** statement
- The access privilege of a **INSERT INTO name RETURNING** statement
- The access privilege of a **UPDATE name RETURNING** statement
- The access privilege of a **DELETE FROM name RETURNING** statement

Syntax Rules and Parameters

cursor_name

It is the cursor name to be declared.
It should be a unique name within the session.
The length of cursor name should be shorter than 128 bytes.

{ FOR | IS }

Either FOR or IS is used as a syntax keyword in SQL standard.

<cursor properties>

It defines the cursor properties.

- If <cursor sensitivity> is not specified, the default value is INSENSITIVE.
- If <cursor scrollability> is not specified, the default value is NO SCROLL.
- If <cursor holdability> is not specified, the default value is determined by <cursor updatability>.

updatable query

To use a cursor property such as SENSITIVE or FOR UPDATE, a query of the cursor should identify changes in rows of the base table, or it should be the updatable query which can acquire a lock on the row.

The updatable query should satisfy all of the following conditions.

- DISTINCT should not exist at the top level query.
  - (X) SELECT DISTINCT * FROM t1;
- GROUP BY, HAVING, aggregation function should not exist at the top level query.
  - (X) SELECT MAX(c1) FROM t1;
- It should not be a returning query.
  - (X) DELETE FROM t1 RETURNING c1;
- Set operator should not exist.
  - (X) SELECT * FROM t1 UNION ALL SELECT * FROM t2;
- At least one updatable column is required on the listed tables in FROM clause.
  - The columns of the tables, which are included in join operations and are not used in the cross joi n, are not the updatable columns.
    - FULL OUTER JOIN is not cross join.
    - NATURAL JOIN is not cross join.
    - If INNER JOIN uses USING phrase, it is not cross join.
  - The columns of the following tables are not the updatable columns.
    - Dictionary table, fixed table, performance view
  - The columns of the view are not updatable tables.

〈cursor sensitivity〉

It determines whether the following data changes which affects the query results can be queried when o perating the cursor.

- INSENSITIVE
  - It can not detect the data updated during the cursor operation.
- SENSITIVE
  - 〈cursor query〉 should be an updatable query.
  - It detects the updated and deleted data in the transaction as same as that of the cursor.
  - It detects the data updated and deleted through COMMIT of other transactions.
- ASENSITIVE
  - INSENSITIVE or SENSITIVE is determined depending on the type of 〈cursor query〉.
    - For the updatable query, it is SENSITIVE.
    - For no updatable query, it is INSENSITIVE.
- If it is not specified, the default value is INSENSITIVE.

〈cursor scrollability〉

It specifies whether the result set of the cursor can be fetched sequentially or non-sequentially.

- NO SCROLL
  - Only sequential FETCH (FETCH NEXT) is possible.
- SCROLL
  - Non-sequential FETCH is possible.
- If not specified, the default value is NO SCROLL.

〈cursor holdability〉

It determines whether the cursor is maintained after the cursor is OPEN and the transaction is committed.

- WITH HOLD
  - The cursor is maintained after the transaction is committed.
  - It can not be used together with FOR UPDATE statement.
  - It can not be used together with **INSERT INTO name RETURNING** statement.
  - It can not be used together with **UPDATE name RETURNING** statement.
  - It can not be used together with **DELETE FROM name RETURNING** statement.
  - It can not be used in the query including the global temporary table whose table commit action is ON COMMIT DELETE ROWS.

- WITHOUT HOLD
  - When the transaction is committed or rolled back, the cursor is closed.

- Rollback and cursor
  - It closes a cursor included in a transaction when rolling back the transaction.
  - It closes a cursor created since the savepoint when rolling back up to the savepoint.

- If not specified, the default value of 〈cursor holdability〉 is determined by 〈cursor updatability〉.
  - If it is FOR READ ONLY or 〈cursor updatability〉 is not specified, the default value is WITH HOLD.
  - If it is used together with FOR UPDATE statement, the default value is WITHOUT HOLD.

〈odbc cursor type〉

It is the cursor type in the ODBC standard, and it has the SCROLL property.

- STATIC CURSOR
  - It is as same as INSENSITIVE SCROLL in SQL standard.
  - Non-sequential FETCH is possible.
  - It is the static scroll cursor in the ODBC standard.
- KEYSET CURSOR

- It is as same as ASENSITIVE SCROLL in SQL standard.
- It is the keyset-driven scroll cursor in the ODBC standard.
- The property of sensitivity is determined according to the following characteristics.

Table 19-14 Sensitivity according to FOR [UPDATE / READ ONLY] statement and the query type

Updatability	Query type	Sensitivity
FOR UPDATE	Updatable query	SENSITIVE
FOR UPDATE	Non-updatable query	Query error
FOR READ ONLY	Any query	INSENSITIVE
N/A	Updatable query	SENSITIVE
N/A	Non-updatable query	INSENSITIVE

〈cursor specification〉

It defines a query which is a target of the cursor.
If statement_name is used, a dynamic cursor whose query has not been defined is declared.
If 〈cursor query〉 is used, a standing cursor whose query is defined is declared.

statement_name

It is a statement_name to be referenced by the cursor, and it can be used in an embedded SQL.

statement_name should exist before performing 〈declare cursor〉 statement, and the SQL statement refe renced by statement_name should be the query prepared by **PREPARE statement_name** statement.

If it is not a query, an error occurs when executing **OPEN cursor_name** statement.

〈cursor query〉

For more information about available query types in the cursor, refer to the followings.

- SELECT
- SELECT .. FOR UPDATE
- INSERT INTO name RETURNING
- UPDATE name RETURNING
- DELETE FROM name RETURNING

〈updatability clause〉

It specifies whether to change rows by using the cursor.

- FOR READ ONLY
  - It declares a read-only cursor.
- FOR UPDATE
  - It declares a writable cursor.
  - When opening the cursor, the x lock for the corresponding rows is acquired to prevent the chang e by other transactions until the transaction is completed.
  - It can not be used together with WITH HOLD statement.
  - 〈cursor query〉 should be an updatable query.
- If not specified, the default value is FOR READ ONLY.

FOR UPDATE OF …

It lists the columns associated with the lock obtaining when OPENing the cursor.

- If it is the columns listed in FOR UPDATE OF statement
  - It should be an updatable column of the table listed in FROM clause of 〈select statement〉.
  - It acquires the lock for the table of listed columns.
- If only FOR UPDATE statement is used
  - It is the same meaning as listing all updatable columns of the table in FROM clause of 〈select stat ement〉.
  - It acquires the lock for the table of all columns.

〈lock wait mode〉

It is used together with FOR UPDATE clause, and it specifies the lock acquisition method.

- WAIT
  - It acquires the lock for all rows of the query result when OPENing the cursor.
  - It waits until acquiring the lock.
- WAIT second
  - It acquires the lock for all rows of the query result when OPENing the cursor.
  - An error occurs if the lock is not acquired within the specified time.
  - The wait time is in seconds, and it can use the value between 0 and 1000000000.
- NOWAIT
  - It acquires the lock for all rows of the query result when OPENing the cursor.
  - An error occurs if the lock is not immediately acquired.
- If not specified, the default value is WAIT.

Description

When controlling the query property, using DECLARE CURSOR, OPEN, FETCH, CLOSE statements have th e performance burden compared to using the cursor with the ODBC or JDBC statements. It is because usi ng DECLARE CURSOR, OPEN, FETCH, CLOSE statements control the cursor of the server.

Before executing the query, the cursor property can be controlled by ODBC statement and JDBC stateme nt. The SQL cursor property control method by DECLARE CURSOR statement, and cursor property control method by the ODBC standard and the JDBC standard are as follows.

Table 19-15 Controlling the cursor property of ODBC/ JDBC

Property	GOLDILOCKS cursor property	ODBC standard cursor property	JDBC standard cursor property
Sensitivity	INSENSITIVE	SQLSetStmtAttr(stmt, SQL_ATTR_C URSOR_SENSITIVITY, SQL_INSENSITI VE, len)	It can not be set.
	SENSITIVE	SQLSetStmtAttr(stmt, SQL_ATTR_C URSOR_SENSITIVITY, SQL_SENSITIV E, len)	It can not be set.
	ASENSITIVE	SQLSetStmtAttr(stmt, SQL_ATTR_C URSOR_SENSITIVITY, SQL_UNSPECI FIED, len)	It can not be set.
Scrollability	NO SCROLL	SQLSetStmtAttr(stmt, SQL_ATTR_C URSOR_SCROLLABLE, SQL_NONSC ROLLABLE, len)	It can not be set.
	SCROLL	SQLSetStmtAttr(stmt, SQL_ATTR_C URSOR_SCROLLABLE, SQL_SCROLL ABLE, len)	It can not be set.
Holdability	WITHOUT HOLD	It can not be set.	java.sql.Connection::prepareState ment(query, type, conc, ResultSet. CLOSE_CURSORS_AT_COMMIT)
	WITH HOLD	It can not be set.	java.sql.Connection::prepareState ment(query, type, conc, ResultSet. HOLD_CURSORS_OVER_COMMIT)

SQL cursor declaration corresponding to ODBC cursor type is as follows.

Table 19-16 SQL cursor declaration corresponding to ODBC cursor type

ODBC cursor type	SQL cursor declaration
SQLSetStmtAttr(stmt, SQL_ATTR_CURSOR_TYPE, SQL_CURSOR_FORWARD_ONLY, le n)	NO SCROLL CURSOR

ODBC cursor type	SQL cursor declaration
SQLSetStmtAttr(stmt, SQL_ATTR_CURSOR_TYPE, SQL_CURSOR_STATIC, len)	STATIC CURSOR
SQLSetStmtAttr(stmt, SQL_ATTR_CURSOR_TYPE, SQL_CURSOR_KEYSET_DRIVEN, len)	KEYSET CURSOR

SQL cursor declaration corresponding to JDBC cursor type is as follows.

Table 19-17 SQL cursor declaration corresponding to JDBC cursor type

JDBC cursor type	SQL cursor declaration
java.sql.Connection::prepareStatement(query, ResultSet.TYPE_FORWARD_ONLY, conc, hold)	INSENSITIVE NO SCROLL CURSOR
java.sql.Connection::prepareStatement(query, ResultSet.TYPE_SCROLL_INSENSITIVE, conc, hold)	INSENSITIVE SCROLL CURSOR
java.sql.Connection::prepareStatement(query, ResultSet.TYPE_SCROLL_SENSITIVE, conc, hold)	KEYSET CURSOR

Examples

The following is an example of declaring the cursor by using interactive SQL (gsql), and using it.

```
gSQL> DECLARE cur1 CURSOR FOR SELECT id, data FROM t1;
Cursor declared.
gSQL> OPEN cur1;
Cursor is open.
gSQL> \var v_id   INTEGER
gSQL> \var v_data VARCHAR(128)
gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   1 data_1
1 row fetched.
gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   2 data_2
1 row fetched.
gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
---- ------
```

```
   3 data_3
1 row fetched.
gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   4 data_4
1 row fetched.
gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   5 data_5
1 row fetched.
gSQL> FETCH cur1 INTO :v_id, :v_data;
no rows fetched.
gSQL> CLOSE cur1;
Cursor closed.
```

The following is an example of declaring KEYSET cursor, sequentially searching, then completing the tran saction of UPDATE, DELETE statements, and searching for it in the reverse direction.

```
gSQL> DECLARE cur_keyset KEYSET CURSOR FOR SELECT id, data FROM t1;
Cursor declared.
gSQL> OPEN cur_keyset;
Cursor is open.
gSQL> \var v_id   INTEGER
gSQL> \var v_data VARCHAR(128)
gSQL> FETCH NEXT cur_keyset INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   1 data_1
1 row fetched.
gSQL> FETCH NEXT cur_keyset INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   2 data_2
1 row fetched.
gSQL> FETCH NEXT cur_keyset INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   3 data_3
1 row fetched.
```

```
gSQL> FETCH NEXT cur_keyset INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   4 data_4
1 row fetched.
gSQL> FETCH NEXT cur_keyset INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   5 data_5
1 row fetched.
gSQL> FETCH NEXT cur_keyset INTO :v_id, :v_data;
no rows fetched.
gSQL> UPDATE t1 SET data = 'new data_2' WHERE id = 2;
1 row updated.
gSQL> COMMIT;
Commit complete.
gSQL> DELETE FROM t1 WHERE id = 4;
1 row deleted.
gSQL> COMMIT;
Commit complete.
gSQL> FETCH PRIOR cur_keyset INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   5 data_5
1 row fetched.
gSQL> FETCH PRIOR cur_keyset INTO :v_id, :v_data;
no rows fetched.
gSQL> FETCH PRIOR cur_keyset INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   3 data_3
1 row fetched.
gSQL> FETCH PRIOR cur_keyset INTO :v_id, :v_data;
V_ID V_DATA
---- ----------
   2 new data_2
1 row fetched.
gSQL> FETCH PRIOR cur_keyset INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   1 data_1
```

```
1 row fetched.
gSQL> CLOSE cur_keyset;
Cursor closed.
```

The following is an example of declaring SCROLL cursor, and using the cursor through the fetch orientati on.

```
gSQL> DECLARE cur_scroll SCROLL CURSOR FOR SELECT id, data FROM t1;
Cursor declared.
gSQL> OPEN cur_scroll;
Cursor is open.
gSQL> \var v_id   INTEGER
gSQL> \var v_data VARCHAR(128)
gSQL> FETCH LAST cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   5 data_5
1 row fetched.
gSQL> FETCH PRIOR cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   4 data_4
1 row fetched.
gSQL> FETCH FIRST cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   1 data_1
1 row fetched.
gSQL> FETCH ABSOLUTE 3 cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   3 data_3
1 row fetched.
gSQL> FETCH RELATIVE -1 cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   2 data_2
1 row fetched.
gSQL> FETCH ABSOLUTE 3 cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
---- ------
```

```
   3 data_3
1 row fetched.
gSQL> CLOSE cur_scroll;
Cursor closed.
```

Compatibility

<declare cursor> statement has the following differences compared to the SQL standard.

- In SQL standard, the default value of <cursor sensitivity> is ASENSITIVE, but in GOLDILOCKS, the default value is INSENSITIVE.
- The SQL standard does not cover the following <odbc cursor type>.
  - STATIC CURSOR
  - KEYSET CURSOR
- In SQL standard, the default value of <cursor holdability> is WITHOUT HOLD, but in GOLDILOCKS, the default value depends on <cursor updatability>.
- SQL standard can use only <select statement> as <cursor query>, but GOLDILOCKS can use the returning query as follows.
  - **INSERT INTO name RETURNING**
  - **UPDATE name RETURNING**
  - **DELETE FROM name RETURNING**
- In SQL standard, the default value of <cursor updatability> is determined by <select statement>, but in GOLDILOCKS the default value is FOR READ ONLY.
- In SQL standard, <lock wait mode> statement does not exist.

Table 19-18 SQL standard compatibility

Feature ID	Description	Compatibility
F831	Full cursor update	O
T231	Sensitive cursors	O
F791	Insensitive cursors	O
F431	Read-only scrollable cursors	O
T471	Result sets return value	X
T551	Optional key words for default syntax	O
T111	Updatable joins, unions, and columns	X
B031	Basic dynamic SQL	O

For More Information

Refer to the followings.

- OPEN cursor_name
- FETCH cursor_name
- CLOSE cursor_name
- PREPARE statement_name
- SELECT
- SELECT .. FOR UPDATE
- INSERT INTO name RETURNING
- UPDATE name RETURNING
- DELETE FROM name RETURNING

19.23 DELETE FROM

Function

It deletes rows in a table.

Syntax

```
<delete statement: searched> ::=
    DELETE [ FROM ] table_name [ [ AS ] alias_name ]
        [ WHERE <search condition> ]
        [ <result offset clause> ]
        [ <fetch limit clause> ]
    ;
<result offset clause> ::=
    OFFSET skip_count [ ROW ¦ ROWS ]
<fetch limit clause> ::=
      <fetch first clause>
    ¦ <limit clause>
<fetch first clause> ::=
    FETCH [ FIRST ¦ NEXT ] [ row_count ] [ ROW ONLY ¦ ROWS ONLY ]
<limit clause>
    LIMIT { fetch_row_count ¦ offset_row_count, fetch_row_count ¦ ALL }
```

Invocation and Access Rules

One of the following privileges is required to perform <delete statement: searched>.

- (DELETE or CONTROL TABLE) ON TABLE for the table
- (DELETE TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- DELETE ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the name of a target table whose rows are to be deleted.
It defines the schema to which the table belongs such as schema_name.table_name.
If schema_name is omitted, the default schema name of the user performing the statement is used.

[AS alias_name]

It is the alias of table_name.

WHERE 〈search condition〉

It deletes the rows which satisfy WHERE condition.
If WHERE condition is omitted, it deletes all rows.
For more information about WHERE condition, refer to **where clause** of **SELECT** statement.

〈result offset clause〉

It specifies the number of rows to be skipped among the query result.
For more information, refer to **offset limit clause** of **SELECT** statement.

〈fetch limit clause〉

The following two ways are used to specify the number of rows to be fetched.

- 〈fetch first clause〉
  - It specifies the number of rows to be fetched.
  - For more information, refer to **〈fetch first clause〉** of **SELECT** statement.
- 〈limit clause〉
  - It specifies the number of rows to be fetched, or it simultaneously specifies both the number of r ows to be skipped and the number of rows to be fetched.
  - For more information, refer to **〈limit clause〉** of **SELECT** statement.

Description

Differences among DELETE-related Statements

- **DELETE FROM**
  - It deletes multiple rows which satisfy conditions.
  - e.g. DELETE FROM t1 WHERE c1 = 0;
- **DELETE FROM name WHERE CURRENT OF cursor_name**
  - It deletes the row which the current cursor indicates.
  - e.g. DELETE FROM t1 WHERE CURRENT OF cursor;
- **DELETE FROM name RETURNING**
  - It deletes multiple rows which satisfy the conditions, and the deleted rows can be retrieved in the same way as **SELECT** statement (API such as SQLFetch ()).
  - e.g. DELETE FROM t1 WHERE c1 = 0 RETURNING c2;
- **DELETE FROM name RETURNING .. INTO**
  - It deletes row equal to or less than one, and if one row is deleted, the value is obtained into the h ost variable of RETURNING INTO clause.
  - e.g. DELETE FROM t1 WHERE c1 = 0 RETURNING c2 INTO :v1;

Examples

The following is an example of DELETE statement.

```
gSQL> DELETE FROM t1 WHERE id > 3;
2 rows deleted.
```

The following is an example of skipping some rows (two rows) and deleting some rows (two rows) amon g the rows which satisfy the conditions by using <result offset clause> and <fetch first clause> clauses.

```
gSQL> DELETE FROM t1 OFFSET 2 FETCH 2;
2 rows deleted.
gSQL> SELECT * FROM t1 ORDER BY 1;
ID DATA
-- ------
 1 data_1
 2 data_2
 5 data_5
3 rows selected.
```

Compatibility

The SQL standard does not define the following clauses of DELETE statement.

- 〈result offset clause〉
- 〈fetch limit clause〉

Table 19-19 SQL standard compatibility

Feature ID	Description	Compatibility
F781	Self-referencing operations	X
T111	Updatable joins, unions, and columns	X

For More Information

Refer to the followings.

- **DELETE FROM name WHERE CURRENT OF cursor_name**
- **DELETE FROM name RETURNING**
- **DELETE FROM name RETURNING .. INTO**
- **SELECT**

19.24 DELETE FROM name RETURNING

Function

It deletes rows of the table, and retrieves the deleted rows.

Syntax

```
<delete returning query statement> ::=
    DELETE [ FROM ] table_name [ [ AS ] alias_name ]
        [ WHERE <search condition> ]
        [ <result offset clause> ]
        [ <fetch limit clause> ]
        <returning clause>
    ;
<result offset clause> ::=
    OFFSET skip_count [ ROW ¦ ROWS ]
<fetch limit clause> ::=
      <fetch first clause>
    ¦ <limit clause>
<fetch first clause> ::=
    FETCH [ FIRST ¦ NEXT ] [ row_count ] [ ROW ONLY ¦ ROWS ONLY ]
<limit clause>
    LIMIT { fetch_row_count ¦ offset_row_count, fetch_row_count ¦ ALL }
<returning clause> ::=
    { RETURN ¦ RETURNING } { * ¦ { <value expression> [ [AS] alias_name] } [, ...] }
```

Invocation and Access Rules

The user should satisfy the following conditions to perform <delete returning query statement>.

- One of the following privileges is required to perform DELETE statement.
  - (DELETE or CONTROL TABLE) ON TABLE for the table
  - (DELETE TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - DELETE ANY TABLE ON DATABASE

- One of the following privileges is required for all the columns used in RETURNING clause.
  - SELECT(columns) ON TABLE for all columns used in RETURNING clause.
  - (SELECT or CONTROL TABLE) ON TABLE for the table
  - (SELECT TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - SELECT ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the name of a target table whose rows are to be deleted.

[AS alias_name]

It is the alias of table_name.

WHERE 〈search condition〉

It deletes the rows which satisfy WHERE condition.
For more information, refer to **DELETE FROM** statement.

〈result offset clause〉

It specifies the number of rows to be skipped among the query result.
For more information, refer to **DELETE FROM** statement.

〈fetch first clause〉

It specifies the number of rows to be fetched.
For more information, refer to **DELETE FROM** statement.

〈limit clause〉

It specifies the number of rows to be fetched, or it simultaneously specifies both the number of rows to b e skipped and the number of rows to be fetched.
For more information, refer to **DELETE FROM** statement.

〈returning clause〉

It sets the deleted rows as a result set, and it specifies the columns to be searched from the set.

- RETURNING clause returns the rows deleted by DELETE statement as a result set.
- 〈value expression〉
  - It is as same as 〈select list〉 of SELECT statement, but it can not use aggregation.
- [[AS] alias_name]
  - It can give the name to the value expression by using AS clause.

The keywords RETURNING and RETURN have the same meaning.

Description

For more information, refer to **Differences among DELETE-related Statements**.

Examples

The following is an example of deleting rows which satisfy the condition, and searching for the deleted rows.

```
gSQL> DELETE FROM t1 WHERE id > 3 RETURNING *;
ID DATA
-- ------
 4 data_4
 5 data_5
2 rows deleted.
```

The following is an example of querying information of the deleted rows by using operation in RETURNING clause.

```
gSQL> DELETE FROM t1
             WHERE id > 3
             RETURNING 'ID: ' || id || ', DATA: ' || data AS id_data;
ID_DATA
--------------------
ID: 4, DATA: data_4
ID: 5, DATA: data_5
2 rows deleted.
```

Compatibility

The SQL standard does not cover <delete returning query statement>.

For More Information

Refer to the followings.

- DELETE FROM
- DELETE FROM name WHERE CURRENT OF cursor_name
- DELETE FROM name RETURNING .. INTO
- SELECT

19.25 DELETE FROM name RETURNING .. INTO

Function

It deletes a single row from the table, and the value of the deleted row is obtained into the host variable.

Syntax

```
<delete returning query statement> ::=
    DELETE [ FROM ] table_name [ [ AS ] alias_name ]
        [ WHERE <search condition> ]
        [ <result offset clause> ]
        [ <fetch limit clause> ]
        <returning into clause>
    ;
<result offset clause> ::=
    OFFSET skip_count [ ROW ¦ ROWS ]
<fetch limit clause> ::=
      <fetch first clause>
    ¦ <limit clause>
<fetch first clause> ::=
    FETCH [ FIRST ¦ NEXT ] [ row_count ] [ ROW ONLY ¦ ROWS ONLY ]
<limit clause>
    LIMIT { fetch_row_count ¦ offset_row_count, fetch_row_count ¦ ALL }
<returning into clause> ::=
    { RETURN ¦ RETURNING } { * ¦ { <value expression> [ [AS] alias_name] } [, ...] } INTO
variable_name [, ...]
```

Invocation and Access Rules

The user should satisfy the following conditions to perform <delete returning into statement>.

- One of the following privileges is required to perform DELETE statement.
  - (DELETE or CONTROL TABLE) ON TABLE for the table
  - (DELETE TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs

    - DELETE ANY TABLE ON DATABASE

- One of the following privileges is required for all columns used in RETURNING clause.
    - SELECT(columns) ON TABLE for all columns used in RETURNING clause
    - (SELECT or CONTROL TABLE) ON TABLE for the table
    - (SELECT TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
    - SELECT ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the name of a target table whose rows are to be deleted.

[AS alias_name]

It is the alias of table_name.

WHERE 〈search condition〉

It deletes the rows which satisfy WHERE condition.
For more information, refer to **DELETE FROM** statement.

〈result offset clause〉

It specifies the number of rows to be skipped among the query result.
For more information, refer to **DELETE FROM** statement.

〈fetch first clause〉

It specifies the number of rows to be fetched.
For more information, refer to **DELETE FROM** statement.

〈limit clause〉

It specifies the number of rows to be fetched, or it simultaneously specifies both the number of rows to b e skipped and the number of rows to be fetched.
For more information, refer to **DELETE FROM** statement.

〈returning into clause〉

- RETURNING .. AS ..
  - For more information, refer to 〈returning clause〉 of **DELETE FROM name RETURNING** statement.
- INTO variable_name [, ...]
  - The number of variables specified in INTO clause should be equal to the number of the expressio ns specified in RETURNING clause.

Description

The number of rows to be deleted should be equal to or less than one.
If two or more rows are deleted, then an error occurs.

For more information, refer to **Differences among DELETE-related Statements**.

Example

The following is an example of deleting rows and obtaining the value of deleted rows into the host variab le in an interactive SQL (gsql).

```
gSQL> \var v_id    INTEGER
gSQL> \var v_data  VARCHAR(128)
gSQL> DELETE FROM t1 WHERE id = 3 RETURNING id, data INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   3 data_3
1 row deleted.
```

Compatibility

The SQL standard does not cover 〈delete returning into statement〉.

For More Information

Refer to the followings.

- DELETE FROM
- DELETE FROM name WHERE CURRENT OF cursor_name
- DELETE FROM name RETURNING
- SELECT

19.26 DELETE FROM name WHERE CURRENT OF cursor_name

Function

It deletes a single row which the cursor indicates.

Syntax

```
<delete statement: positioned> ::=
    DELETE [ FROM ] table_name [ [ AS ] alias_name ]
        WHERE CURRENT OF cursor_name
    ;
```

Invocation and Access Rules

The privilege to perform **DELETE FROM** statement is required to perform <delete statement: positioned>.

Syntax Rules and Parameters

table_name

It is the name of a target table whose rows are to be deleted.

[AS alias_name]

It is the alias of table_name.

cursor_name

The cursor corresponding to cursor_name should satisfy the following conditions.

- The cursor should be OPEN. (Refer to **OPEN cursor_name**.)

- Fetched rows by using the cursor should exist. (Refer to **FETCH cursor_name**.)
- The query used for the cursor should identify table_name. (Refer to **DECLARE cursor_name**.)
- The cursor should be updatable for table_name. (Refer to **DECLARE cursor_name**.)

Description

For more information, refer to **Differences among DELETE-related Statements**.

Example

The following is an example of declaring the FOR UPDATE cursor, and deleting rows by using the cursor i n interactive SQL (gsql).

```
gSQL> DECLARE cur1 CURSOR FOR SELECT id, data FROM t1 FOR UPDATE;
Cursor declared.
gSQL> OPEN cur1;
Cursor is open.
gSQL> \var v_id   INTEGER
gSQL> \var v_data VARCHAR(128)
gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   1 data_1
1 row fetched.
gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   2 data_2
1 row fetched.
gSQL> DELETE FROM t1 WHERE CURRENT OF cur1;
1 row deleted.
gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   3 data_3
1 row fetched.
gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
```

```
---- ------
   4 data_4
1 row fetched.
gSQL> DELETE FROM t1 WHERE CURRENT OF cur1;
1 row deleted.
gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   5 data_5
1 row fetched.
gSQL> FETCH cur1 INTO :v_id, :v_data;
no rows fetched.
gSQL> CLOSE cur1;
Cursor closed.
gSQL> SELECT id, data FROM t1 ORDER BY 1;
ID DATA
-- ------
 1 data_1
 3 data_3
 5 data_5
3 rows selected.
```

Compatibility

Table 19-20 SQL standard compatibility

Feature ID	Description	Compatibility
S111	ONLY in query expressions	X
B031	Basic dynamic SQL	O

For More Information

Refer to the followings.

- DECLARE cursor_name
- OPEN cursor_name
- FETCH cursor_name
- DELETE FROM
- DELETE FROM name RETURNING

- DELETE FROM name RETURNING .. INTO

19.27 DROP AUDIT POLICY

Function

It drops an audit policy.

Syntax

```
<drop audit policy statement> ::=
    DROP AUDIT POLICY [ IF EXISTS ] policy_name
;
```

Invocation and Access Rules

AUDIT SYSTEM ON DATABASE privilege is required to perform <drop audit policy statement>.

Syntax Rules and Parameters

IF EXISTS

An error does not occur even when a policy_name does not exist.

policy_name

It is the name of an audit policy object to be dropped.

Description

The audit policy object which is already activated can not be dropped. In this case, the audit policy should be deactivated by using NOAUDIT POLICY statement.

Examples

The following is an example of dropping an audit policy.

```
DROP AUDIT POLICY policy_table;
```

Compatibility

In the SQL standard, an audit policy does not exist.

For More Information

Refer to the followings.

- Managing audit policy object
  - **CREATE AUDIT POLICY**
  - **DROP AUDIT POLICY**
  - **ALTER AUDIT POLICY**
- Activating/ deactivating audit policy
  - **AUDIT POLICY**
  - **NOAUDIT POLICY**
- Retrieving audit trail: **AUDIT_TRAIL**
- Clearing audit trail: **ALTER DATABASE CLEAR AUDIT TRAIL**

19.28 DROP CLUSTER GROUP

Function

It drops a cluster group from a cluster system.

Syntax

```
<drop cluster group statement> ::=
    DROP CLUSTER GROUP [IF EXISTS] group_name
    ;
```

Invocation and Access Rules

It can be performed in a cluster system.
ADMINISTRATION ON DATABASE privilege is required to perform <drop cluster group statement>.

Syntax Rules and Parameters

[IF EXISTS]

An error does not occur even when a cluster group does not exist.

group_name

It is the name of a cluster group.
A cluster group without any shard can be dropped.

Description

A cluster group can be dropped only when dropping the cluster group does not cause the data loss.

All members in the target cluster group should be inactive.
Otherwise, the following error occurs.

```
gSQL> DROP CLUSTER GROUP g3;
ERR-42000(16582): there are active cluster members in the target cluster group 'G3'
```

Examples

The following is an example of dropping a cluster group.

```
gSQL> DROP CLUSTER GROUP g3;
Cluster Group dropped.
```

Compatibility

The SQL standard does not define the concepts of the cluster.

For More Information

Refer to **CREATE CLUSTER GROUP**.

19.29 DROP CLUSTER LOCATION

Function

It drops the access information of a cluster member.

Syntax

```
<drop cluster location statement> ::=
    DROP CLUSTER LOCATION member_name
    ;
```

Invocation and Access Rules

It can be performed in a cluster system.
ADMINISTRATION ON DATABASE privilege is required to perform <drop cluster location statement>.

Syntax Rules and Parameters

member_name

It is the name of a cluster member.
The same cluster member name should exist in a registered cluster location information.
The length of the name should be shorter than 128 bytes.

Description

Generally, the information of the cluster location is automatically created by using the connection inform ation provided when creating the cluster group or adding the cluster member. The created information is deleted together when deleting the cluster member and the cluster group.

If the access information of the cluster location is modified, then the connection information can be modi

fied by using **ALTER CLUSTER LOCATION** without deleting or recreating the cluster member.

Example

```
gSQL>
DROP CLUSTER LOCATION g1n2
;
Created
```

Compatibility

The SQL standard does not define the concepts of the cluster.

For More Information

Refer to the followings.

- **CREATE CLUSTER LOCATION**
- **ALTER CLUSTER LOCATION**

19.30 DROP INDEX

Function

It drops an index.

Syntax

```
<drop index statement> ::=
    DROP INDEX [ IF EXISTS ] index_name
    ;
```

Invocation and Access Rules

One of the following privileges is required to perform <drop index statement>.

- The owner of that index
- The owner of the table to which the index belongs
- CONTROL TABLE ON TABLE for the table to which the index belongs
- (DROP INDEX or CONTROL SCHEMA) ON SCHEMA for the schema to which the index belongs
- DROP ANY INDEX ON DATABASE

Syntax Rules and Parameters

IF EXISTS

Even when the index does not exist, an error does not occur.

index_name

It is the index name to be dropped.
It can define schema to which the index belongs such as schema_name.index_name and if schema_name is omitted, the default schema name of the user performing the statement is used.

The indexes created for UNIQUE constraint, PRIMARY KEY constraint can not be dropped.
To drop the indexes created for the constraints above, the constraints should be removed through **ALTER TABLE name DROP CONSTRAINT** statement.

Description

Data Definition Language (DDL) statement such as DROP INDEX can be rolled back if it is before when th e transaction is committed.

Examples

The following is an example of dropping an index.

```
gSQL> DROP INDEX idx_t1_id;
Index dropped.
```

The following is an example of preventing an error even when the index does not exist by using IF EXISTS statement.

```
gSQL> DROP INDEX IF EXISTS not_exist_index;
Index dropped.
```

Compatibility

The SQL standard does not cover the concepts of the index.

For More Information

Refer to the followings.

- **CREATE INDEX**
- **DROP TABLE**
- **ALTER TABLE name DROP CONSTRAINT**

19.31 DROP PROFILE

Function

It drops a profile.

Syntax

```
<drop profile statement> ::=
    DROP PROFILE [ IF EXISTS ] profile_name [ CASCADE ] ;
```

Invocation and Access Rules

DROP PROFILE ON DATABASE privilege is required to perform <drop profile statement>.

Syntax Rules and Parameters

IF EXISTS

Even when the profile does not exist, an error does not occur.

profile_name

It specifies the profile name to be dropped.
It can not drop the DEFAULT profile.

CASCADE

If the profile has already been assigned to users, CASCADE clause should be explicitly specified to drop th e profile.
The profile which is assigned to users and to be dropped is changed to DEFAULT profile.

Example

The following is an example of dropping a profile by using CASCADE statement.

```
gSQL> DROP PROFILE prof CASCADE;
Profile dropped.
gSQL> COMMIT;
Commit complete.
```

Compatibility

The SQL standard does not cover the concepts of the profile.

For More Information

Refer to the followings.

- CREATE PROFILE
- ALTER PROFILE

19.32 DROP SCHEMA

Function

It drops a schema.

Syntax

```
<drop schema statement> ::=
    DROP SCHEMA [ IF EXISTS ] schema_name
        [ <drop behavior> ]
    ;
<drop behavior> ::=
      RESTRICT
    | CASCADE
```

Invocation and Access Rules

One of the following privileges is required to perform <drop schema statement>.

- The owner of that schema
- CONTROL SCHEMA ON SCHEMA for the schema
- DROP SCHEMA ON DATABASE

Syntax Rules and Parameters

IF EXISTS

Even when the schema does not exist, an error does not occur.

schema_name

It is the schema name to be dropped.
However, it can not drop the built-in schema such as "DICTIONARY_SCHEMA", "INFORMATION_SCHEMA" and "PUBLIC" which are automatically created when creating the database.

〈drop behavior〉

- When it is RESTRICT
  - Objects should not exist within the schema.
- When it is CASCADE
  - It drops all objects in the schema together.
- When it is omitted, the default value is RESTRICT.

Description

Data Definition Language (DDL) statement such as DROP SCHEMA can be rolled back if it is before when the transaction is committed. In this case, recycle bin objects which are included in the schema to be dropped are also dropped.

Examples

The following is an example of dropping a schema and all objects which exist within the schema.

```
gSQL> DROP SCHEMA s1 CASCADE;
Schema dropped.
```

The following is an example of preventing an error even when the schema does not exist by using IF EXISTS statement.

```
gSQL> DROP SCHEMA IF EXISTS not_exist_schema;
Schema dropped.
```

Compatibility

The SQL standard does not define IF EXISTS clause.

Table 19-21 SQL standard compatibility

Feature ID	Description	Compatibility
F032	CASCADE drop behavior	O
F381	Extended schema manipulation	O

For More Information

Refer to **CREATE SCHEMA**.

19.33 DROP SEQUENCE

Function

It drops a sequence.

Syntax

```
<drop sequence generator statement> ::=
    DROP SEQUENCE [ IF EXISTS ] [schema_name.] sequence_name
    ;
```

Invocation and Access Rules

One of the following privileges is required to perform <drop sequence generator statement>.

- The owner of that sequence
- (DROP SEQUENCE or CONTROL SCHEMA) ON SCHEMA for the schema to which the sequence belon gs
- DROP ANY SEQUENCE ON DATABASE

Syntax Rules and Parameters

IF EXISTS

Even when the sequence does not exist, an error does not occur.

sequence_name

It is the sequence name to be dropped.
It can define schema to which the sequence belongs such as schema_name.sequence_name and if schem a_name is omitted, the default schema name of the user performing the statement is used.

Description

Data Definition Language (DDL) statement such as DROP SEQUENCE can be rolled back if it is before wh
en the transaction is committed.

Examples

The following is an example of dropping a sequence.

```
gSQL> DROP SEQUENCE seq1;
Sequence dropped.
```

The following is an example of preventing an error even when the sequence does not exist by using IF EXI
STS statement.

```
gSQL> DROP SEQUENCE invalid_sequence;
ERR-42000(16044): sequence does not exist :
DROP SEQUENCE invalid_sequence
              *
ERROR at line 1:
gSQL> DROP SEQUENCE IF EXISTS invalid_sequence;
Sequence dropped.
```

Compatibility

The SQL standard does not define IF EXISTS clause.

Table 19-22 SQL standard compatibility

Feature ID	Description	Compatibility
T176	Sequence generator support	O

For More Information

Refer to the followings.

- CREATE SEQUENCE

- ALTER SEQUENCE

19.34 DROP SYNONYM

Function

It drops a synonym.

Syntax

```
<drop synonym statement> ::=
    DROP [ PUBLIC ] SYNONYM [ IF EXISTS ] [schema_name.]synonym_name
    ;
```

Invocation and Access Rules

DROP PUBLIC SYNONYM ON DATABASE privilege is required to drop a public synonym by specifying PUB LIC.

One of the following privileges is required to drop a private synonym.

- The owner of that synonym
- (DROP SYNONYM or CONTROL SCHEMA) ON SCHEMA for the schema to which the synonym belon gs
- DROP ANY SYNONYM ON DATABASE

Syntax Rules and Parameters

[PUBLIC]

It is specified when dropping the public synonym.
If this clause is omitted, the private synonym is dropped.

IF EXISTS

Even when the synonym does not exist, an error does not occur.

synonym_name

It is the synonym name to be dropped.
It can define schema to which the synonym belongs such as schema_name.synonym_name and if schema _name is omitted, the default schema name of the user performing the statement is used.
If PUBLIC is explicitly specified, the schema name can not be specified.

Description

Data Definition Language (DDL) statement such as DROP SYNONYM can be rolled back if it is before whe n the transaction is committed.

Examples

The following is an example of dropping a private synonym.

```
gSQL> DROP SYNONYM MyEmp;
Synonym dropped.
```

The following is an example of dropping a public synonym.

```
gSQL> DROP PUBLIC SYNONYM MainEmp;
Synonym dropped.
```

Compatibility

The SQL standard does not define DROP SYNONYM statement.

For More Information

Refer to **CREATE SYNONYM**.

19.35 DROP TABLE

Function

It drops a table.

If the recycle bin feature is activated, the table is not completely dropped but it is stored in the r ecycle bin.

Syntax

```
<drop table statement> ::=
    DROP TABLE [ IF EXISTS ] table_name
    [ <drop behavior> ]
    [ PURGE ]
    ;
<drop behavior> ::=
      RESTRICT
    ¦ CASCADE
    ¦ CASCADE CONSTRAINTS
```

Invocation and Access Rules

One of the following privileges is required to perform <drop table statement>.

- The owner of that table
- CONTROL TABLE ON TABLE for that table
- (DROP TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- DROP ANY TABLE ON DATABASE

Syntax Rules and Parameters

IF EXISTS

Even when the table does not exist, an error does not occur.

table_name

It is the table name to be dropped.
It can define schema to which the table belongs such as schema_name.table_name and if schema_name is omitted, the default schema name of the user performing the statement is used.

The following tables which are automatically created during creating the database, can not be dropped.

- The tables in "DEFINITION_SCHEMA" schema
- The tables in "FIXED_TABLE_SCHEMA" schema

It also drops constraints and indexes created in the table.

drop behavior

Currently, both RESTRICT and CASCADE are operated same.
When it is omitted, the default value is RESTRICT.

purge

It immediately drops a table instead of storing it in the recyclebin even when the recyclebin feature is activated.

Description

Data Definition Language (DDL) statement such as DROP TABLE can be rolled back if it is before when the transaction is committed.

Examples

The following is an example of dropping an ordinary table.

```
gSQL> DROP TABLE region;
Table dropped.
```

The following is an example of preventing an error even when the table does not exist by using IF EXISTS statement.

```
gSQL> DROP TABLE IF EXISTS invalid_table;
Table dropped.
```

The following is an example of rolling back the dropped table.

```
gSQL> SELECT r_regionkey, r_name FROM region;
R_REGIONKEY R_NAME
----------- -------------------------
          0 AFRICA
          1 AMERICA
          2 ASIA
          3 EUROPE
          4 MIDDLE EAST
5 rows selected.
gSQL> DROP TABLE region;
Table dropped.
gSQL> SELECT r_regionkey, r_name FROM region;
ERR-42000(16040): table or view does not exist :
SELECT r_regionkey, r_name FROM region
                               *
ERROR at line 1:
gSQL> ROLLBACK;
Rollback complete.
gSQL> SELECT r_regionkey, r_name FROM region;
R_REGIONKEY R_NAME
----------- -------------------------
          0 AFRICA
          1 AMERICA
          2 ASIA
          3 EUROPE
          4 MIDDLE EAST
```

```
5 rows selected.
```

Compatibility

The SQL standard does not define the following clauses.

- IF EXISTS
- CASCADE CONSTRAINTS

Table 19-23 SQL standard compatibility

Feature ID	Description	Compatibility
F032	CASCADE drop behavior	O

For More Information

Refer to **CREATE TABLE**.

19.36 DROP TABLESPACE

Function

It drops a tablespace.

Syntax

```
<drop tablespace statement> ::=
    DROP TABLESPACE [ IF EXISTS ] tablespace_name
        [ INCLUDING CONTENTS ]
        [ { AND | KEEP } DATAFILES ]
        [ <drop behavior> ]
    ;
<drop behavior> ::=
      RESTRICT
    | CASCADE
    | CASCADE CONSTRAINTS
```

Invocation and Access Rules

DROP TABLESPACE ON DATABASE privilege is required to perform <drop tablespace definition>.

Syntax Rules and Parameters

IF EXISTS

Even when the tablespace does not exist, an error does not occur.

tablespace_name

It is the tablespace name to be dropped.
The following system tablespaces which are automatically created during creating the database, can not

be dropped.

- DICTIONARY_TBS: system tablespace for dictionary management
- MEM_UNDO_TBS: system tablespace for default undo tablespace
- MEM_DATA_TBS: system tablespace for default user data tablespace
- MEM_TEMP_TBS: system tablespace for default temporary tablespace

If tablespace_name was used as a default tablespace of a user, the space for the objects can not be allocated after dropping the tablespace.
After dropping the tablespace, the default tablespace should be changed by using **ALTER USER** statement.

INCLUDING CONTENTS

It drops objects (table, index, key constraint) which belong to the tablespace. If the index or key constrain t which refers to the table which belongs to the tablespace exists outside of the tablespace, then it is also dropped.

If INCLUDING CONTENTS clause is not used, then any object which belongs to the tablespace should not exist.

[{ AND | KEEP } DATAFILES]

It specifies whether to drop the datafiles which configure the tablespace together.
The datafiles are not in the memory temporary tablespace, so the clause is ignored.

- AND DATAFILES
  - It drops the datafiles together.
- KEEP DATAFILES
  - It does not drop the datafiles, but keeps them.
- If it is not specified, the default value is KEEP DATAFILES.

drop behavior

Currently, both RESTRICT and CASCADE are operated same.
When it is omitted, the default value is RESTRICT.

Description

Unlike other Data Definition Language (DDL), DROP TABLESPACE statement can not be rolled back and t he executed transaction is automatically committed. In this case, recycle bin objects which are included in the tablespace to be dropped are also dropped.

Examples

The following is an example of dropping a tablespace together with all objects in the tablespace and data files which configure the tablespace.

```
gSQL> DROP TABLESPACE space1 INCLUDING CONTENTS AND DATAFILES CASCADE CONSTRAINTS;
Tablespace dropped.
```

The following is an example of preventing an error even when the tablespace does not exist by using IF E XISTS statement.

```
gSQL> DROP TABLESPACE IF EXISTS not_exist_tablespace;
Tablespace dropped.
```

Compatibility

The SQL standard does not cover the concepts of the tablespace.

For More Information

Refer to the followings.

- CREATE MEMORY DATA TABLESPACE
- CREATE MEMORY TEMPORARY TABLESPACE
- ALTER TABLESPACE

19.37 DROP USER

Function

It drops a database user.

Syntax

```
<drop user statement> ::=
    DROP USER [ IF EXISTS ] user_identifier [ <drop behavior> ]
    ;
<drop behavior> ::=
      RESTRICT
    ¦ CASCADE
```

Invocation and Access Rules

DROP USER ON DATABASE privilege is required to perform <drop user statement>.

The schema owned by user_identifier should not exist.
For more information about dropping the schema, refer to **DROP SCHEMA**.

Syntax Rules and Parameters

IF EXISTS

Even when the user does not exist, an error does not occur.

user_identifier

It is the database username to be dropped.
However, the user which is automatically created during creating the database such as "SYS", can not be

dropped.

It does not drop the object which is created by user_identifier but is not an owner as follows.

- Role
- Tablespace

〈drop behavior〉

- RESTRICT
  - There should not be the following user owned SQL schema objects.
    - Table, view
    - Index
    - Sequence
    - Table constraint
- CASCADE
  - It drops all the following user owned SQL schema objects.
    - Table, view
    - Index
    - Sequence
    - Table constraint
- When it is omitted, the default value is RESTRICT.

The relationship between user and schema in other DBMS

- Oracle
  - User : schema = 1 : 1.
  - When CASCADE, the schema is also dropped.
- DB2
  - It is as same as the OS user.
  - The separate SQL statements which create or delete a user do not exist.
- Postgres
  - User : schema = 1 : N.
  - It does not support CASCADE option, and a user can be dropped after dropping all the objects owned by the user and all the privileges granted to other users.

> - MySQL
>   - Database : schema = 1 : 1.
>   - User is a subordinate object of database(schema), and it does not support CASCADE option.

Description

In GOLDILOCKS, relationship between the user and the schema is 1 : N. A user does not own a schema, or the user can have multiple schemas.

To drop a user, all schema owned by the user should be dropped. In this case, recycle bin objects of the user to be dropped are also dropped.

Examples

The following is an example of dropping all schema owned by the user and then dropping the user.

```
gSQL> DROP SCHEMA u1 CASCADE;
Schema dropped.
gSQL> DROP USER u1 CASCADE;
User dropped.
```

The following is an example of preventing an error even when the user does not exist by using IF EXISTS statement.

```
gSQL> DROP USER IF EXISTS not_exist_user;
User dropped.
```

Compatibility

The SQL standard covers the concepts of the user, but it does not define the SQL statements related to creating or dropping a user.

For More Information

Refer to the followings.

- CREATE USER
- ALTER USER
- DROP SCHEMA

19.38 DROP VIEW

Function

It drops a view.

Syntax

```
<drop view statement> ::=
    DROP VIEW [ IF EXISTS ] view_name
    ;
```

Invocation and Access Rules

One of the following privileges is required to perform <drop view statement>.

- The owner of that view
- CONTROL TABLE ON TABLE for that view
- (DROP VIEW or CONTROL SCHEMA) ON SCHEMA for the schema to which the view belongs
- DROP ANY VIEW ON DATABASE

Syntax Rules and Parameters

IF EXISTS

Even when the view does not exist, an error does not occur.

view_name

It is the view name to be dropped.
The schema to which the table belongs, such as schema_name.view_name, can be defined. If schema_name is omitted, the default schema name of the user performing the statement is used.

Description

Data Definition Language (DDL) statement such as DROP VIEW can be rolled back if it is before when the transaction is committed.

Examples

The following is an example of dropping a view.

```
gSQL> DROP VIEW v1;
View dropped.
```

The following is an example of preventing an error even when the view does not exist by using IF EXISTS statement.

```
gSQL> DROP VIEW IF EXISTS not_exist_view;
View dropped.
```

Compatibility

The SQL standard does not define IF EXISTS clause.

Table 19-24 SQL standard compatibility

Feature ID	Description	Compatibility
F032	CASCADE drop behavior	X

For More Information

Refer to the followings.

- CREATE VIEW
- ALTER VIEW

19.39 EXECUTE IMMEDIATE 'sql_string'

Function

It executes a dynamic SQL statement which was not defined when writing a program.

Syntax

```
<execute immediate statement> ::=
    EXECUTE IMMEDIATE <SQL statement variable>
    ;
<SQL statement variable> ::=
      variable_name
    ¦ 'sql statement'
    ¦ "sql statement"
    ¦ sql statement
```

Invocation and Access Rules

It can be used in an embedded SQL.
An appropriate privilege according to the type of a dynamic SQL statement is required.

Syntax Rules and Parameters

<SQL statement variable>

The dynamic SQL statement referenced by <SQL statement variable> can not use a host variable (:var) or parameter marker (?).

The following four types of <SQL statement variable> can be used.

- variable_name: It is a variable in which an SQL statement is stored.
- 'sql statement': It is an SQL statement which is enclosed with a single quote (').
- "sql statement": It is an SQL statement which is enclosed with double quotes (").

- sql statement: It is an SQL statement without quote.

The single quote (') is used twice as follows to represent string data within a single-quoted string.

```
{
    ...
    EXEC SQL EXECUTE IMMEDIATE 'INSERT INTO t1 VALUES ( ''literal data'' )';
    ...
}
```

If the SQL statement is a query including a query result, it is successfully executed, but the result can not be obtained.

variable_name

The type corresponding to the variable_name should be a character string.
The dynamic SQL statement defined in the variable_name should be valid.

sql statement

The dynamic SQL statement defined in the sql statement should be valid.

Description

EXECUTE IMMEDIATE 'sql_string' statement can be used as the non-query SQL without a host variable in dynamic embedded SQL application. It is appropriate to execute DDL, DML as one-off because it does not require separate preparation procedure.

For more information, refer to **Embedded Dynamic SQL**.

Example

The following is an example of using EXECUTE IMMEDIATE 'sql_string' in the embedded SQL source cod e.

```
{
    ...
    sprintf(sSqlStmt, "INSERT INTO EMP_RND\n"
```

```
            "SELECT *\n"
            "FROM   EMP\n"
            "WHERE  JOB = 'RND'\n" );
    EXEC SQL EXECUTE IMMEDIATE :sSqlStmt;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    ...
}
```

The full source code in which EXECUTE IMMEDIATE 'sql_string' was used can be viewed in **Dynamic Embedded SQL Example Program**.

Compatibility

Table 19-25 SQL standard compatibility

Feature ID	Description	Compatibility
B031	Basic Dynamic SQL	O

For More Information

Refer to the followings.

- **PREPARE statement_name**
- **EXECUTE statement_name**
- **Embedded Dynamic SQL**

19.40 EXECUTE statement_name

Function

It executes the prepared statement.

Syntax

```
<execute statement> ::=
    EXECUTE statement_name [ <parameter using clause> ] [ <result into clause> ]
    ;
<parameter using clause> ::=
      <using parameter arguments>
<using parameter arguments> ::=
    USING variable_name [, ...]
<result into clause> ::=
      <into result arguments>
<into result arguments> ::=
    INTO variable_name [, ...]
```

Invocation and Access Rules

It can be used in an embedded SQL.
An appropriate privilege according to the type of a dynamic SQL statement is required.

Syntax Rules and Parameters

statement_name

It is the name of a prepared statement.
Statement_name should be prepared by using **PREPARE statement_name**.

If the dynamic SQL statement referenced by statement_name contains a dynamic parameter, <parameter using clause> should explicitly be specified.

```
{
    ...
    EXEC SQL PREPARE stmt1 FROM 'DELETE FROM t1 WHERE c1 > ?';
    EXEC SQL EXECUTE stmt1 USING :sValue;
    ...
}
```

```
{
    ...
    EXEC SQL PREPARE stmt1 FROM 'SELECT COUNT(*) INTO :v1 FROM t1';
    EXEC SQL EXECUTE stmt1 USING :sValue;
    ...
}
```

If the dynamic SQL statement referenced by statement_name is a query or a stored function including the result, <result into clause> should explicitly be specified.

```
{
    ...
    EXEC SQL PREPARE stmt1 FROM 'SELECT COUNT(*) FROM t1';
    EXEC SQL EXECUTE stmt1 INTO :sValue;
    ...
}
```

If there are multiple queries they are normally executed, but only the first query can get the result.
To get multiple results, the following statements related to the cursor should be used.

- DECLARE cursor_name
- OPEN cursor_name
- FETCH cursor_name
- CLOSE cursor_name

If there is not any query result, it is completed as NO DATA.

[<parameter using clause>] [<result into clause>]

<parameter using clause> and <result into clause> can be specified in any order, but they should not be repeated.

〈parameter using clause〉

If any parameter exists in a dynamic SQL statement referenced by statement_name, the parameter inform ation is specified with 〈using parameter arguments〉 clause.

〈using parameter arguments〉

If 〈using parameter arguments〉 statement is used, the number of variable_name should be equal to the number of the parameter included in the dynamic SQL statement referenced by statement_name.

The listed variable_name corresponds to the dynamic parameter in an order of its description.

```
{
    ...
    EXEC SQL PREPARE stmt1 FROM 'DELETE FROM t1 WHERE c1 IN ( ?, ?, ? )';
    EXEC SQL EXECUTE stmt1 USING :sValue1, :sValue2, :sValue3;
    ...
}
```

〈result into clause〉

If the dynamic SQL statement referenced by statement_name is a query, the information about the result columns is specified with 〈into result arguments〉 clause.

If the result is null and INDICATOR is not specified, [DATA EXCEPTION, NULL VALUE, NO INDICATOR PAR AMETER] error occurs.

〈into result arguments〉

If 〈into result arguments〉 clause is used, the number of variable_name should be equal to the number of the result column in the dynamic SQL statement referenced by statement_name.

The listed variable_name corresponds to the dynamic parameter in an order of its description.

```
{
    ...
    EXEC SQL PREPARE stmt1 FROM 'SELECT MIN(salary), MAX(salary), AVG(salary) FROM employee';
    EXEC SQL EXECUTE stmt1 INTO :sMinValue, :sMaxValue, :sAvgValue;
    ...
}
```

Description

Statement_name is an identifier which informs the precompiler the statement in an embedded SQL sourc e code.
A separate type or declaration is not required because statement_name is not a host variable. EXECUTE st atement_name should be written after PREPARE statement_name.

For more information, refer to **Embedded Dynamic SQL**.

Example

The following is an example of using EXECUTE statement_name in an embedded SQL source code.

```
{
    ...
    sprintf( sUpdateSql, "UPDATE EMP SET sal = sal * :v1 WHERE JOB = 'SALES'");
    EXEC SQL PREPARE UPDATE_STMT FROM :sUpdateSql;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    sRatio = 1.1;
    EXEC SQL EXECUTE UPDATE_STMT USING :sRatio;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    ...
}
```

The full source code in which EXECUTE statement_name was used can be viewed in **Dynamic Embedded SQL Example Program**.

Compatibility

Table 19-26 SQL standard compatibility

Feature ID	Description	Compatibility
B031	Basic Dynamic SQL	O
B032	Extended dynamic SQL	X

For More Information

Refer to the followings.

- **PREPARE statement_name**
- **DECLARE cursor_name**
- **OPEN cursor_name**
- **FETCH cursor_name**
- **CLOSE cursor_name**
- **Embedded Dynamic SQL**

19.41 FETCH cursor_name

Function

It locates the cursor on a specific row of result set, and obtains the value of that row to a host variable.

Syntax

```
<fetch statement> ::=
    FETCH [ <fetch orientation> ] [ FROM ] cursor_name
        <result into clause>
    ;
<fetch orientation> ::=
      NEXT
    ¦ PRIOR
    ¦ FIRST
    ¦ LAST
    ¦ CURRENT
    ¦ ABSOLUTE position
    ¦ RELATIVE position
<result into clause> ::=
      <into result arguments>
<into result arguments> ::=
    INTO variable_name [, ...]
```

Syntax Rules and Parameters

[FROM] cursor_name

It should be an open cursor in a session.
FROM can be omitted.

〈fetch orientation〉

To use 〈fetch orientation〉 other than FETCH NEXT, a scrollable cursor should be used.
If 〈fetch orientation〉 is omitted, the default value is NEXT.

The open cursor has the cursor position information for the result set as follows.

Figure 1 The position of cursor

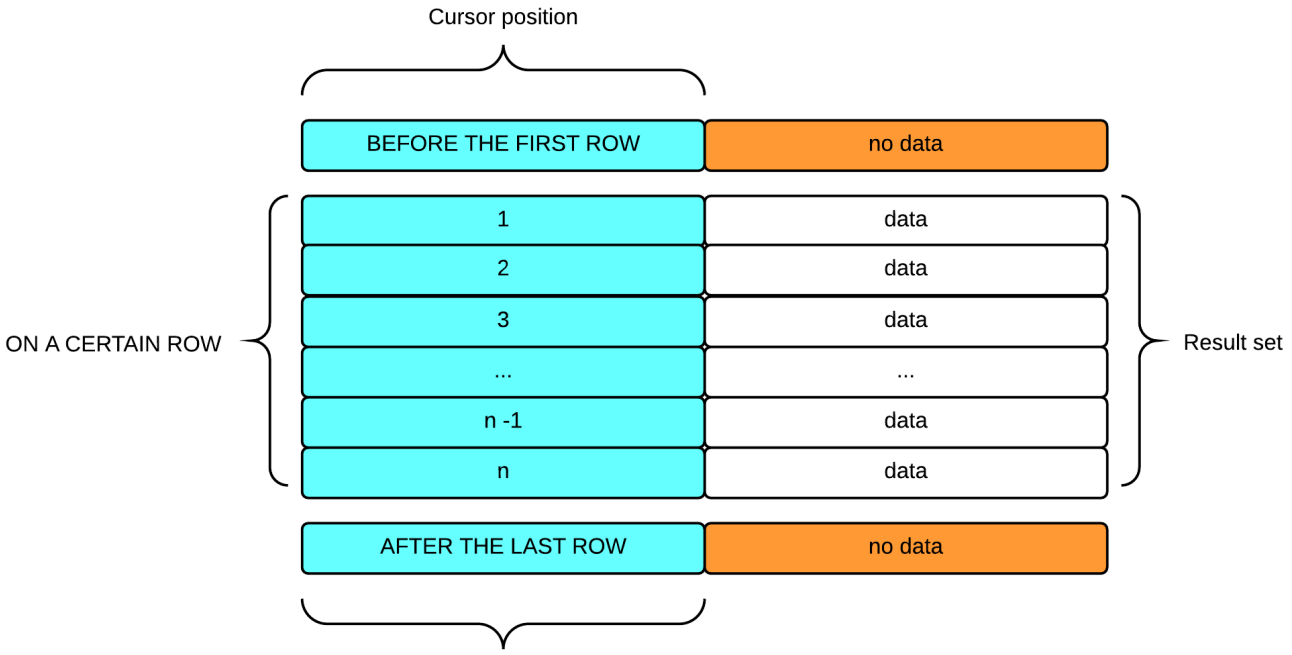


Table 19-27 The position of cursor

Cursor position	Description
BEFORE THE FIRST ROW	The cursor is positioned before the first row of the result set. It is also the cursor positi on when opening it.
ON A CERTAIN ROW	The cursor is positioned on a certain row of the result set through FETCH.
AFTER THE LAST ROW	The cursor is positioned after the last row of the result set.

Each 〈fetch orientation〉 operates based on the cursor position as follows.

- NEXT: It searches for the row next to the current position.
- PRIOR: It searches for the row prior to the current position.
- FIRST: It searches for the first row of the result set.
- LAST: It searches for the last row of the result set.
- CURRENT: It searches for the row in the current position.
- ABSOLUTE position
  - It searches for the row corresponding to the position from the result set.
  - If the position value is negative, it searches for the row at previous position from AFTER THE LAS T ROW.

- RELATIVE position
  - It searches for the row apart as much as position from the current position.

〈result into clause〉

The variable information to obtain the result column is specified by using 〈into result arguments〉.

If the result is null and INDICATOR is not specified, [DATA EXCEPTION, NULL VALUE, NO INDICATOR PAR AMETER] error occurs.

〈into result arguments〉

The number of variables in INTO clause should be as same as the number of columns in the result set of t he cursor.

Description

If the cursor is BEFORE THE FIRST ROW or AFTER THE LAST ROW after performing FETCH, it is positioned at the same position regardless of the entered position in 〈fetch orientation〉.

Example

The following is an example of declaring SCROLL cursor by using the interactive SQL (gsql), then operatin g the various 〈fetch orientation〉.

```
gSQL> DECLARE cur_scroll SCROLL CURSOR FOR SELECT id, data FROM t1;
Cursor declared.
gSQL> OPEN cur_scroll;
Cursor is open.
gSQL> \var v_id   INTEGER
gSQL> \var v_data VARCHAR(128)
gSQL> FETCH NEXT cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   1 data_1
1 row fetched.
gSQL> FETCH NEXT cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
```

```
---- ------
   2 data_2
1 row fetched.
gSQL> FETCH PRIOR cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   1 data_1
1 row fetched.
gSQL> FETCH PRIOR cur_scroll INTO :v_id, :v_data;
no rows fetched.
gSQL> FETCH FIRST cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   1 data_1
1 row fetched.
gSQL> FETCH FIRST cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   1 data_1
1 row fetched.
gSQL> FETCH LAST cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   5 data_5
1 row fetched.
gSQL> FETCH LAST cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   5 data_5
1 row fetched.
gSQL> FETCH FIRST cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   1 data_1
1 row fetched.
gSQL> FETCH CURRENT cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   1 data_1
1 row fetched.
gSQL> FETCH LAST cur_scroll INTO :v_id, :v_data;
```

```
V_ID V_DATA
---- ------
   5 data_5
1 row fetched.
gSQL> FETCH CURRENT cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   5 data_5
1 row fetched.
gSQL> FETCH ABSOLUTE 3 cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   3 data_3
1 row fetched.
gSQL> FETCH CURRENT cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   3 data_3
1 row fetched.
gSQL> FETCH ABSOLUTE 1 cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   1 data_1
1 row fetched.
gSQL> FETCH ABSOLUTE -1 cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   5 data_5
1 row fetched.
gSQL> FETCH ABSOLUTE 6 cur_scroll INTO :v_id, :v_data;
no rows fetched.
gSQL> FETCH ABSOLUTE -6 cur_scroll INTO :v_id, :v_data;
no rows fetched.
gSQL> FETCH ABSOLUTE 3 cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   3 data_3
1 row fetched.
gSQL> FETCH ABSOLUTE -3 cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
---- ------
```

```
   3 data_3
1 row fetched.
gSQL> FETCH RELATIVE 1 cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   4 data_4
1 row fetched.
gSQL> FETCH RELATIVE -1 cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   3 data_3
1 row fetched.
gSQL> FETCH RELATIVE 5 cur_scroll INTO :v_id, :v_data;
no rows fetched.
gSQL> FETCH RELATIVE -5 cur_scroll INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   1 data_1
1 row fetched.
gSQL> CLOSE cur_scroll;
Cursor closed.
```

Compatibility

The SQL standard does not define CURRENT among <fetch orientation>.

Table 19-28 SQL standard compatibility

Feature ID	Description	Compatibility
F431	Read-only scrollable cursors	O
B031	Basic dynamic SQL	O

For More Information

Refer to the followings.

- **DECLARE cursor_name**
- **OPEN cursor_name**
- **CLOSE cursor_name**

19.42 FLASHBACK TABLE

Function

It restores the table object which is stored in the recycle bin.

Syntax

```
<flashback table statement> ::=
    FLASHBACK TABLE table_name
    TO BEFORE DROP [ RENAME TO new_table_name ]
    ;
```

Invocation and Access Rules

One of the following privileges is required to perform <flashback table statement>.

- The owner of that table
- CONTROL TABLE ON TABLE for that table
- (DROP TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- DROP ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the name of the object stored or of the dropped table in the recycle bin.
It can define the schema to which the table belongs in the dropped table name, such as schema_name.table_name. If schema_name is omitted, the default schema name of the user performing the statement is used.

new_table_name

It is a new name of the table to be restored.
The duplicate table name should not exist within a single schema.

Description

It restores the table object which is stored in the recycle bin by using the object name or the dropped tabl e name stored in the recycle bin. If the name which is as same as that of the dropped table exists, then th e newest table object is restored.

If the name which is as same as that of the table object to be restored exists, then an error occurs, but it c an be restored with the new name by using RENAME TO clause. The constraints and the indexes of the re stored tables are restored in its name of when before they were dropped. However, if their names of whe n before the constraints and the indexes were dropped already exist, then the object is restored in the na me of when it is stored in the recycle bin.

Unlike other Data Definition Language (DDL), FLASHBACK TABLE statement can not be rolled back and t he executed transaction is automatically committed.

Example

The following is an example of restoring a table with the name of the object stored in the recycle bin.

```
gSQL> SELECT SCHEMA_NAME, OBJECT_NAME, ORIGINAL_NAME, OBJECT_TYPE FROM USER_RECYCLEBIN;
SCHEMA_NAME OBJECT_NAME                          ORIGINAL_NAME OBJECT_TYPE
----------- ------------------------------------ ------------- -----------
PUBLIC      BIN$106A4F90165D11EA9C5C835D3E4BBBF7 T1            TABLE
1 row selected.
gSQL> FLASHBACK TABLE "BIN$106A4F90165D11EA9C5C835D3E4BBBF7" TO BEFORE DROP;
Flashback complete.
```

The following is an example of restoring the table from the recycle bin in the name of when before it is dr opped.

```
gSQL> SELECT SCHEMA_NAME, OBJECT_NAME, ORIGINAL_NAME, OBJECT_TYPE FROM USER_RECYCLEBIN;
SCHEMA_NAME OBJECT_NAME                          ORIGINAL_NAME OBJECT_TYPE
----------- ------------------------------------ ------------- -----------
PUBLIC      BIN$106A4F90165D11EA9C5C835D3E4BBBF7 T1            TABLE
```

```
gSQL> FLASHBACK TABLE T1 TO BEFORE DROP;
Flashback complete.
```

Compatibility

The SQL standard does not define <flashback table statement>.

For More Information

Refer to the followings.

- **Managing Recycle Bin of Table**
- **PURGE**

19.43 GRANT privileges TO

Function

It grants privileges to a user.

Syntax

```
<grant privilege statement> ::=
    GRANT <privilege> TO <grantee> [, ...]
        [ WITH GRANT OPTION ]
    ;
<grantee> ::=
      PUBLIC
    ¦ user_identifier
    ;

<privilege> ::=
      <database privilege>
    ¦ <tablespace privilege>
    ¦ <schema privilege>
    ¦ <table privilege>
    ¦ <sequence privilege>
    ¦ <procedure privilege>
<database privilege> ::=
      ALL [ PRIVILEGES ] [ON DATABASE]
    ¦ <database action> [, ...] [ON DATABASE]
<database action> ::=
      ADMINISTRATION
    ¦ ANALYZE ANY
    ¦ ALTER DATABASE
    ¦ ALTER SYSTEM
    ¦ AUDIT SYSTEM
    ¦ ACCESS CONTROL
    ¦ CREATE SESSION
    ¦ CREATE PROFILE
    ¦ ALTER PROFILE
```

```
| DROP PROFILE
| CREATE USER
| ALTER USER
| DROP USER
| CREATE ROLE
| ALTER ROLE
| DROP ROLE
| CREATE TABLESPACE
| ALTER TABLESPACE
| DROP TABLESPACE
| USAGE TABLESPACE
| CREATE SCHEMA
| ALTER SCHEMA
| DROP SCHEMA
| CREATE PUBLIC SYNONYM
| DROP PUBLIC SYNONYM
| CREATE ANY TABLE
| ALTER ANY TABLE
| DROP ANY TABLE
| SELECT ANY TABLE
| INSERT ANY TABLE
| DELETE ANY TABLE
| UPDATE ANY TABLE
| LOCK ANY TABLE
| CREATE ANY VIEW
| DROP ANY VIEW
| CREATE ANY SEQUENCE
| ALTER ANY SEQUENCE
| DROP ANY SEQUENCE
| USAGE ANY SEQUENCE
| CREATE ANY INDEX
| ALTER ANY INDEX
| DROP ANY INDEX
| CREATE ANY SYNONYM
| DROP ANY SYNONYM
| CREATE ANY PROCEDURE
| ALTER ANY PROCEDURE
| DROP ANY PROCEDURE
| EXECUTE ANY PROCEDURE
| CREATE ANY PACKAGE
| ALTER ANY PACKAGE
```

```
    ¦ DROP ANY PACKAGE
    ¦ EXECUTE ANY PACKAGE
    ¦ PURGE DBA_RECYCLEBIN
<tablespace privilege> ::=
      ALL [ PRIVILEGES ] ON TABLESPACE tablespace_name
    ¦ <tablespace action> [, ...] ON TABLESPACE tablespace_name
<tablespace action> ::=
    CREATE OBJECT
<schema privilege> ::=
      ALL [ PRIVILEGES ] ON SCHEMA schema_name
    ¦ <schema action> [, ...] [ON SCHEMA schema_name]
<schema action> ::=
      CONTROL SCHEMA
    ¦ CREATE TABLE
    ¦ ALTER TABLE
    ¦ DROP TABLE
    ¦ SELECT TABLE
    ¦ INSERT TABLE
    ¦ DELETE TABLE
    ¦ UPDATE TABLE
    ¦ LOCK TABLE
    ¦ CREATE VIEW
    ¦ DROP VIEW
    ¦ CREATE SEQUENCE
    ¦ ALTER SEQUENCE
    ¦ DROP SEQUENCE
    ¦ USAGE SEQUENCE
    ¦ CREATE INDEX
    ¦ ALTER INDEX
    ¦ DROP INDEX
    ¦ ADD CONSTRAINT
    ¦ CREATE SYNONYM
    ¦ DROP SYNONYM
    ¦ CREATE PROCEDURE
    ¦ ALTER PROCEDURE
    ¦ DROP PROCEDURE
    ¦ EXECUTE PROCEDURE
    ¦ CREATE PACKAGE
    ¦ ALTER PACKAGE
    ¦ DROP PACKAGE
    ¦ EXECUTE PACKAGE
```

```
<table privilege> ::=
      ALL [ PRIVILEGES ] ON [TABLE] table_name
    ¦ { <table action> ¦ <column action> } [, ...] ON [TABLE] table_name
<table action> ::=
      CONTROL TABLE
    ¦ SELECT
    ¦ INSERT
    ¦ UPDATE
    ¦ DELETE
    ¦ REFERENCES
    ¦ LOCK
    ¦ INDEX
    ¦ ALTER
<column action> ::=
      SELECT ( column_name [, ...] )
    ¦ INSERT ( column_name [, ...] )
    ¦ UPDATE ( column_name [, ...] )
    ¦ REFERENCES ( column_name [, ...] )
<sequence privilege> ::=
      ALL [ PRIVILEGES ] ON SEQUENCE sequence_name
    ¦ <sequence action> ON SEQUENCE sequence_name
<sequence action> ::=
    USAGE
<procedure privilege> ::=
      ALL [ PRIVILEGES ] ON PROCEDURE procedure_name
    ¦ <procedure action> ON PROCEDURE procedure_name
<procedure action> ::=
    EXECUTE
<package privilege> ::=
      ALL [ PRIVILEGES ] ON PACKAGE package_name
    ¦ <package action> ON PACKAGE package_name
<package action> ::=
    EXECUTE
```

Syntax Rules and Parameters

〈grantee〉

It is the user to be granted the privileges.

- user_identifier
  - It grants the privilege to a user.
- PUBLIC
  - It is an authorization object which means all users.

WITH GRANT OPTION

It allows the grantee to grant the privilege to other users.

When the same 〈privilege〉 is granted as follows, WITH GRANT OPTION is maintained.

- GRANT SELECT ON t1 TO u1 WITH GRANT OPTION;
- GRANT SELECT ON t1 TO u1;

〈privilege〉

It is a privilege which is to be granted to a grantee (the user to be granted the privilege).

The grantor (the user to perform the statement) should satisfy one of the following conditions.

- The grantor owns that 〈privilege〉 by using WITH GRANT OPTION.
  - The grantor is the user who performs the statement.
- The grantor owns the ACCESS CONTROL ON DATABASE privilege.
  - The grantor is the object owner.
    - 〈database privilege〉: _SYSTEM account
    - 〈tablespace privilege〉: _SYSTEM account
    - 〈schema privilege〉: _SYSTEM account
    - 〈table privilege〉: The owner of the table
    - 〈sequence privilege〉: The owner of the sequence
    - 〈procedure privilege〉: The owner of the procedure/function
    - 〈package privilege〉: The owner of the package

〈database privilege〉

It is the privilege for the database objects.
[ON DATABASE] statement can be omitted.

The database action which can be defined with the database privilege is as follows.

- ALL [PRIVILEGES] [ON DATABASE]
  - All privileges for the database which the grantor (the user who performs the statement) owns by using WITH GRANT OPTION.

Table 19-29 Database privilege

〈database action〉	Description
ADMINISTRATION	Privilege for starting or terminating the server
ALTER DATABASE	Privilege for executing ALTER DATABASE
ALTER SYSTEM	Privilege for executing ALTER SYSTEM
AUDIT SYSTEM	Privilege for controlling the audit policy
ACCESS CONTROL	Privilege for controlling all the privileges
CREATE SESSION	Privilege for connecting to the database
CREATE PROFILE	Privilege for creating profiles in the database
ALTER PROFILE	Privilege for altering any profile in the database
DROP PROFILE	Privilege for dropping any profile in the database
CREATE USER	Privilege for creating users in the database
ALTER USER	Privilege for altering any user in the database
DROP USER	Privilege for dropping any user in the database
CREATE ROLE	Privilege for creating roles in the database
ALTER ROLE	Privilege for altering any role in the database
DROP ROLE	Privilege for dropping any role in the database
CREATE TABLESPACE	Privilege for creating tablespaces in the database
ALTER TABLESPACE	Privilege for altering any tablespace in the database
DROP TABLESPACE	Privilege for dropping any tablespace in the database
USAGE TABLESPACE	Privilege for using any tablespace in the database
CREATE SCHEMA	Privilege for creating schemas in the database
ALTER SCHEMA	Privilege for altering any schema in the database
DROP SCHEMA	Privilege for dropping any schema in the database
CREATE PUBLIC SYNONYM	Privilege for creating public synonyms in the database
DROP PUBLIC SYNONYM	Privilege for dropping any public synonym in the database
CREATE ANY TABLE	Privilege for creating tables in any schema of the database
ALTER ANY TABLE	Privilege for altering any table in the database
DROP ANY TABLE	Privilege for dropping any table in the database
SELECT ANY TABLE	Privilege for querying rows of any table in the database
INSERT ANY TABLE	Privilege for creating rows of any table in the database
DELETE ANY TABLE	Privilege for deleting rows of any table in the database
UPDATE ANY TABLE	Privilege for updating rows of any table in the database
LOCK ANY TABLE	Privilege for locking any table in the database
CREATE ANY VIEW	Privilege for creating views in any schema of the database

〈database action〉	Description
DROP ANY VIEW	Privilege for dropping any view in the database
CREATE ANY SEQUENCE	Privilege for creating sequences in any schema of the database
ALTER ANY SEQUENCE	Privilege for altering any sequence of the database
DROP ANY SEQUENCE	Privilege for dropping any sequence of the database
USAGE ANY SEQUENCE	Privilege for using any sequence of the database
CREATE ANY INDEX	Privilege for creating indexes in any schema of the database
ALTER ANY INDEX	Privilege for altering any index in the database
DROP ANY INDEX	Privilege for dropping any index in the database
CREATE ANY SYNONYM	Privilege for creating synonyms in the database
DROP ANY SYNONYM	Privilege for dropping any synonym in the database
CREATE ANY PROCEDURE	Privilege for creating any procedure/function in any schema of the database
ALTER ANY PROCEDURE	Privilege for altering any procedure/function in the database
DROP ANY PROCEDURE	Privilege for dropping any procedure/function in the database
EXECUTE ANY PROCEDURE	Privilege for executing any procedure/function in the database
CREATE ANY PACKAGE	Privilege for creating any package in any schema of the database
ALTER ANY PACKAGE	Privilege for altering any package in the database
DROP ANY PACKAGE	Privilege for dropping any package in the database
EXECUTE ANY PACKAGE	Privilege for executing any package in the database
PURGE DBA_RECYCLEBIN	Privilege for dropping any package in the database

〈tablespace privilege〉

It is the privilege for the tablespace objects.

The tablespace action which can be defined with the tablespace privilege is as follows.

- ALL [PRIVILEGES] ON TABLESPACE tablespace_name
  - All privileges for the tablespace which the grantor (the user who performs the statement) owns by using WITH GRANT OPTION.

Table 19-30 Tablespace privilege

〈tablespace action〉	Description
CREATE OBJECT	Privilege for creating objects in the tablespace

〈schema privilege〉

It is the privilege for the schema objects.

- Whether to omit [ON SCHEMA schema_name]

- If multiple <grantee> exist, [ON SCHEMA schema_name] statement can not be omitted.
- If ALL [PRIVILEGES] is used, [ON SCHEMA schema_name] statement can not be omitted.
- If [ON SCHEMA schema_name] statement is omitted, <grantee> can use only one user_identifier, and the privilege on the first schema in the schema search path of <grantee> is granted.

The schema action which can be defined with the schema privilege is as follows.

- ALL [PRIVILEGES] ON SCHEMA schema_name
  - All privileges for the schema which the grantor (the user who performs the statement) owns by u sing WITH GRANT OPTION

Table 19-31 Schema privilege

<schema action>	Description
CONTROL SCHEMA	All privileges for that schema
CREATE TABLE	Privilege for creating tables in the schema
ALTER TABLE	Privilege for altering any table in the schema
DROP TABLE	Privilege for dropping any table in the schema
SELECT TABLE	Privilege for querying rows of any table in the schema
INSERT TABLE	Privilege for creating rows of any table in the schema
DELETE TABLE	Privilege for deleting rows of any table in the schema
UPDATE TABLE	Privilege for updating rows of any table in the schema
LOCK TABLE	Privilege for locking any table of the schema
CREATE VIEW	Privilege for creating views in the schema
DROP VIEW	Privilege for dropping any view of the schema
CREATE SEQUENCE	Privilege for creating sequences in the schema
ALTER SEQUENCE	Privilege for altering any sequence of the schema
DROP SEQUENCE	Privilege for dropping any sequence of the schema
USAGE SEQUENCE	Privilege for using any sequence of the schema
CREATE INDEX	Privilege for creating indexes in the schema
ALTER INDEX	Privilege for altering any index in the schema
DROP INDEX	Privilege for dropping any index in the schema
ADD CONSTRAINT	Privilege for creating constraints in the schema
CREATE SYNONYM	Privilege for creating synonyms in the schema
DROP SYNONYM	Privilege for dropping any synonym of the schema
CREATE PROCEDURE	Privilege for creating any procedure/function in the schema
ALTER PROCEDURE	Privilege for altering any procedure/function in the schema
DROP PROCEDURE	Privilege for dropping any procedure/function in the schema
EXECUTE PROCEDURE	Privilege for executing any procedure/function in the schema
CREATE PACKAGE	Privilege for creating any package in the schema
ALTER PACKAGE	Privilege for altering any package in the schema
DROP PACKAGE	Privilege for dropping any package in the schema
EXECUTE PACKAGE	Privilege for executing any package in the schema

〈table privilege〉

It is the privilege for the table object or the view object.
[TABLE] statement can be omitted.

The table action which can be defined with the table privilege is as follows.

- ALL [PRIVILEGES] ON [TABLE] table_name
  - All privileges for the table which the grantor (the user who performs the statement) owns by usin g WITH GRANT OPTION

Table 19-32 Table privilege

〈table action〉	Description
CONTROL TABLE	All privileges for that table
SELECT	Privilege for querying rows of the table
INSERT	Privilege for creating rows into the table
UPDATE	Privilege for updating rows in the table
DELETE	Privilege for deleting rows from the table
REFERENCES	Privilege for creating referential constraints which refers to the table
LOCK	Privilege for locking the table
INDEX	Privilege for creating indexes in the table
ALTER	Privilege for altering the table

For SELECT, INSERT, UPDATE, REFERENCES, additional privileges are granted to all columns of the table.

The column action which can be defined with the table privilege is as follows. However, the column actio n is applicable only to the base table.

Table 19-33 Column privilege

〈column action〉	Description
SELECT (columns)	Privilege for querying that columns
INSERT (columns)	Privilege for creating rows including that columns
UPDATE (columns)	Privilege for updating that columns
REFERENCES (columns)	Privilege for creating referential constraints which refers to that columns

〈sequence privilege〉

It is the privilege for the sequence object.

The sequence action which can be defined with the sequence privilege is as follows.

- ALL [PRIVILEGES] ON SEQUENCE sequence_name

- All privileges for the sequence which the grantor (the user who performs the statement) owns by using WITH GRANT OPTION.

Table 19-34 Sequence privilege

〈sequence action〉	Description
USAGE	Privilege for using the sequence

〈procedure privilege〉

It is the privilege for the procedure/ function object.

The action which can be defined with the procedure privilege is as follows.

- ALL [PRIVILEGES] ON PROCEDURE procedure_name
  - All privileges for the procedure/ function which the grantor (the user who performs the statemen t) owns by using WITH GRANT OPTION

Table 19-35 Procedure privilege

〈procedure action〉	Description
EXECUTE	Privilege for executing the procedure/function

〈package privilege〉

It is the privilege for the package object.

The action which can be defined with the package privilege is as follows.

- ALL [PRIVILEGES] ON PACKAGE package_name
  - All privileges for the package which is given to the grantor (the user who performs the statement) by using WITH GRANT OPTION

Table 19-36 Package privilege

〈package action〉	Description
EXECUTE	Privilege for executing the package

Description

Data Definition Language (DDL) such as GRANT privilege can be rolled back if it is before when the trans action is committed.

The owner who created SQL schema object, such as table, sequence, has certain privileges without being granted any separate privilege for the object.
For more information, refer to the following CREATE statements.

- CREATE TABLE
- CREATE VIEW
- CREATE SEQUENCE
- ALTER TABLE name ADD COLUMN
- CREATE FUNCTION
- CREATE PROCEDURE
- CREATE PACKAGE

The owner who created non-schema object such as schema, tablespace, does not automatically have any privilege for the object. Therefore, the privilege should be separately granted.
For more information, refer to the following CREATE statements.

- CREATE SCHEMA
- CREATE TABLESPACE
- CREATE USER

Examples

The following is an example of granting SELECT ON TABLE t1 privilege to the user u1.

```
gSQL> GRANT SELECT ON t1 TO u1;
Grant succeeded.
```

The following is an example of granting SELECT ON TABLE t1 privilege to the PUBLIC account (all users).

```
gSQL> GRANT SELECT ON t1 TO PUBLIC;
Grant succeeded.
```

The following is an example that the user u1 grants the privilege to the other user by using WITH GRANT OPTION.

```
gSQL> GRANT SELECT ON t1 TO u1 WITH GRANT OPTION;
Grant succeeded.
```

The following is the example that the user executing the statement grants all privileges on the TABLE t1 t o user u1 by using WITH GRANT OPTION.

```
gSQL> GRANT ALL PRIVILEGES ON TABLE t1 TO u1;
Grant succeeded.
```

The following is an example of granting CREATE SESSION ON DATABASE privilege which is a privilege for connecting to the database.

```
gSQL> GRANT CREATE SESSION ON DATABASE TO u1;
Grant succeeded.
```

The following is an example of granting multiple privileges for creating objects such as the table, view, in dex, sequence, constraint in the SCHEMA s1 to the user u1.

```
gSQL> GRANT CREATE TABLE, CREATE VIEW, CREATE INDEX, CREATE SEQUENCE, ADD CONSTRAINT ON SCHEMA
s1 TO u1;
Grant succeeded.
```

The following is an example of granting the privileges for creating objects in TABLESPACE mem_data_tb to the user u1.

```
gSQL> GRANT CREATE OBJECT ON TABLESPACE mem_data_tbs TO u1;
Grant succeeded.
```

The following is an example of granting the privilege for querying some columns in the TABLE t1 to the u ser u1.

```
gSQL> GRANT SELECT( id, name ) ON TABLE t1 TO u1;
Grant succeeded.
```

The following is an example of granting the privilege to the user u1 for using NEXTVAL(), CURRVAL() fun ctions in the SEQUENCE seq1.

```
gSQL> GRANT USAGE ON SEQUENCE seq1 TO u1;
Grant succeeded.
```

Compatibility

The SQL standard does not define the following privileges.

- 〈database privilege〉
- 〈tablespace privilege〉
- 〈schema privilege〉

Table 19-37 SQL standard compatibility

Feature ID	Description	Compatibility
S023	Basic structured types	X
S024	Enhanced structured types	X
S081	Subtables	X
T211	Basic trigger capability	X
T281	SELECT privilege with column granularity	O
T332	Extended Roles	X
F731	INSERT column privileges	O

For More Information

Refer to the followings.

- **REVOKE privileges FROM**
- **CREATE USER**
- **DROP USER**
- **ALTER USER**

20.

SQL References (H~Z)

20.1 INSERT INTO

Function

It creates new rows in a table.

Syntax

```
<insert statement> ::=
    INSERT INTO table_name [ ( column_name [, ...] ) ]
        <insert source>
    ;
<insert source> ::=
      <values clause>
    ¦ <from subquery>
    ¦ <from default>
<values clause> ::=
    VALUES { ( { <value expression> ¦ DEFAULT } [, ...] ) } [, ...]
<from subquery> ::=
    <query expression>
<from default> ::=
    DEFAULT VALUES
```

Invocation and Access Rules

A user should satisfy the following conditions to perform <Insert statement>.

- One of the following privileges is required to perform the INSERT statement.
  - INSERT(columns) ON TABLE for all columns which are targets of insert
  - (INSERT or CONTROL TABLE) ON TABLE for the table
  - (INSERT TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - INSERT ANY TABLE ON DATABASE

- One of the following privileges is required for all tables used in <from subquery>.
  - SELECT(columns) ON TABLE for all columns of tables which were used in the statement
  - (SELECT or CONTROL TABLE) ON TABLE for the table

- (SELECT TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- SELECT ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the name of a target table in which the row is to be created.
It can define schema to which the table belongs such as schema_name.table_name and if schema_name is omitted, the default schema name of the user performing the statement is used.

[(column_name [, ...])]

It is the column name of a table.
The column list can be omitted.
The number of columns and the number of 〈insert source〉 values should be same, and DEFAULT value is assigned to the omitted column.

〈values clause〉

It is the list of values to be assigned to the corresponding columns.

- 〈value expression〉
  - It is the value or expression to be assigned to the corresponding column.
- DEFAULT
  - The value of corresponding column will use the default value which were defined through **CREATE TABLE**.
  - If it is not defined, NULL will be assigned.

Multiple rows can be created as follows.

```
INSERT INTO table_name VALUES ( 1, 'A' ), ( 2, 'B' ), ( 3, 'C' )
```

〈from subquery〉

It is the query to create rows.
For more information, refer to **query expression** clause of **SELECT** statement.

DEFAULT VALUES

It fills every column with default value.

DEFAULT VALUES clause means as same as the following.

```
VALUES ( DEFAULT, DEFAULT, ..., DEFAULT )
```

Description

Differences among INSERT-related Statements

- **INSERT INTO**
  - It creates one or multiple rows into the table.
  - e.g. INSERT INTO t1 SELECT * FROM t1;
- **INSERT INTO name RETURNING**
  - It creates one or multiple rows into the table, then the created rows can be retrieved in the same way as SELECT statement(API such as SQLFetch()).
  - e.g. INSERT INTO t1 SELECT * FROM t1 RETURNING c1;
- **INSERT INTO name RETURNING .. INTO**
  - It creates one or less row, and if a single row is created, it obtains the value to the host variable o f RETURNING INTO clause.
  - e.g. INSERT INTO t1 DEFAULT VALUES RETURNING c1 INTO :v1;

Examples

The following is an example of creating a single row by using INSERT statement.

```
gSQL> INSERT INTO region VALUES ( 0, 'AFRICA' );
1 row created.
```

The following is an example of using the DEFAULT value or identity value of the column in INSERT statem ent.

```
gSQL> CREATE TABLE region
(
    r_regionkey   BIGINT    GENERATED BY DEFAULT AS IDENTITY
  , r_name        CHAR(25)  DEFAULT 'N/A'
);
```

```
Table created.
gSQL> COMMIT;
Commit complete.
```

- DEFAULT is inserted into all columns.

```
gSQL> INSERT INTO region DEFAULT VALUES;
1 row created.
```

- DEFAULT is inserted into all columns.

```
gSQL> INSERT INTO region VALUES (DEFAULT, DEFAULT);
1 row created.
```

- If a column is omitted, the DEFAULT value of r_name column is used.

```
gSQL> INSERT INTO region(r_regionkey) VALUES (-100);
1 row created.
```

- If a column is omitted, the identity value of r_regionkey column is used.

```
gSQL> INSERT INTO region(r_name) VALUES ('ASIA');
1 row created.
gSQL> SELECT * FROM region;
R_REGIONKEY R_NAME
----------- -------------------------
          1 N/A
          2 N/A
       -100 N/A
          3 ASIA
4 rows selected.
```

The following is an example of creating multiple rows by describing them in VALUES clause.

```
gSQL> INSERT INTO region
       VALUES ( 1, 'AFRICA' ),
              ( 2, 'ASIA'   ),
              ( 3, 'EUROPE' );
3 rows created.
```

The following is an example of creating multiple rows by using a subquery.

```
gSQL> INSERT INTO region SELECT r_regionkey, r_name FROM tmp_region WHERE r_regionkey < 3;
3 rows created.
```

Compatibility

Table 20-1 SQL standard compatibility

Feature ID	Description	Compatibility
F781	Self-referencing operations	X
F222	INSERT statement: DEFAULT VALUES clause	O
S204	Enhanced structured types	X
S043	Enhanced reference types	X
T111	Updatable joins, unions, and columns	X

For More Information

Refer to the followings.

- **SELECT**
- **INSERT INTO name RETURNING**
- **INSERT INTO name RETURNING .. INTO**

20.2 INSERT INTO name RETURNING

Function

It creates new rows in the table, and retrieves them.

Syntax

```
<insert statement> ::=
    INSERT INTO table_name [ ( column_name [, ...] ) ]
        <insert source>
        <returning clause>
    ;
<insert source> ::=
      <values clause>
    | <from subquery>
    | <from default>
<values clause> ::=
    VALUES { ( { <value expression> | DEFAULT } [, ...] ) } [, ...]
<from subquery> ::=
    <query expression>
<from default> ::=
    DEFAULT VALUES
<returning clause> ::=
      [ RETURN | RETURNING ] { * | { <value expression> [ [AS] alias_name ] } [, ...]
```

Invocation and Access Rules

A user should satisfy the following conditions to perform <insert returning query statement>.

- One of the following privileges is required to perform INSERT statement.
  - INSERT(columns) ON TABLE for all columns which are targets of insert
  - (INSERT or CONTROL TABLE) ON TABLE for the table
  - (INSERT TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - INSERT ANY TABLE ON DATABASE

- One of the following privileges is required for all tables used in <from subquery>.
  - SELECT(columns) ON TABLE for all columns of tables which were used in the statement
  - (SELECT or CONTROL TABLE) ON TABLE for the table
  - (SELECT TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - SELECT ANY TABLE ON DATABASE

- One of the following privileges is required for all columns used in RETURNING clause.
  - SELECT(columns) ON TABLE for all columns which were used in RETURNING clause.
  - (SELECT or CONTROL TABLE) ON TABLE for the table
  - (SELECT TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - SELECT ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the name of a target table in which the row is to be created.

[(column_name [, ...])]

It is the column name of a table.
For more information, refer to **INSERT INTO**.

<values clause>

It is the list of values to be assigned to the corresponding columns.
For more information, refer to **INSERT INTO**.

<from subquery>

It is the query to create rows.
For more information, refer to **INSERT INTO**.

DEFAULT VALUES

It fills every column with default value.
For more information, refer to **INSERT INTO**.

〈returning clause〉

It returns the inserted rows.

- It sets the inserted rows as a result set, and specifies the rows to be retrieved.
  - RETURNING clause returns the rows which were inserted by INSERT statement, and which is a res ult set.
  - 〈value expression〉
    - It is as same as 〈select list〉 in SELECT statement, but it can not use the aggregation.
  - [[AS] alias_name]
    - It can name a value expression by using AS clause.

The keywords RETURNING and RETURN have the same meaning.

Description

For more information, refer to **Differences among INSERT-related Statements**.

Examples

The following is an example of retrieving the column values created by using INSERT statement.

```
gSQL> CREATE TABLE region
(
    r_regionkey   BIGINT    GENERATED BY DEFAULT AS IDENTITY
  , r_name        CHAR(25)  DEFAULT 'N/A'
);
Table created.
gSQL> COMMIT;
Commit complete.
```

- The following is an example of returning the created DEFAULT value (RETURNING).

```
gSQL> INSERT INTO region VALUES ( DEFAULT, DEFAULT ) RETURNING r_regionkey, r_name;
R_REGIONKEY R_NAME
----------- -------------------------
          1 N/A
1 row created.
```

- The following is an example of returning the omitted column value (RETURNING).

```
gSQL> INSERT INTO region(r_name) VALUES ('ASIA') RETURNING r_regionkey;
R_REGIONKEY
-----------
          2
1 row created.
```

The following is an example of retrieving the rows created by using the subquery.

```
gSQL> INSERT INTO region
      SELECT r_regionkey, r_name FROM tmp_region WHERE r_regionkey < 3
      RETURNING r_regionkey, r_name;
R_REGIONKEY R_NAME
----------- -------------------------
          0 AFRICA
          1 AMERICA
          2 ASIA
3 rows created.
```

Compatibility

The SQL standard does not define <insert returning query statement>.

For More Information

Refer to the followings.

- **INSERT INTO**
- **INSERT INTO name RETURNING .. INTO**

20.3 INSERT INTO name RETURNING .. INTO

Function

It creates a single row in a table, and obtains the value of the created row into a host variable.

Syntax

```
<insert statement> ::=
    INSERT INTO table_name [ ( column_name [, ...] ) ]
        <insert source>
        <returning into clause>
    ;
<insert source> ::=
      <values clause>
    | <from subquery>
    | <from default>
<values clause> ::=
    VALUES { ( { <value expression> | DEFAULT } [, ...] ) } [, ...]
<from subquery> ::=
    <query expression>
<from default> ::=
    DEFAULT VALUES
<returning into clause> ::=
      [ RETURN | RETURNING ] { * | { <value expression> [ [AS] alias_name ] } [, ...] INTO
variable_name [, ...]
```

Invocation and Access Rules

A user should satisfy the following conditions to perform <insert returning into statement>.

- One of the following privileges is required to perform INSERT statement.
  - INSERT(columns) ON TABLE for all columns which are targets of insert.
  - (INSERT or CONTROL TABLE) ON TABLE for the table
  - (INSERT TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs

    - INSERT ANY TABLE ON DATABASE

- One of the following privileges is required for all tables used in 〈from subquery〉.
    - SELECT(columns) ON TABLE for all columns of tables which were used in the statement
    - (SELECT or CONTROL TABLE) ON TABLE for the table
    - (SELECT TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
    - SELECT ANY TABLE ON DATABASE

- One of the following privileges is required for all columns used in RETURNING clause.
    - SELECT(columns) ON TABLE for all columns which were used in RETURNING clause
    - (SELECT or CONTROL TABLE) ON TABLE for the table
    - (SELECT TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
    - SELECT ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the name of a target table in which the row is to be created.

[(column_name [, ...])]

It is the column name of a table.
For more information, refer to **INSERT INTO**.

〈values clause〉

It is the list of values to be assigned to the corresponding columns.
For more information, refer to **INSERT INTO**.

〈from subquery〉

It is the query to create rows.
For more information, refer to **INSERT INTO**.

DEFAULT VALUES

It fills every column with default value.
For more information, refer to **INSERT INTO**.

〈returning clause〉

It returns the inserted rows.
For more information, refer to **〈returning clause〉** in **INSERT INTO name RETURNING** statement.

INTO variable_name [, ...]

The number of variables in INTO clause should be equal to the number of the expressions in RETURNING clause.
The row to be created should be one or less. If two or more rows are created, an error occurs.

Description

For more information, refer to **Differences among INSERT-related Statements**.

Example

The following is an example of obtaining the value of the created row into a host variable.

```
gSQL> CREATE TABLE region
(
    r_regionkey   BIGINT    GENERATED BY DEFAULT AS IDENTITY
  , r_name        CHAR(25)  DEFAULT 'N/A'
);
Table created.
gSQL> COMMIT;
Commit complete.
```

- The host variables are declared.

```
\VAR v_key  BIGINT
\VAR v_name VARCHAR(128)
```

- The created DEFAULT values are obtained into the host variables.

```
gSQL> INSERT INTO region
      VALUES ( DEFAULT, DEFAULT )
      RETURNING r_regionkey, r_name
      INTO :v_key, :v_name;
V_KEY V_NAME
----- -------------------------
    1 N/A
1 row created.
```

- The omitted column value is obtained into the host variable.

```
gSQL> INSERT INTO region(r_name)
      VALUES ('ASIA')
      RETURNING r_regionkey
      INTO :v_key;
V_KEY
-----
    2
1 row created.
```

Compatibility

The SQL standard does not define <insert returning into statement>.

For More Information

Refer to the followings.

- **INSERT INTO**
- **INSERT INTO name RETURNING**

20.4 INSERT INTO name ... UPDATE

Function

It creates new rows in a table. If it violates the unique constraint, then it updates the existing rows.

Syntax

```
<upsert statement> ::=
    INSERT INTO table_name [ ( column_name [, ...] ) ]
        <insert source>
        <duplicate key clause>
    ;
<insert source> ::=
      <values clause>
    | <from subquery>
    | DEFAULT VALUES
<values clause> ::=
    VALUES { ( { <value expression> | DEFAULT } [, ...] ) } [, ...]
<from subquery> ::=
    <query expression>
<duplicate key clause>
    ON DUPLICATE KEY { DO NOTHING | <do update clause> }
<do update clause> ::=
    [DO] UPDATE [SET] <set clause>  [, ...]
<set value clause> ::=
      <value expression>
    | DEFAULT
    | VALUES( column_name )
<set clause> ::=
      column_name = <set value clause>
    | ( column_name [, ...] ) = ( <set value clause> [, ...] )
    | ( column_name [, ...] ) = ( <query expression> )
```

Invocation and Access Rules

A user should satisfy the following conditions to perform <upsert statement>.

- One of the following privileges is required to perform the corresponding statement.
  - INSERT(columns) ON TABLE for all columns which are targets of insert
  - (INSERT or CONTROL TABLE) ON TABLE for the table
  - (INSERT TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - INSERT ANY TABLE ON DATABASE
  - UPDATE(columns) ON TABLE for all columns which are targets of update
  - (UPDATE or CONTROL TABLE) ON TABLE for the table
  - (UPDATE TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - UPDATE ANY TABLE ON DATABASE

- One of the following privileges is required for all tables used in <from subquery>.
  - SELECT(columns) ON TABLE for all columns of tables which were used in the statement
  - (SELECT or CONTROL TABLE) ON TABLE for the table
  - (SELECT TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - SELECT ANY TABLE ON DATABASE

- One of the following privileges is required for all columns used in RETURNING clause.
  - SELECT(columns) ON TABLE for all columns which were used in RETURNING clause
  - (SELECT or CONTROL TABLE) ON TABLE for the table
  - (SELECT TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - SELECT ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the name of a target table in which the row is to be created.
Or, it is the name of a target table to be updated when they are updated because it violates the unique c onstraint.
It can define schema to which the table belongs such as schema_name.table_name and if schema_name is omitted, the default schema name of the user performing the statement is used.

[(column_name [, ...])]

It is the column name of a table.
For more information, refer to **[(column_name [, ...])]** clause of **INSERT INTO** statement.

〈values clause〉

It is the list of values to be assigned to the corresponding columns.
For more information, refer to **〈values clause〉** of **INSERT INTO** statement.

〈from subquery〉

It is the query to create rows.
For more information, refer to **query expression** clause of **SELECT** statement.

DEFAULT VALUES

It fills every column with default value.
For more information, refer to **DEFAULT VALUES** clause of **INSERT INTO** statement.

〈duplicate key clause〉

It defines the action to perform when it violates the unique constraint.

DO NOTHING

It does not perform any operation when it violates the unique constraint.

〈do update clause〉

It updates the values in columns according to 〈set clause〉 when it violates the unique constraint.

〈set value clause〉

It defines the values to assign to the columns to be updated.

It can be defined as follows.

- column_name = 〈value expression〉

```
DO UPDATE SET column1 = value1, column2 = value2, column3 = value3
```

- column_name = DEFAULT

```
DO UPDATE SET column1 = DEFAULT, column2 = DEFAULT, column3 = DEFAULT
```

- column_name = VALUES(column_name)

<insert source> value is used to update the value.

```
DO UPDATE SET column1 = VALUES(column1), column2 = VALUES(column2), column3 = VALUES(column2)
```

<set clause>

It defines the columns to be updated and the values to be assigned, and the number of columns in <set clause> and the number of values should be same.

It can be defined as follows.

- column_name = { <set value clause> }

```
ON DUPLICATE KEY
   DO UPDATE SET column1 = value1, column2 = value2, column3 = value3
```

- (column_name [, ...]) = (<set value clause> } [, ...])

```
ON DUPLICATE KEY
   DO UPDATE SET ( column1, column2, column3 ) = ( value1, value2, value3 )
```

- (column_name [, ...]) = (<query expression>)

```
ON DUPLICATE KEY
   DO UPDATE SET column1 = ( SELECT max(value1) FROM other_table_name )
```

<query expression> should be a query creating a single row.

If DEFAULT is used as a column value, it uses the default value (refer to **<default clause>**.) defined when performing **CREATE TABLE**, and NULL value is assigned when it is not defined.

Description

Differences among INSERT INTO name ... UPDATE-related Statements

- **INSERT INTO name ... UPDATE**
  - It creates rows in a table. If it violates the unique constraint, then it updates the existing rows.
  - e.g. INSERT INTO t1 VALUES (1, 1) ON DUPLICATE KEY UPDATE c2 = c2 + 1;
- **INSERT INTO name ... UPDATE RETURNING**
  - It creates rows in a table or updates the existing rows. The inserted rows or the updated rows ca n be retrieved in the same way as SELECT statement (API such as SQLFetch()).
  - e.g. INSERT INTO t1 VALUES (1, 1) ON DUPLICATE KEY UPDATE c2 = c2 + 1 RETURNING c2;
- **INSERT INTO name ... UPDATE RETURNING ... INTO**
  - It creates or updates one or less row. If the created row or the updated row is a single row, then i t obtains the value into a host variable of RETURNING INTO clause.
  - e.g. INSERT INTO t1 VALUES (1, 1) ON DUPLICATE KEY UPDATE c2 = c2 + 1 RETURNING c2 INT O :v1;

〈upsert statement〉 is a deterministic statement.

The results of the two equivalent and different UPSERT statements should be same as follows.

- INSERT INTO t1 VALUES(1),(2),(3) ON DUPLICATE KEY UPDATE c1 = c1 + 1;
- INSERT INTO t1 VALUES(3),(2),(1) ON DUPLICATE KEY UPDATE c1 = c1 + 1;

```
gSQL> CREATE TABLE t1 ( c1 INTEGER UNIQUE );
Table created.
gSQL> INSERT INTO t1 VALUES( 1 ),( 2 ),( 3 );
3 rows created.
gSQL> INSERT INTO t1 VALUES( 1 ),( 2 ),( 3 ) ON DUPLICATE KEY UPDATE c1 = c1 + 1;
3 rows created.
gSQL> SELECT * FROM t1;
C1
--
 2
 3
 4
3 rows selected.
```

```
gSQL> CREATE TABLE t1 ( c1 INTEGER UNIQUE );
Table created.
gSQL> INSERT INTO t1 VALUES( 1 ),( 2 ),( 3 );
```

```
3 rows created.
gSQL> INSERT INTO t1 VALUES( 3 ),( 2 ),( 1 ) ON DUPLICATE KEY UPDATE c1 = c1 + 1;
3 rows created.
gSQL> SELECT * FROM t1;
C1
--
 2
 3
 4
3 rows selected.
```

Examples

The following is an example of updating a single row because it violates the unique constraint.

```
gSQL> CREATE TABLE t1 ( c1 INTEGER UNIQUE );
Table created.
gSQL> INSERT INTO t1 VALUES( 1 );
1 row created.
gSQL> INSERT INTO t1 VALUES( 1 ) ON DUPLICATE KEY UPDATE c1 = c1 + 1;
1 row created.
gSQL> SELECT * FROM t1;
C1
--
 2
1 row selected.
```

The following is an example of not updating a row even when it violates the unique constraint.

```
gSQL> CREATE TABLE t1 ( c1 INTEGER UNIQUE );
Table created.
gSQL> INSERT INTO t1 VALUES( 1 );
1 row created.
gSQL> INSERT INTO t1 VALUES( 1 ) ON DUPLICATE KEY DO NOTHING;
no rows created.
gSQL> SELECT * FROM t1;
C1
--
 1
1 row selected.
```

The following is an example of inserting or updating multiple rows by using a subquery.

```
gSQL> CREATE TABLE t1 ( c1 INTEGER UNIQUE );
Table created.
gSQL> INSERT INTO t1 VALUES( 1 ),( 2 ),( 3 ),( 4 );
4 rows created.
gSQL> INSERT INTO t1 ( SELECT c1 FROM t1 ) ON DUPLICATE KEY UPDATE c1 = c1 + 1;
4 rows created.
gSQL> SELECT * FROM t1;
C1
--
 2
 3
 4
 5
4 rows selected.
```

Compatibility

The SQL standard does not define <upsert statement>.

For More Information

Refer to the followings.

- INSERT INTO
- INSERT INTO name ... UPDATE RETURNING
- INSERT INTO name ... UPDATE RETURNING ... INTO
- UPDATE

20.5 INSERT INTO name ... UPDATE RETURNING

Function

It creates new rows in a table. If it violates the unique constraint, then it updates the existing rows. Then, it retrieves the created rows or the updated rows.

Syntax

```
<upsert returning statement> ::=
    INSERT INTO table_name [ ( column_name [, ...] ) ]
        <insert source>
        <duplicate key clause>
        <returning clause>
    ;
<insert source> ::=
      <values clause>
    | <from subquery>
    | DEFAULT VALUES
<values clause> ::=
    VALUES { ( { <value expression> | DEFAULT } [, ...] ) } [, ...]
<from subquery> ::=
    <query expression>
<duplicate key clause>
    ON DUPLICATE KEY { DO NOTHING | <do update clause> }
<do update clause> ::=
    [DO] UPDATE [SET] <set clause>  [, ...]
<set value clause> ::=
      <value expression>
    | DEFAULT
    | VALUES( column_name )
<set clause> ::=
      column_name = <set value clause>
    | ( column_name [, ...] ) = ( <set value clause> [, ...] )
    | ( column_name [, ...] ) = ( <query expression> )
<returning clause> ::=
    [ RETURN | RETURNING ] { * | { <value expression> [ [AS] alias_name ] } [, ...]
```

Invocation and Access Rules

A user should satisfy the following conditions to perform <upsert returning statement>.

- One of the following privileges is required to perform the corresponding statement.
  - INSERT(columns) ON TABLE for all columns which are targets of insert
  - (INSERT or CONTROL TABLE) ON TABLE for the table
  - (INSERT TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - INSERT ANY TABLE ON DATABASE
  - UPDATE(columns) ON TABLE for all columns which are targets of update
  - (UPDATE or CONTROL TABLE) ON TABLE for the table
  - (UPDATE TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - UPDATE ANY TABLE ON DATABASE

- One of the following privileges is required for all tables used in <from subquery>.
  - SELECT(columns) ON TABLE for all columns of tables which were used in the statement
  - (SELECT or CONTROL TABLE) ON TABLE for the table
  - (SELECT TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - SELECT ANY TABLE ON DATABASE

- One of the following privileges is required for all columns used in RETURNING clause.
  - SELECT(columns) ON TABLE for all columns which were used in RETURNING clause
  - (SELECT or CONTROL TABLE) ON TABLE for the table
  - (SELECT TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - SELECT ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the name of a target table in which the row is to be created.
Or, it is the name of a target table to be updated when they are updated because it violates the unique c onstraint.
For more information, refer to **table_name** clause of **INSERT INTO name ... UPDATE** statement.

[(column_name [, ...])]

It is the column name of a table.
For more information, refer to **[(column_name [, ...])]** clause of **INSERT INTO** statement.

〈values clause〉

It is the list of values to be assigned to the corresponding columns.
For more information, refer to **〈values clause〉** of **INSERT INTO** statement.

〈from subquery〉

It is the query to create rows.
For more information, refer to **query expression** clause of **SELECT** statement.

DEFAULT VALUES

It fills every column with default value.
For more information, refer to **DEFAULT VALUES** clause of **INSERT INTO** statement.

〈duplicate key clause〉

It defines the action to perform when it violates the unique constraint.

DO NOTHING

It does not perform any operation when it violates the unique constraint.

〈do update clause〉

It updates the values in columns according to 〈set clause〉 when it violates the unique constraint.

〈set value clause〉

It defines the values to assign to the columns to be updated.
For more information, refer to **〈set value clause〉** of **INSERT INTO name ... UPDATE** statement.

〈set clause〉

It defines the columns to be updated and the values to be assigned, and the number of columns in 〈set clause〉 and the number of values should be same.
For more information, refer to **〈set clause〉** of **INSERT INTO name ... UPDATE**.

〈returning clause〉

It returns the inserted rows or the updated rows.

- It sets the created rows as a result set, and specifies the rows to be retrieved.
  - RETURNING clause returns the result which sets inserted rows or updated rows as a result set.
  - 〈value expression〉
    - It is as same as 〈select list〉 in SELECT statement, but it can not use the aggregation.
  - [[AS] alias_name]
    - It can name a value expression by using AS clause.

Description

For more information, refer to **Differences among INSERT INTO name ... UPDATE-related Statements**.

The following is an example of inserting four rows, and returning the inserted results.

```
gSQL> CREATE TABLE t1 ( c1 INTEGER UNIQUE );
Table created.
gSQL> INSERT INTO t1 VALUES( 1 ), ( 2 ), ( 3 ), ( 4 ) ON DUPLICATE KEY UPDATE c1 = c1 + 1
RETURNING c1;
C1
--
 1
 2
 3
 4
4 rows created.
```

The following is an example of updating rows because it violates the unique constraint, and returning the updated results.

```
gSQL> CREATE TABLE t1 ( c1 INTEGER UNIQUE );
Table created.
```

```
gSQL> INSERT INTO t1 VALUES( 1 ),( 2 ),( 3 ),( 4 );
4 rows created.
gSQL> INSERT INTO t1 ( SELECT c1 FROM t1 ) ON DUPLICATE KEY UPDATE c1 = c1 + 1 RETURNING c1;
C1
--
 2
 3
 4
 5
4 rows created.
```

Compatibility

The SQL standard does not define <upsert returning statement>.

For More Information

Refer to the followings.

- **INSERT INTO name ... UPDATE**
- **INSERT INTO name ... UPDATE RETURNING ... INTO**

20.6 INSERT INTO name ... UPDATE RETURNING ... INTO

Function

It creates a single row in a table. If it violates the unique constraint, then it updates the existing rows. Then, it obtains the created rows or the updated rows as the host variable.

Syntax

```
<upsert returning into statement> ::=
    INSERT INTO table_name [ ( column_name [, ...] ) ]
        <insert source>
        <duplicate key clause>
        <returning clause>
        <into clause>
    ;
<insert source> ::=
      <values clause>
    ¦ <from subquery>
    ¦ DEFAULT VALUES
<values clause> ::=
    VALUES { ( { <value expression> ¦ DEFAULT } [, ...] ) } [, ...]
<from subquery> ::=
    <query expression>
<duplicate key clause>
    ON DUPLICATE KEY { DO NOTHING ¦ <do update clause> }
<do update clause> ::=
    [DO] UPDATE [SET] <set clause>  [, ...]
<set value clause> ::=
      <value expression>
    ¦ DEFAULT
    ¦ VALUES( column_name )
<set clause> ::=
      column_name = <set value clause>
    ¦ ( column_name [, ...] ) = ( <set value clause> [, ...] )
```

```
    ¦ ( column_name [, ...] ) = ( <query expression> )
<returning clause> ::=
    [ RETURN ¦ RETURNING ] { * ¦ { <value expression> [ [AS] alias_name ] } [, ...]
<into clause> ::= INTO variable_name [, ...]
```

Invocation and Access Rules

A user should satisfy the following conditions to perform <upsert returning into statement>.

- One of the following privileges is required to perform the corresponding statement.
  - INSERT(columns) ON TABLE for all columns which are targets of insert
  - (INSERT or CONTROL TABLE) ON TABLE for the table
  - (INSERT TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - INSERT ANY TABLE ON DATABASE
  - UPDATE(columns) ON TABLE for all columns which are targets of update
  - (UPDATE or CONTROL TABLE) ON TABLE for the table
  - (UPDATE TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - UPDATE ANY TABLE ON DATABASE

- One of the following privileges is required for all tables used in <from subquery>.
  - SELECT(columns) ON TABLE for all columns of tables which were used in the statement
  - (SELECT or CONTROL TABLE) ON TABLE for the table
  - (SELECT TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - SELECT ANY TABLE ON DATABASE

- One of the following privileges is required for all columns used in RETURNING clause.
  - SELECT(columns) ON TABLE for all columns which were used in RETURNING clause
  - (SELECT or CONTROL TABLE) ON TABLE for the table
  - (SELECT TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - SELECT ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the name of a target table in which the row is to be created.
Or, it is the name of a target table to be updated when they are updated because it violates the unique c onstraint.

For more information, refer to **table_name** clause of **INSERT INTO name ... UPDATE** statement.

[(column_name [, ...])]

It is the column name of a table.
For more information, refer to **[(column_name [, ...])]** clause of **INSERT INTO** statement.

〈values clause〉

It is the list of values to be assigned to the corresponding columns.
For more information, refer to **〈values clause〉** of **INSERT INTO** statement.

〈from subquery〉

It is the query to create rows.
For more information, refer to **query expression** clause of **SELECT** statement.

DEFAULT VALUES

It fills every column with default value.
For more information, refer to **DEFAULT VALUES** clause of **INSERT INTO** statement.

〈duplicate key clause〉

It defines the action to perform when it violates the unique constraint.

DO NOTHING

It does not perform any operation when it violates the unique constraint.

〈do update clause〉

It updates the values in columns according to 〈set clause〉 when it violates the unique constraint.

〈set value clause〉

It defines the values to assign to the columns to be updated.
For more information, refer to **〈set value clause〉** of **INSERT INTO name ... UPDATE** statement.

〈set clause〉

It defines the columns to be updated and the values to be assigned, and the number of columns in 〈set clause〉 and the number of values should be same.
For more information, refer to **〈set clause〉** of **INSERT INTO name ... UPDATE**.

〈returning clause〉

It returns the inserted rows or the updated rows.
For more information, refer to **〈returning clause〉** of **INSERT INTO name ... UPDATE RETURNING**.

〈into clause〉

The number of variables specified in INTO clause should be same as the number of expressions specified in RETURNING clause.
The row should be created one or less. If two or more rows are created, then an error occurs.

Description

For more information, refer to **Differences among INSERT INTO name ... UPDATE-related Statements**.

The following is an example of inserting a single row, then obtaining the inserted result as the host variable.

```
gSQL> \VAR v_c1 INTEGER;
gSQL> CREATE TABLE t1 ( c1 INTEGER UNIQUE );
Table created.
gSQL> INSERT INTO t1 VALUES( 1 ) ON DUPLICATE KEY UPDATE c1 = c1 + 1 RETURNING c1 INTO :v_c1;
V_C1
----
   1
1 row created.
```

The following is an example of updating a single row because it violates the unique constraint, then obtaining the updated result as the host variable.

```
gSQL> \VAR v_c1 INTEGER;
gSQL> CREATE TABLE t1 ( c1 INTEGER UNIQUE );
Table created.
gSQL> INSERT INTO t1 VALUES( 1 );
1 row created.
gSQL> INSERT INTO t1 VALUES( 1 ) ON DUPLICATE KEY UPDATE c1 = c1 + 1 RETURNING c1 INTO :v_c1;
V_C1
----
   2
1 row created.
```

Compatibility

The SQL standard does not define <upsert returning into statement>.

For More Information

Refer to the followings.

- **INSERT INTO name ... UPDATE**
- **INSERT INTO name ... UPDATE RETURNING**

20.7 LOCK TABLE

Function

It locks one or more tables.

Syntax

```
<lock table statement> ::=
    LOCK TABLE lock target [, ...]
    IN <lock mode> MODE [<wait clause>]
    ;
<lock mode> ::=
    SHARE
    ¦ EXCLUSIVE
    ¦ ROW SHARE
    ¦ ROW EXCLUSIVE
    ¦ SHARE ROW EXCLUSIVE
<wait clause> ::=
    NOWAIT
    ¦ WAIT time
```

Invocation and Access Rules

One of the following privileges is required to perform <lock table statement>.

- (LOCK or CONTROL TABLE) ON TABLE for the table
- (LOCK TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- LOCK ANY TABLE ON DATABASE

Syntax Rules and Parameters

〈lock target〉

It specifies the target table to be locked.

〈lock mode〉

It specifies the LOCK mode.

- SHARE
  - It allows concurrent queries for the locked table, but prohibits updating the table.
- EXCLUSIVE
  - It allows exclusive queries for the locked table.
- ROW SHARE
  - It allows concurrent access to the locked table, but prohibits locking the entire table for exclusive access.
- ROW EXCLUSIVE
  - It allows concurrent access to the locked table, but prohibits locking the entire table for exclusive access.
  - If ROW EXCLUSIVE mode is set, it prohibits locking in SHARE mode.
  - ROW EXCLUSIVE mode is automatically obtained when updating, inserting, deleting.
- SHARE ROW EXCLUSIVE
  - It is used to search for the entire table or to make other users search for the rows in the table.
  - It prohibits other users from accessing the locked tables in SHARE mode or accessing the rows be ing updated.

〈wait clause〉

It specifies the waiting time to acquire the lock.

- NOWAIT
  - It immediately acquires the lock contol for the object.
  - If the lock is already set by another user, the control is immediately handed over.
    - In this case, the database generates a message.
- WAIT time
  - It sets the waiting time for acquiring the lock.
  - It is specified in seconds, and its value is from 0 to 1,000,000,000.
- If it is not specified, it waits indefinitely until acquiring the lock.

Description

If the transaction is committed or rolled back all acquired locks are automatically released. When using R OLLBACK TO SAVEPOINT statement, all locks acquired since that savepoint are released.

Examples

The following is an example of locking the TABLE t1 to prevent any updating operation by another transa ction.

```
gSQL> LOCK TABLE t1 IN EXCLUSIVE MODE;
Table locked.
```

The following is an example of performing LOCK statement for multiple tables.

```
gSQL> LOCK TABLE t1, t2 IN EXCLUSIVE MODE;
Table locked.
```

The following is an example of acquiring SHARE ROW EXCLUSIVE lock for the TABLE t1.

```
gSQL> LOCK TABLE t1 IN SHARE ROW EXCLUSIVE MODE;
Table locked.
```

The following statement is performed only when the lock can be immediately acquired for the table. If th e lock can not be acquired, an error occurs.

```
gSQL> LOCK TABLE t1 IN EXCLUSIVE MODE NOWAIT;
Table locked.
```

The following is an example of waiting 10 seconds to acquire the lock.

```
gSQL> LOCK TABLE t1 IN EXCLUSIVE MODE WAIT 10;
Table locked.
```

Compatibility

The SQL standard does not cover the concepts of the lock table.

For More Information

Refer to the followings.

- COMMIT
- ROLLBACK

20.8 NOAUDIT POLICY

Function

It deactivates the audit policy.

Syntax

```
<noaudit policy statement> ::=
    NOAUDIT POLICY policy_name
    [ <specified_user_option> ]
    ;
<specified_user_option> ::=
      BY user_name [, ...]
```

Invocation and Access Rules

AUDIT SYSTEM ON DATABASE privilege is required to perform <noaudit policy statement>.

Syntax Rules and Parameters

policy_name

It is the name of the audit policy object to be deactivated.
The deactivated audit policy does not effect on the existing session, and it effects only on the newly creat ed session.

<specified_user_option>

It specifies the user to be excluded from the auditing target.

Unlike AUDIT POLICY statement, NOAUDIT POLICY does not have EXCEPT option.

If AUDIT POLICY name BY clause is used, NOAUDIT POLICY name BY statement should be used to deacti vate it.
If AUDIT POLICY name EXCEPT clause is used, NOAUDIT POLICY name statement without BY clause shou ld be used to deactivate it.

NOAUDIT POLICY statement should be used as follows according to the usage of AUDIT POLICY stateme nt to deactivate it.

Table 20-2 Activating/ deactivating audit policy

Type	AUDIT POLICY statement	NOAUDIT POLICY statement
All users	AUDIT POLICY p1	NOAUDIT POLICY p1
Using BY	AUDIT POLICY p1 BY u1	NOAUDIT POLICY p1 BY u1
Using EXCEPT	AUDIT POLICY p1 EXCEPT u1	NOAUDIT POLICY p1

When deactivating all activated users, the audit policy object is completely deactivated.

Description

The information about activationthe audit policy object is retrieved as follows.

```
SELECT policy_name
     , enabled_opt
     , user_name
  FROM audit_policy_enabled
 WHERE policy_name = 'P1';
```

NOAUDIT POLICY statement deletes each created information about activationaccording to the AUDIT P OLICY specifying method.
If the information activated through the query above does not exist, then the audit policy is completely d eactivated.

If all users are activated as follows, NOAUDIT POLICY BY clause does not does not affect it.

```
AUDIT POLICY p1;
```

- It does not does not affect any.

```
NOAUDIT POLICY p1 BY u1;
```

- It should be deactivated as follows.

```
NOAUDIT POLICY p1;
```

If one or more users are separately activated, use NOAUDIT POLICY statement according to the AUDIT PO LICY specifying method.

When Activated by Using BY

If the audit policy is activated as follows,

```
AUDIT POLICY p1 WHENEVER NOT SUCCESSFUL;
AUDIT POLICY p1 BY u1;
AUDIT POLICY p1 BY u2;
```

the information about activation is as follows.

```
SELECT policy_name
     , enabled_opt
     , user_name
     , when_success
     , when_failure
  FROM audit_policy_enabled
 WHERE policy_name = 'P1';
POLICY_NAME  ENABLED_OPT  USER_NAME    WHEN_SUCCESS  WHEN_FAILURE
-----------  -----------  ---------    ------------  ------------
P1           BY           ALL USERS    NO            YES
P1           BY           U1           YES           YES
P1           BY           U2           YES           YES
```

The following is an example of performing NOAUDIT statement and the information about activation.

```
NOAUDIT POLICY p1;
SELECT policy_name
     , enabled_opt
     , user_name
     , when_success
     , when_failure
  FROM audit_policy_enabled
 WHERE policy_name = 'P1';
POLICY_NAME  ENABLED_OPT  USER_NAME    WHEN_SUCCESS    WHEN_FAILURE
-----------  -----------  ---------    ------------    ------------
P1           BY           U1           YES             YES
P1           BY           U2           YES             YES
```

The auditing for a failure for ALL USERS is deactivated, but the auditing for user u1, u2 is still activated.

If NOAUDIT POLICY statement is additionally used through BY option as follows, then audit policy p1 is c ompletely deactivated.

```
NOAUDIT POLICY p1 BY u1, u2;
SELECT policy_name
     , enabled_opt
     , user_name
     , when_success
     , when_failure
  FROM audit_policy_enabled
 WHERE policy_name = 'P1';
no rows selected.
```

When Activated by Using EXCEPT

If the audit policy is activated as follows,

```
AUDIT POLICY p1 EXCEPT u1, sys;
```

the information about activation is as follows.

```
SELECT policy_name
     , enabled_opt
     , user_name
     , when_success
     , when_failure
  FROM audit_policy_enabled
 WHERE policy_name = 'P1';
POLICY_NAME  ENABLED_OPT  USER_NAME    WHEN_SUCCESS    WHEN_FAILURE
-----------  -----------  ---------    ------------    ------------
P1           EXCEPT       U1           YES             YES
P1           EXCEPT       SYS          YES             YES
```

Unlike AUDIT POLICY statement, NOAUDIT POLICY does not have EXCEPT option, so execute the statem ent without an option as follows.

```
NOAUDIT POLICY p1;
SELECT policy_name
     , enabled_opt
     , user_name
```

```
     , when_success
     , when_failure
  FROM audit_policy_enabled
 WHERE policy_name = 'P1';
no rows selected.
```

In other words, if the audit policy is activated by using EXCEPT option, each user can not be activated aga in by using NOAUDIT POLICY statement.

Examples

The following is an example of deactivating all users.

```
NOAUDIT POLICY table_pol;
```

The following is an example of deactivating a specific activated user by using BY.

```
NOAUDIT POLICY table_pol BY u1;
```

Compatibility

The SQL standard does not have the audit policy.

For More Information

Refer to the followings.

- Managing audit policy object
  - **CREATE AUDIT POLICY**
  - **DROP AUDIT POLICY**
  - **ALTER AUDIT POLICY**
- Activating/ deactivating audit policy
  - **AUDIT POLICY**
  - **NOAUDIT POLICY**
- Viewing audit trail: **AUDIT_TRAIL**

- Clearing audit trail: **ALTER DATABASE CLEAR AUDIT TRAIL**

20.9 OPEN cursor_name

Function

It opens a cursor.

Syntax

```
<open statement> ::=
    OPEN cursor_name [ <parameter using clause> ]
    ;
<parameter using clause> ::=
      <using parameter arguments>
<using parameter arguments> ::=
    USING variable_name [, ...]
```

Invocation and Access Rules

If cursor_name is a dynamic cursor which is declared by using **PREPARE statement_name** and **DECLARE c ursor_name**, it can be used in an embedded SQL.

It is same with the privilege of **<cursor query>** included in **DECLARE cursor_name** which declared cursor_n ame.

Syntax Rules and Parameters

cursor_name

It should be a cursor declared with **DECLARE cursor_name** within the session.

〈parameter using clause〉

It can be used in an embedded SQL.

When 〈parameter using clause〉 is used, cursor_name should be a dynamic cursor declared by using **PREP ARE statement_name** and **DECLARE cursor_name**.

〈using parameter arguments〉

When 〈using parameter arguments〉 is used, the number of variable_name should be equal to the numb er of the parameter included in a query which is referenced by **PREPARE statement_name**.

The listed variable_name corresponds to the dynamic parameter in an order of its description.

```
{
    ...
    EXEC SQL PREPARE stmt1 FROM 'SELECT c1, c2 FROM t1 WHERE c1 IN ( ?, ?, ? )';
    EXEC SQL DECLARE cur1 CURSOR FOR stmt1;
    EXEC SQL OPEN cur1 USING :sValue1, :sValue2, :sValue3;
    ...
    EXEC SQL WHENEVER NOT FOUND DO break;
    for(;;)
    {
        EXEC SQL FETCH cur1 INTO :sC1, :sC2;
    }
    EXEC SQL WHENEVER NOT FOUND CONTINUE;
    ...
    EXEC SQL CLOSE cur1;
    ...
}
```

Description

The cursor is a distinguishable object in a session. The cursor being used in the current session has nothin g to do with the cursor being used in another session.

To use OPEN cursor_name statement, it should be a cursor declared with **DECLARE cursor_name**, and it s hould be a closed cursor.

Examples

The following is an example of declaring a cursor and using OPEN cursor statement in an interactive SQL (gsql).

```
gSQL> DECLARE cur1 CURSOR FOR SELECT id, data FROM t1;
Cursor declared.
gSQL> OPEN cur1;
Cursor is open.
gSQL> \var v_id   INTEGER
gSQL> \var v_data VARCHAR(128)
gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   1 data_1
1 row fetched.
gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   2 data_2
1 row fetched.
gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   3 data_3
1 row fetched.
gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   4 data_4
1 row fetched.
gSQL> FETCH cur1 INTO :v_id, :v_data;
V_ID V_DATA
---- ------
   5 data_5
1 row fetched.
gSQL> FETCH cur1 INTO :v_id, :v_data;
no rows fetched.
gSQL> CLOSE cur1;
Cursor closed.
```

Compatibility

Table 20-3 SQL standard compatibility

Feature ID	Description	Compatibility
B031	Basic Dynamic SQL	O

For More Information

Refer to the followings.

- **DECLARE cursor_name**
- **FETCH cursor_name**
- **CLOSE cursor_name**
- **PREPARE statement_name**

20.10 PREPARE statement_name

Function

It prepares a dynamic SQL statement for a repeated execution.

Syntax

```
<prepare statement> ::=
    PREPARE statement_name FROM <SQL statement variable>
    ;
<SQL statement variable> ::=
      variable_name
    ¦ 'sql statement'
    ¦ "sql statement"
    ¦ sql statement
```

Invocation and Access Rules

It can be used in an embedded SQL.
An appropriate privilege according to the type of a dynamic SQL statement is required.

Syntax Rules and Parameters

statement_name

It is the name of the statement to be prepared.
The length of the statement name should be shorter than 128 bytes.
EXECUTE statement_name and **DECLARE cursor_name**, which are to be performed later, refers to the sta tement_name.
If the same statement_name exists, the previously prepared dynamic SQL is dropped.

```
{
    ...
    EXEC SQL PREPARE stmt1 FROM 'DELETE FROM t1';
    ...
    EXEC SQL PREPARE stmt1 FROM 'UPDATE t1 SET c1 = c1 + 10';
    ...
}
```

〈SQL statement variable〉

〈SQL statement variable〉 can be used as following four types.

- variable_name: It is a variable in which an SQL statement is stored.
- 'sql statement': It is an SQL statement which is enclosed with single quote (').
- "sql statement": It is an SQL statement which is enclosed with double quotes (").
- sql statement: It is an SQL statement without quote.

The single quote (') is used twice as follows to represent string data within single-quoted string.

```
{
    ...
    PREPARE stmt_name FROM 'INSERT INTO t1 VALUES ( ''literal data'' )';
    ...
}
```

The dynamic SQL statement referenced by 〈SQL statement variable〉 can use a host variable (:var) or para meter marker (?).
However, if the unquoted SQL statement is used, the parameter marker (?) can not be used.

Depending on the characteristics of the referenced dynamic SQL statements, the variable can be either in put or output dynamic parameter.
The dynamic parameter described in the dynamic SQL statement does not have a meaning for the variabl e name, and it is identified by the specified order regardless of its type.

- Example 1

```
{
    ...
    int sValue1;
    int sValue2;
    ...
    EXEC SQL PREPARE stmt1 FROM 'DELETE FROM t1 WHERE c1 BETWEEN ? AND ?';
```

```
    EXEC SQL EXECUTE stmt1 USING :sValue1, :sValue2;
    ...
}
```

- All parameter markers are the input dynamic parameters.
- The order to identify
  - No. 1 - BETWEEN ?
    - Input dynamic parameter
    - It uses the value of :sValue1.
  - No. 2 - AND ?
    - Input dynamic parameter
    - It uses the value of :sValue2.

- Example 2

```
{
    ...
    int sValue1;
    int sValue2;
    ...
    EXEC SQL PREPARE stmt1 FROM 'SELECT SUM(c2) INTO :v1 FROM t1 WHERE c1 > :v2';
    EXEC SQL EXECUTE stmt1 USING :sValue1, :sValue2;
    ...
}
```

- The input dynamic parameter and the output dynamic parameter exist.
- The order to identify
  - No. 1 - :v1
    - Output dynamic parameter
    - It stores the value in :sValue1.
  - No. 2 - :v2
    - Input dynamic parameter
    - It uses the value of :sValue2.

variable_name

The type corresponding to variable_name should be a character string.
The dynamic SQL statement defined in variable_name should be valid.

sql statement

The dynamic SQL statement defined in the sql statement should be valid.

Description

PREPARE statement_name FROM sql_string statement analyzes SQL statement to use EXECUTE or cursor statement. Statement_name is an identifier which informs the precompiler the statement in an embedde d SQL source code. A separate type or declaration is not required because statement_name is not a host variable.

For more information, refer to **Embedded Dynamic SQL**.

Example

The following is an example of using PREPARE statement_name in an embedded SQL source code.

```
{
    ...
    sprintf( sUpdateSql, "UPDATE EMP SET sal = sal * :v1 WHERE JOB = 'SALES'");
    EXEC SQL PREPARE UPDATE_STMT FROM :sUpdateSql;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    sRatio = 1.1;
    EXEC SQL EXECUTE UPDATE_STMT USING :sRatio;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    ...
}
```

The full source code in which PREPARE statement_name was used can be viewed in **Dynamic Embedded SQL Example Program**.

Compatibility

Table 20-4 SQL standard compatibility

Feature ID	Description	Compatibility
B031	Basic Dynamic SQL	O
B034	Dynamic specification of cursor attributes	X

For More Information

Refer to the followings.

- **EXECUTE statement_name**
- **DECLARE cursor_name**
- **EXECUTE IMMEDIATE 'sql_string'**
- **Embedded Dynamic SQL**

20.11 PURGE

Function

It permanently drops objects stored in the recycle bin.

Syntax

```
<purge statement> :==
    PURGE <purge action>
    ;
<purge action> :==
     TABLE table_name
   ¦ INDEX index_name
   ¦ CONSTRAINT constraint_name
   ¦ TABLESPACE tablespace_name [ USER user_name ]
   ¦ RECYCLEBIN
   ¦ USER_RECYCLEBIN
   ¦ DBA_RECYCLEBIN
```

Invocation and Access Rules

One of the following privilege is required for a user to perform <purge statement>.

- The owner of that table
- CONTROL TABLE ON TABLE for that table
- (DROP TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- DROP ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the name of the object stored or of the dropped table in the recycle bin.
It can define the schema to which the table belongs in the dropped table name, such as schema_name.table_name. If schema_name is omitted, the default schema name of the user performing the statement is used. In this case, indexes and constraints which are related to the table are also dropped.

index_name

It is the name of the object stored or of the dropped index in the recycle bin.
It can define the schema to which the index belongs such as schema_name.index_name.
If schema_name is omitted, the default schema name of the user performing the statement is used.
The key index which is created with a constraint should be dropped with the constraint.

constraint_name

It is the name of the object stored or of the dropped constraint in the recycle bin.

tablespace_name

It is the name of the tablespace.
When assigning USER, DROP ANY TABLE ON DATABASE privilege is required.

user_name

It is the name of the user.

recyclebin

It is the alias of user_recyclebin.

user_recyclebin

It drops all recycle bins owned by a user.

dba_recyclebin

It drops all recycle bins in the database.
PURGE DBA_RECYCLEBIN ON DATABASE privilege is required.

Description

It permanently drops objects stored in the recycle bin by using the object name or the dropped table nam e stored in the recycle bin. If the name which is as same as that of the dropped table exists, then the olde st table object is dropped.

When specifying a tablespace in the recycle bin object owned by a user, then only the objects included in the tablespace are dropped. In this case, if a user is assigned, then only the objects included in the specifi ed tablespace owned by the user are dropped.

PURGE TABLE, INDEX, CONSTRAINT statements can be rolled back if it is before when the transaction is committed. However, PURGE TABLESPACE, RECYCLEBIN, DBA_RECYCLEBIN statements can not be rolle d back, and the transaction which performed the statement is automatically committed.

Example

The following is an example of dropping a table stored in the recycle bin.

```
gSQL> SELECT OBJECT_NAME, ORIGINAL_NAME, OBJECT_TYPE FROM USER_RECYCLEBIN;
OBJECT_NAME                          ORIGINAL_NAME        OBJECT_TYPE
------------------------------------ -------------------- -----------
BIN$135B9908166111EA9C5C835D3E4BBBF7 T1                   TABLE
BIN$135B993A166111EA9C5C835D3E4BBBF7 T1_PRIMARY_KEY       CONSTRAINT
BIN$135B991C166111EA9C5C835D3E4BBBF7 T1_PRIMARY_KEY_INDEX INDEX
BIN$135B9926166111EA9C5C835D3E4BBBF7 T1_IDX1              INDEX
4 rows selected.
gSQL> PURGE TABLE t1;
Table purged.
```

The following is an example of dropping an index stored in the recycle bin.

```
gSQL> SELECT OBJECT_NAME, ORIGINAL_NAME, OBJECT_TYPE FROM USER_RECYCLEBIN;
OBJECT_NAME                          ORIGINAL_NAME        OBJECT_TYPE
------------------------------------ -------------------- -----------
```

```
BIN$135B9908166111EA9C5C835D3E4BBBF7 T1                   TABLE
BIN$135B993A166111EA9C5C835D3E4BBBF7 T1_PRIMARY_KEY       CONSTRAINT
BIN$135B991C166111EA9C5C835D3E4BBBF7 T1_PRIMARY_KEY_INDEX INDEX
BIN$135B9926166111EA9C5C835D3E4BBBF7 T1_IDX1              INDEX
4 rows selected.
gSQL> PURGE INDEX t1_idx1;
Index purged.
```

The following is an example of dropping constraints stored in the recycle bin.

```
gSQL> SELECT OBJECT_NAME, ORIGINAL_NAME, OBJECT_TYPE FROM USER_RECYCLEBIN;
OBJECT_NAME                          ORIGINAL_NAME        OBJECT_TYPE
------------------------------------ -------------------- -----------
BIN$135B9908166111EA9C5C835D3E4BBBF7 T1                   TABLE
BIN$135B993A166111EA9C5C835D3E4BBBF7 T1_PRIMARY_KEY       CONSTRAINT
BIN$135B991C166111EA9C5C835D3E4BBBF7 T1_PRIMARY_KEY_INDEX INDEX
3 rows selected.
gSQL> PURGE CONSTRAINT t1_primary_key;
Constraints purged.
```

The following is an example of dropping objects included in the tablespace stored in the recycle bin.

```
gSQL> SELECT OBJECT_NAME, ORIGINAL_NAME, OBJECT_TYPE, TABLESPACE_NAME FROM USER_RECYCLEBIN;
OBJECT_NAME                          ORIGINAL_NAME OBJECT_TYPE TABLESPACE_NAME
------------------------------------ ------------- ----------- ---------------
BIN$02C76B24166311EA9C5C835D3E4BBBF7 T1            TABLE       MEM_DATA_TBS
1 row selected.
gSQL> PURGE TABLESPACE MEM_DATA_TBS;
Tablespace purged.
```

The following is an example of dropping all recycle bins owned by a user.

```
gSQL> SELECT OBJECT_NAME, ORIGINAL_NAME, OBJECT_TYPE FROM USER_RECYCLEBIN;
OBJECT_NAME                          ORIGINAL_NAME        OBJECT_TYPE
------------------------------------ -------------------- -----------
BIN$64F6BFFC166311EA9C5C835D3E4BBBF7 T1                   TABLE
BIN$64F6C042166311EA9C5C835D3E4BBBF7 T1_PRIMARY_KEY       CONSTRAINT
BIN$64F6C010166311EA9C5C835D3E4BBBF7 T1_PRIMARY_KEY_INDEX INDEX
BIN$64F6C024166311EA9C5C835D3E4BBBF7 T1_IDX1              INDEX
4 rows selected.
gSQL> PURGE USER_RECYCLEBIN;
Recyclebin purged.
```

The following is an example of dropping all recycle bins in the system.

```
gSQL> SELECT OWNER, OBJECT_NAME, ORIGINAL_NAME, OBJECT_TYPE FROM USER_RECYCLEBIN;
OWNER OBJECT_NAME                          ORIGINAL_NAME        OBJECT_TYPE
----- ------------------------------------ -------------------- -----------
TEST  BIN$F0FB26F0166311EAA7C5D51B86D72AB6 T1                   TABLE
TEST  BIN$F0FB272C166311EAA7C5D51B86D72AB6 T1_PRIMARY_KEY       CONSTRAINT
TEST  BIN$F0FB2704166311EAA7C5D51B86D72AB6 T1_PRIMARY_KEY_INDEX INDEX
TEST  BIN$F0FB2718166311EAA7C5D51B86D72AB6 T1_IDX1              INDEX
4 rows selected.
gSQL> PURGE DBA_RECYCLEBIN;
DBA Recyclebin purged.
```

Compatibility

The SQL standard does not define <purge statement>.

For More Information

Refer to the followings.

- **Managing Recycle Bin of Table**
- **FLASHBACK TABLE**

20.12 RELEASE SAVEPOINT savepoint_specifier

Function

It releases a savepoint.

Syntax

```
<release savepoint statement> ::=
    RELEASE SAVEPOINT savepoint_name
    ;
```

Syntax Rules and Parameters

savepoint_name

It is a name of the savepoint, and it should exist.
The length of the savepoint name should be shorter than 128 bytes.

Description

If multiple savepoints are defined and RELEASE SAVEPOINT savepoint_name statement is performed, all s avepoints defined since the savepoint_name are also released.

Example

The following is an example of releasing a savepoint.

```
gSQL> RELEASE SAVEPOINT sp2;
Savepoint dropped.
```

Compatibility

Table 20-5 SQL standard compatibility

Feature ID	Description	Compatibility
T271	Savepoints	O

For More Information

Refer to the followings.

- **COMMIT**
- **ROLLBACK**
- **SAVEPOINT savepoint_specifier**

20.13 REVOKE privileges FROM

Function

It revokes the granted privilege from a user.

Syntax

```
<revoke privilege statement> ::=
    REVOKE [ <revoke option extention> ] <privilege>
      FROM <grantee> [, ...]
      [ <revoke behavior> ]
    ;
<revoke option extention> ::=
      GRANT OPTION FOR
<revoke behavior> ::=
      RESTRICT
    ¦ CASCADE
    ¦ CASCADE CONSTRAINTS
```

Syntax Rules and Parameters

<privilege>

It is a privilege which is to be revoked from the revokee (the user whose privilege is to be revoked).

The revoker (the user who performs the statement) should satisfy one of the following conditions.

- If it is <privilege> which the revoker grants to the revokee.
  - Only the <privilege> which the revoker grants to the revokee is revoked.
- If the revoker owns ACCESS CONTROL ON DATABASE privilege.
  - The <privilege> which other grantors grant to the revokee is revoked.

When using ALL [PRIVILEGES], it succeeds even when the satisfying <privilege> does not exist.

For more information about the types of <privilege>, refer to **<privilege>** clause of **GRANT privileges TO** st atement.

<grantee>

It is a user whose privilege is to be revoked.

- user_identifier
  - It revokes the privilege of that user.
- PUBLIC
  - They are authorization objects which mean all users.

GRANT OPTION FOR

It revokes WITH GRANT OPTION included in the privilege.
It also revokes WITH GRANT OPTION of the dependent privilege.

The privilege is maintained.

<revoke behavior>

- Dependent privilege: It is as same as the <privilege> which was granted to the revokee by using WITH GRANT OPTION and granted to another user by the revokee.
- RESTRICT
  - If the dependent privilege exists, it can not be revoked.
- CASCADE
  - The dependent privilege should also be revoked.
- CASCADE CONSTRAINTS
  - The dependent privilege should also be revoked.
- If it is omitted, the default value is CASCADE.

Description

Data Definition Language (DDL) such as REVOKE privilege can be rolled back if it is before when the trans action is committed.

When performing the following DROP statement, all privilege information related to the object is revoked even without performing any separate REVOKE statement.

- DROP statement related to SQL schema object

    - DROP TABLE
    - DROP VIEW
    - DROP SEQUENCE
    - ALTER TABLE name SET UNUSED COLUMN
    - DROP FUNCTION
    - DROP PROCEDURE
    - DROP PACKAGE

- DROP statement related to non-schema object
    - DROP SCHEMA
    - DROP TABLESPACE
    - DROP USER

Examples

The following is an example of revoking multiple privileges for the table t1.

```
gSQL> REVOKE INSERT, UPDATE, DELETE, LOCK, ALTER, INDEX ON t1 FROM u1;
Revoke succeeded.
```

The following is an example of revoking SELECT ON TABLE t1 privilege granted to the PUBLIC account, w hich means all users. However, only the privilege for PUBLIC account is revoked, and SELECT ON TABLE t1 privilege which was explicitly granted to a specific user is not revoked.

```
gSQL> REVOKE SELECT ON t1 FROM PUBLIC;
Revoke succeeded.
```

The following is an example that SELECT ON TABLE t1 privilege granted to user u1 is remained, and only REVOKE GRANT OPTION which can grant the privilege to another user is revoked.

```
gSQL> REVOKE GRANT OPTION FOR SELECT ON t1 FROM u1;
Revoke succeeded.
```

The following is an example that an error occurs when the privilege granted to the user u1 is revoked by using RESTRICT option and the user u1 grants it to another user. CASCADE option is used to revoke these dependent privileges as well.

```
gSQL> REVOKE SELECT ON t1 FROM u1 RESTRICT;
ERR-2B000(16235): dependent privilege descriptors still exist
gSQL> REVOKE SELECT ON t1 FROM u1 CASCADE;
Revoke succeeded.
```

Compatibility

The SQL standard does not define the following privileges.

- 〈database privilege〉
- 〈tablespace privilege〉
- 〈schema privilege〉

〈revoke behavior〉 of the SQL standard has the following differences.

- The default value of the SQL standard is RESTRICT.
- The SQL standard does not cover CASCADE CONSTRAINTS.

Table 20-6 SQL standard compatibility

Feature ID	Description	Compatibility
T311	Basic roles	X
F034	Extended REVOKE statement	X
S081	Subtables	X

For More Information

Refer to the followings.

- **GRANT privileges TO**
- **〈database privilege〉**
- **〈tablespace privilege〉**
- **〈schema privilege〉**
- **〈table privilege〉**
- **Column privilege**
- **〈sequence privilege〉**

20.14 ROLLBACK

Function

It rolls back a transaction, or the operation after the savepoint.

Syntax

```
<rollback statement> ::=
    ROLLBACK [ WORK ] [ <rollback force clause> ¦ <savepoint clause> ]
    ;
<rollback force clause> ::=
    FORCE 'xid_string' [ COMMENT 'comment_string' ]
<savepoint clause> ::=
    TO SAVEPOINT savepoint_name
```

Syntax Rules and Parameters

WORK

It is a reserved word which does not affect the operation.

<rollback force clause>

It is used to manually rollback the distributed transaction.

- FORCE 'xid_string'
  - It rolls back the distributed transaction corresponding to 'xid_string'.
  - 'xid_string' consists of 'format_id.transaction_id.branch_id'.
- COMMENT 'comment_string'
  - It specifies the comment on a transaction when rolling back the distributed transaction.

〈savepoint clause〉

It specifies the rollback scope of the current transaction.

- If it is omitted
  - It undoes all operations of the current transaction.
  - It ends the transaction.
  - It deletes all savepoints.
  - It releases all transaction locks.

- TO SAVEPOINT savepoint_name
  - It undoes the operations of the current transaction since savepoint_name.
  - It does not end the transaction.
  - It deletes all savepoints since savepoint_name.
  - It releases all transaction locks acquired since savepoint_name.

Description

ROLLBACK statement undoes the following statements performed in the transaction.

- Data Manipulation Language (DML) statement
  - The statements to update data, such as INSERT, UPDATE and DELETE
- Data Definition Language (DDL) statement
  - The statements to alter the structure and definition of the object such as CREATE, DROP, ALTER, TRUNCATE, GRANT and REVOKE

Exceptionally, the following statements of DDL which deals with OS resources or alters the DATA TYPE ca n not be rolled back, but are automatically committed when executing the statement.

- **CREATE TABLESPACE**
- **DROP TABLESPACE**
- **ALTER TABLESPACE**
- ALTER TABLE .. ALTER COLUMN .. SET DATA TYPE: **〈alter column data type clause〉**

Examples

The following is an example of rolling back the INSERT statement.

```
gSQL> INSERT INTO t1 VALUES ( 1, 'anonymous' );
1 row created.
gSQL> SELECT * FROM t1;
ID DATA
-- ---------
 1 anonymous
1 row selected.
gSQL> ROLLBACK;
Rollback complete.
gSQL> SELECT * FROM t1;
no rows selected.
```

The following is an example of ROLLBACK after performing the DROP TABLE statement.

```
gSQL> DROP TABLE t1;
Table dropped.
gSQL> SELECT * FROM t1;
ERR-42000(16040): table or view does not exist :
SELECT * FROM t1
              *
ERROR at line 1:
gSQL> ROLLBACK;
Rollback complete.
gSQL> SELECT * FROM t1;
ID DATA
-- ---------
 1 anonymous
1 row selected.
```

Compatibility

Table 20-7 SQL standard compatibility

Feature ID	Description	Compatibility
T271	Savepoints	O
T261	Chained transactions	X

For More Information

Refer to the followings.

- COMMIT
- SAVEPOINT savepoint_specifier

20.15 SAVEPOINT savepoint_specifier

Function

It defines a savepoint.

Syntax

```
<savepoint statement> ::=
    SAVEPOINT savepoint_name
    ;
```

Syntax Rules and Parameters

savepoint_name

It is a name of the savepoint.
If the savepoint name is as same as the existing savepoint name, then the existing savepoint is deleted.
The length of the savepoint name should be shorter than 128 bytes.

Description

The defined savepoint is used by ROLLBACK TO SAVEPOINT statement (refer to **ROLLBACK**.), and DML or DDL statement which has been performed up to the savepoint is rolled back. Then the locks acquired by using that statement are released, too.

The defined savepoint is automatically deleted when the transaction is committed or rolled back, or it can be explicitly deleted by using **RELEASE SAVEPOINT savepoint_specifier**.

Example

The following is an example of defining the savepoint and using ROLLBACK TO SAVEPOINT statement.

```
gSQL> SAVEPOINT sp1;
Savepoint created.
gSQL> INSERT INTO t1 VALUES ( 1, 'anonymous' );
1 row created.
gSQL> SAVEPOINT sp2;
Savepoint created.
gSQL> INSERT INTO t1 VALUES ( 2, 'someone' );
1 row created.
gSQL> SAVEPOINT sp3;
Savepoint created.
gSQL> INSERT INTO t1 VALUES ( 3, 'anyone' );
1 row created.
gSQL> SELECT * FROM t1;
ID DATA
-- ---------
 1 anonymous
 2 someone
 3 anyone
3 rows selected.
gSQL> ROLLBACK TO SAVEPOINT sp3;
Rollback complete.
gSQL> SELECT * FROM t1;
ID DATA
-- ---------
 1 anonymous
 2 someone
2 rows selected.
gSQL> ROLLBACK TO SAVEPOINT sp2;
Rollback complete.
gSQL> SELECT * FROM t1;
ID DATA
-- ---------
 1 anonymous
1 row selected.
gSQL> ROLLBACK TO SAVEPOINT sp1;
Rollback complete.
```

```
gSQL> SELECT * FROM t1;
no rows selected.
```

Compatibility

Table 20-8 SQL standard compatibility

Feature ID	Description	Compatibility
T271	Savepoints	O

For More Information

Refer to the followings.

- **COMMIT**
- **ROLLBACK**
- **RELEASE SAVEPOINT savepoint_specifier**

20.16 SELECT

query expression

Function

It retrieves desired rows from one or more tables or views.

Syntax

```
<query expression> ::=
    [ <with clause> ] <query expression body> [ <order by clause> ] [ <offset limit clause> ]
<query expression body> ::=
      <query term>
    ¦ <set operator>
<query term> ::=
      <query specification>
    ¦ <left paren> <query expression body> [ <order by clause> ] [ <offset limit clause> ]
<right paren>
```

Invocation and Access Rules

One of the following privileges for all tables used in the statement is required for a user to perform <query expression>.

- SELECT(columns) ON TABLE for all used columns of table in the statement
- (SELECT or CONTROL TABLE) ON TABLE for the table
- (SELECT TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- SELECT ANY TABLE ON DATABASE

Syntax Rules and Parameters

<with clause>

<with clause> defines the temporary result set, and it refers to that result set.
For more information, refer to **with clause**.

〈set operator〉

It performs a set operation among the subqueries.
For more information, refer to **set operator**.

〈query specification〉

It specifies a single subquery.
For more information, refer to **〈query specification〉**.

〈order by clause〉

It specifies sorting information of a query result.
For more information, refer to **order by clause**.

〈offset limit clause〉

It specifies the number of rows to skip and the number of rows to fetch from the query result set.
For more information, refer to **offset limit clause**.

Description

It specifies a query with SELECT statement.
〈with clause〉, 〈order by clause〉, 〈offset limit clause〉 can be omitted.
Two or more subqueries can be specified by using 〈set operator〉.

Examples

The following is an example of SELECT statement.

```
gSQL> SELECT s_name, s_nation FROM supplier;
S_NAME                    S_NATION
------------------------- -------------
Supplier#1                FRANCE
Supplier#2                KOREA
Supplier#3                GERMANY
Supplier#4                UNITED STATES
Supplier#5                CANADA
5 rows selected.
```

The following is an example of the SELECT statement which uses 〈order by clause〉.

```
gSQL> SELECT s_name, s_nation FROM supplier ORDER BY s_name DESC;
S_NAME                    S_NATION
------------------------- -------------
Supplier#5                CANADA
Supplier#4                UNITED STATES
Supplier#3                GERMANY
Supplier#2                KOREA
Supplier#1                FRANCE
5 rows selected.
```

The following is an example of the SELECT statement which uses <offset limit clause>.

```
gSQL> SELECT s_name, s_nation FROM supplier OFFSET 1;
S_NAME                    S_NATION
------------------------- -------------
Supplier#2                KOREA
Supplier#3                GERMANY
Supplier#4                UNITED STATES
Supplier#5                CANADA
4 rows selected.
gSQL> SELECT s_name, s_nation FROM supplier LIMIT 1;
S_NAME                    S_NATION
------------------------- --------
Supplier#1                FRANCE
1 row selected.
```

The following is an example of the SELECT statement which uses <order by clause> and <offset limit claus e>.

```
gSQL> SELECT s_name, s_nation FROM supplier ORDER BY s_name DESC OFFSET 3 LIMIT 1;
S_NAME                    S_NATION
------------------------- --------
Supplier#2                KOREA
1 row selected.
```

The following is an example of SELECT statement which uses <with clause>.

```
* Non Recursive CTE
gSQL> WITH revenue ( supplier_no, total_revenue ) AS
      (
            SELECT
                  l_suppkey,
```

```
                SUM(l_extendedprice * (1 - l_discount))
             FROM lineitem
            WHERE l_shipdate >= DATE '1996-01-01'
              AND l_shipdate < DATE '1996-01-01' + INTERVAL '3' MONTH
            GROUP BY
              l_suppkey
      )
    select
        s_suppkey,
        s_name,
        s_address,
        s_phone,
        ROUND( total_revenue, 2 ) as total_revenue
    from
        supplier,
        revenue
    where
          s_suppkey = supplier_no
      and total_revenue = (
                            select
                                max(total_revenue)
                            from
                                revenue
                           )
    order by
       s_suppkey;
S_SUPPKEY S_NAME                    S_ADDRESS         S_PHONE         TOTAL_REVENUE
--------- ------------------------- ----------------- --------------- -------------
     8449 Supplier#000008449        Wp34zim9qYFbVctdW 20-469-856-8873    1772627.21
1 row selected.
* Recursive CTE
gSQL> WITH GenerateRecord ( c1, c2 ) AS
     (
          SELECT 1, 11
            FROM dual
          UNION ALL
          SELECT c1 + 1, c2 + 1
            FROM GenerateRecord
           WHERE c1 < 10
     )
SELECT c1, c2 FROM GenerateRecord;
```

```
C1 C2
-- --
 1 11
 2 12
 3 13
 4 14
 5 15
 6 16
 7 17
 8 18
 9 19
10 20
10 rows selected.
```

Compatibility

Table 20-9 SQL standard compatibility

Feature ID	Description	Compatibility
T121	WITH(excluding RECURSIVE) in query expression	O
T122	WITH(excluding RECURSIVE) in subquery	O
T131	Recursive query	O
T132	Recursive query in subquery	O
F661	Simple tables	O
F302	INTERSECT table operator	O
F301	CORRESPONDING in query expressions	X
T551	Optional key words for default syntax	O
F304	EXCEPT ALL table operator	O
F850	Top-level <order by clause> in <query expression>	O
F851	<order by clause> in subqueries	O
F855	Nested <order by clause> in <query expression>	O
F856	Nested <fetch first clause> in <query expression>	O
F857	Top-level <fetch first clause> in <query expression>	O
F858	<fetch first clause> in subqueries	O
F860	dynamic <fetch first row count> in <fetch first clause>	X
F861	Top-level <result offset clause> in <query expression>	O
F862	<result offset clause> in subqueries	O
F863	Nested <result offset clause> in <query expression>	O
F865	dynamic <offset row count> in <result offset clause>	X
F866	FETCH FIRST clause: PERCENT option	X
F867	FETCH FIRST clause: WITH TIES option	X

with clause

Function

<with clause> defines the temporary result set, and it refers to that result set.
The result set a temporary result set with a given name, and it is defined and referred in SELECT statemen t. It is called as Common Table Expression (CTE).

Syntax

```
<with clause> ::=
    WITH <with list>
<with list> ::=
    <with list element> [ { <comma> <with list element> }... ]
<with list element> ::=
    <query name> [ <left paren> <with column list> <right paren> ]
        AS <table subquery> [ <search or cycle clause> ]
<with column list> ::=
    <column name list>

<search or cycle clause> ::=
    <search clause>
  ¦ <cycle clause>
  ¦ <search clause> <cycle clause>
<search clause> ::=
    SEARCH <recursive search order> SET <sequence column>
<recursive search order> ::=
    DEPTH FIRST BY <ordering column list>
  ¦ BREADTH FIRST BY <ordering column list>
<ordering column list> ::=
    <ordering column> [ { <comma> <ordering column> }... ]
<ordering column> ::=
    <column name> [ ASC ¦ DESC ] [ NULLS FIRST ¦ NULLS LAST ]
<sequence column> ::=
    <column name>
<cycle clause> ::=
    CYCLE <cycle column list> SET <cycle mark column> TO <cycle mark value>
        DEFAULT <non-cycle mark value>
<cycle column list> ::=
    <cycle column> [ { <comma> <cycle column> }... ]
```

```
<cycle column> ::=
    <column name>

<cycle mark column> ::=
    <column name>

<cycle mark value> ::=
    <value expression>
<non-cycle mark value> ::=
    <value expression>
```

Invocation and Access Rules

It is supported in <query expression> statement, and the user should satisfy the access privilege of <query expression> to perform it.
For more information, refer to **query expression**.

Syntax Rules and Parameters

<with list>

It can define multiple <with list element>.

<with list element>

It defines the temporary result set of specified <query name>.
<with list element> is called as Common Table Expression (CTE).
CTE is divided into a recursive CTE and a non-recursive CTE.
For more information, refer to **Description**.

- Referring to CTE
  - Referring to CTE is restricted according to the described sequence.
  - CTE described before the current CTE can be referenced.
  - CTE described after the current CTE can not be referenced.
- Non-recursive CTE
  - CTE described before the current CTE can be referenced in <with list element>.
- Recursive CTE
  - Either self CTE, or CTE described before the current CTE can be referenced in <with list element>.
  - Self-reference CTE is allowed only once within CTE.

```
--# success : non recursive CTE
WITH CTE_1( c1 ) AS
    (
         SELECT i1
           FROM t1
    ),
    CTE_2( c2 ) AS
    (
         SELECT c1
           FROM CTE_1       ❶ Referring to the leading CTE
    )
SELECT c2 FROM CTE_2;
--# error : non recursive CTE
WITH CTE_1( c1 ) AS
    (
         SELECT c2
           FROM CTE_2       ❷ Referring to the trailing CTE
    ),
    CTE_2( c2 ) AS
    (
         SELECT i1
           FROM t1
    )
SELECT c1 FROM CTE_1;
```

```
--# success : recursive CTE
WITH CTE_1( c1 ) AS
     (
          SELECT i1
            FROM t1
     ),
     CTE_2( c2 ) AS
     (
          SELECT c1
            FROM CTE_1          ❶ Referring to the leading CTE
     ),
     CTE_3( c3 ) AS
     (
          SELECT 1
            FROM CTE_1
          UNION ALL
```

```
          SELECT 1
           FROM CTE_2, CTE_3      ❷ Referring to the leading CTE or self CTE
     )
SELECT c3 FROM CTE_3;
--# error : recursive CTE
WITH CTE_RECURSIVE( c1 ) AS
   (
          SELECT i1
            FROM t1
          WHERE i1 IS NULL
          UNION ALL
          SELECT 1
            FROM CTE_RECURSIVE A, CTE_RECURSIVE B    ❸ Self-reference CTE is allowed only once
          WHERE 1 = 0
     )
SELECT c1 FROM CTE_RECURSIVE;
```

〈query name〉

〈query name〉 should not be duplicated within WITH clause.

〈with column list〉

Recursive CTE can not omit 〈with column list〉.

〈search clause〉

- Non-recursive CTE
  - It can not describe 〈search clause〉.
- Recursive CTE
  - It describes the sort order of CTE result records.

- 〈recursive search order〉
  - DEPTH FIRST BY
    - Child rows are returned before sibling rows are returned.
  - BREADTH FIRST BY
    - Sibling rows are returned before child rows are returned.

- 〈ordering column list〉
  - It specifies the ordering of column list.
  - Sort order
    - ASC

        - DESC
        - If not specified, the default value is ASC.
    - Null ordering
        - NULLS FIRST
        - NULLS LAST
        - If not specified, the default value is NULLS LAST.
    - It should describe <with column list> declared in <with list element>.
    - It does not support LONG type (LONG VARCHAR, LONG VARBINARY).

- <sequence column>
    - It stores the sequence of CTE result records.
    - <column name> should not be a duplicate of following items.
        - The column name declared in <with column list> of <with list element>
        - The column name declared in <cycle column list> of <cycle clause>

<cycle clause>

- Non-recursive CTE
    - It can not describe <cycle clause>.
- Recursive CTE
    - If <cycle clause> is omitted, it returns an error when cycle occurs.

It stores <cycle mark value> or <non-cycle mark value> in <cycle mark column> according to whether cycle occurs.

- <cycle column list>
    - It should describe <with column list> declared in <with list element>.

- <cycle mark column>
    - <column name> should not be a duplicate of following items.
        - The column name declared in <with column list> of <with list element>
        - The column name declared in <sequence column> of <search clause>

- <cycle mark value> or <non-cycle mark value>
    - It can describe 1 byte character only.

Description

<with clause> defines a temporary result set, and it can refer to that result set.
It is defined and referenced within SELECT statement, and it is a temporary result set with a given name.
It is called as Common Table Expression (CTE).
CTE is classified into recursive CTE and non-recursive CTE.

- Recursive CTE: It refers to CTE which is currently defined within CTE (self-reference CTE).

```
WITH RECURSIVE_CTE ( c1 ) AS
     (
          SELECT 1
            FROM dual
          UNION ALL
          SELECT c1 + 1
            FROM RECURSIVE_CTE
           WHERE c1 < 10
     )
SELECT c1 FROM RECURSIVE_CTE;
```

- Non-recursive CTE: It is CTE other than recursive CTE.

```
WITH NON_RECURSIVE_CTE ( c1 ) AS
     (
          SELECT i1
            FROM t1
          UNION ALL
          SELECT i1
            FROM t2
     )
SELECT c1 FROM NON_RECURSIVE_CTE;
```

〈with clause〉 can be described in SELECT, INSERT, UPDATE, DELETE, CREATE TABLE AS SELECT, CREAT E VIEW statement.

〈with list element〉

It defines the temporary result set of the described 〈query name〉.
〈with list element〉 is called as Common Table Expression (CTE).
CTE is classified into recursive CTE and non-recursive CTE.

- Recursive CTE
  - It refers to CTE which is currently defined within CTE (self-reference CTE).
  - The query block including the self-reference CTE is called as a recursive member query.
  - The query block other than a recursive member query is called as an anchor member query.
  - The recursive member query and the anchor member query should be composed of UNION ALL.
  - Only one recursive member query can be described.

- Non-recursive CTE: It is CTE other than recursive CTE.

```
WITH CTE_RECURSIVE( c1, c2 ) AS
    (
         SELECT i1, i2                          ❶ Anchor member query
           FROM t1
          WHERE i2 IS NULL
         UNION ALL
         SELECT i1, i2                          ❷ Recursive member query
           FROM CTE_RECURSIVE, t1       ❸ Self reference
          WHERE CTE_RECURSIVE.c1 = t1.i2
    )
SELECT c1, c2 FROM CTE_RECURSIVE;
```

- The recursive member query can not include the following items.
  - GROUP BY or DISTINCT clause
    - (X) SELECT i1, i2 FROM CTE_RECURSIVE, t1 WHERE CTE_RECURSIVE.c1 = t1.i2 GROUP BY i1, i2
    - (X) SELECT DISTINCT i1, i2 FROM CTE_RECURSIVE, t1 WHERE CTE_RECURSIVE.c1 = t1.i2
  - Referring to CTE in the inner part of LEFT, RIGHT, OUTER JOIN
    - (X) SELECT i1, i2 FROM t1 LEFT OUTER JOIN CTE_RECURSIVE ON t1.i2 = CTE_RECURSIVE.c1
  - Aggregation function
    - (X) SELECT MAX(i1), MAX(i2) FROM CTE_RECURSIVE, t1 WHERE CTE_RECURSIVE.c1 = t1.i2
  - Subquery including self-reference CTE
    - (X) SELECT i1, i2 FROM (SELECT * FROM CTE_RECURSIVE) cte, t1 WHERE cte.c1 = t1.i2

〈search clause〉

It describes the sort order of CTE result records.
Sibling rows are sorted with 〈ordering column list〉, it specifies the returning order of sibling rows and child rows for the sorted records.
The sequence of result records are stored in 〈sequence column〉.

- DEPTH FIRST BY
  - Child rows are returned before sibling rows are returned.
- BREADTH FIRST BY
  - Sibling rows are returned before child rows are returned.

```
gSQL>
SELECT * FROM t1;
I1  I2
--- ---
A   ---
AA  A
```

```
AB  A
AC  A
AAX AA
ABX AB
ACX AC
7 rows selected.
* SEARCH BREADTH FIRST BY
gSQL>
WITH w1( w_i1, w_i2 ) AS
    (
         SELECT i1, i2
           FROM t1
          WHERE i1 = 'A'
         UNION ALL
         SELECT i1, i2
           FROM w1, t1
          WHERE w_i1 = i2
    ) SEARCH BREADTH FIRST BY w_i1, w_i2 SET w_seq
SELECT w_i1, w_i2, w_seq
 FROM w1;
W_I1 W_I2 W_SEQ
---- ---- -----
A    ---      1
AA   A        2
AB   A        3
AC   A        4
AAX  AA       5
ABX  AB       6
ACX  AC       7
7 rows selected.
* SEARCH DEPTH FIRST BY
gSQL>
WITH w1( w_i1, w_i2 ) AS
    (
         SELECT i1, i2
           FROM t1
          WHERE i1 = 'A'
         UNION ALL
         SELECT i1, i2
           FROM w1, t1
          WHERE w_i1 = i2
```

```
    ) SEARCH DEPTH FIRST BY w_i1, w_i2 SET w_seq
SELECT w_i1, w_i2, w_seq
  FROM w1;
W_I1 W_I2 W_SEQ
---- ---- -----
A    ---      1
AA   A        2
AAX  AA       3
AB   A        4
ABX  AB       5
AC   A        6
ACX  AC       7
7 rows selected.
```

〈cycle clause〉

If 〈cycle clause〉 statement is not described, then an error occurs when cycle occurs.

〈cycle column list〉 is used to check cycle.
It stores 〈cycle mark value〉 or 〈non-cycle mark value〉 in 〈cycle mark column〉 according to whether cycle occurs.
Only 1 byte character can be described in 〈cycle mark value〉 or 〈non-cycle mark value〉.
〈cycle mark value〉 is stored in 〈cycle mark column〉 of the record where cycle occurred. In this case, it sto ps further recursion, and returns only until the record where cycle occurred
The recursion continuously proceeds for sibling rows where cycle does not occurred.

```
gSQL>
SELECT * FROM t1;
I1  I2
--- ---
A   ---
AA  A
AB  A
AC  A
AA  AA
AAX AA
ABX AB
ACX AC
8 rows selected.
```

- When cycle occurred and cycle clause is not described

```
gSQL>
WITH w1( w_i1, w_i2 ) AS
     (
          SELECT i1, i2
            FROM t1
           WHERE i1 = 'A'
          UNION ALL
          SELECT i1, i2
            FROM w1, t1
           WHERE w_i1 = i2
     )
SELECT w_i1, w_i2
  FROM w1;
ERR-42000(16511): cycle detected while executing recursive WITH query
```

- When cycle occurred and cycle clause is described

```
gSQL>
WITH w1( w_i1, w_i2 ) AS
     (
          SELECT i1, i2
            FROM t1
           WHERE i1 = 'A'
          UNION ALL
          SELECT i1, i2
            FROM w1, t1
           WHERE w_i1 = i2
     ) CYCLE w_i1, w_i2 SET c_cycle TO 'T' DEFAULT 'F'
SELECT w_i1, w_i2, c_cycle
  FROM w1;
W_I1 W_I2 C_CYCLE
---- ---- -------
A    ---  F
AC   A    F
AB   A    F
AA   A    F
ACX  AC   F
ABX  AB   F
AAX  AA   F
AA   AA   F
AAX  AA   F
```

```
AA   AA   T
10 rows selected.
```

Examples

The following is an example of SELECT statement which uses WITH clause.

- Non-recursive CTE

```
gSQL>
WITH revenue ( supplier_no, total_revenue ) AS
      (
            SELECT
                  l_suppkey,
                  SUM(l_extendedprice * (1 - l_discount))
             FROM lineitem
            WHERE l_shipdate >= DATE '1996-01-01'
              AND l_shipdate < DATE '1996-01-01' + INTERVAL '3' MONTH
            GROUP BY
                  l_suppkey
      )
select
       s_suppkey,
       s_name,
       s_address,
       s_phone,
       ROUND( total_revenue, 2 ) as total_revenue
  from
       supplier,
       revenue
 where
       s_suppkey = supplier_no
   and total_revenue = (
                           select
                                 max(total_revenue)
                             from
                                 revenue
                       )
order by
      s_suppkey;
S_SUPPKEY S_NAME                    S_ADDRESS         S_PHONE         TOTAL_REVENUE
```

```
--------- ------------------------- ------------------ --------------- -------------
     8449 Supplier#000008449        Wp34zim9qYFbVctdW 20-469-856-8873    1772627.21
1 row selected.
```

- Recursive CTE

```
gSQL>
WITH GenerateRecord ( c1, c2 ) AS
     (
          SELECT 1, 11
            FROM dual
          UNION ALL
          SELECT c1 + 1, c2 + 1
            FROM GenerateRecord
           WHERE c1 < 10
     )
SELECT c1, c2 FROM GenerateRecord;
C1 C2
-- --
 1 11
 2 12
 3 13
 4 14
 5 15
 6 16
 7 17
 8 18
 9 19
10 20
10 rows selected.
```

The following is the result of retrieving record in emp table which will be used in the example of WITH cla use.

```
gSQL>
SELECT * FROM emp;
NAME    MGR
------- -------
Kelly   null
Bill    Kelly
Jackson Kelly
Joe     Kelly
```

```
Scott   Bill
Larry   Bill
Paul    Jackson
Bill    Bill
8 rows selected.
```

The following is an example of using SEARCH BREADTH FIRST BY.

- Cycle occurred

```
gSQL>
WITH w_emp( w_name, w_mgr ) AS
     (
         SELECT name, mgr
           FROM emp
          WHERE mgr IS NULL
         UNION ALL
         SELECT name, mgr
           FROM emp, w_emp
          WHERE mgr = w_emp.w_name
     ) SEARCH BREADTH FIRST BY w_name SET w_seq
SELECT w_name, w_mgr, w_seq
  FROM w_emp;
ERR-42000(16511): cycle detected while executing recursive WITH query
```

- Retrieve the data by describing cycle clause in the statement above after cycle occurred.

```
gSQL>
WITH w_emp( w_name, w_mgr ) AS
     (
         SELECT name, mgr
           FROM emp
          WHERE mgr IS NULL
         UNION ALL
         SELECT name, mgr
           FROM emp, w_emp
          WHERE mgr = w_emp.w_name
     ) SEARCH BREADTH FIRST BY w_name SET w_seq
       CYCLE w_name SET w_cycle TO 'T' DEFAULT 'F'
SELECT w_name, w_mgr, w_seq, w_cycle
  FROM w_emp;
W_NAME  W_MGR   W_SEQ W_CYCLE
```

```
------- ------- ----- -------
Kelly   null        1 F
Bill    Kelly       2 F
Jackson Kelly       3 F
Joe     Kelly       4 F
Bill    Bill        5 T
Larry   Bill        6 F
Paul    Jackson     7 F
Scott   Bill        8 F
8 rows selected.
```

The following is an example of using SEARCH DEPTH FIRST BY.

```
gSQL>
WITH w_emp( w_name, w_mgr ) AS
     (
          SELECT name, mgr
            FROM emp
           WHERE mgr IS NULL
          UNION ALL
          SELECT name, mgr
            FROM emp, w_emp
           WHERE mgr = w_emp.w_name
     ) SEARCH DEPTH FIRST BY w_name SET w_seq
       CYCLE w_name SET w_cycle TO 'T' DEFAULT 'F'
SELECT w_name, w_mgr, w_seq, w_cycle
  FROM w_emp;
W_NAME  W_MGR   W_SEQ W_CYCLE
------- ------- ----- -------
Kelly   null        1 F
Bill    Kelly       2 F
Bill    Bill        3 T
Larry   Bill        4 F
Scott   Bill        5 F
Jackson Kelly       6 F
Paul    Jackson     7 F
Joe     Kelly       8 F
8 rows selected.
```

The following is an example of using *with clause* in CREATE TABLE AS SELECT statement.

```
gSQL>
CREATE TABLE new_emp AS
WITH w_emp( w_name, w_mgr ) AS
     (
          SELECT name, mgr
            FROM emp
           WHERE mgr IS NULL
          UNION ALL
          SELECT name, mgr
            FROM emp, w_emp
           WHERE mgr = w_emp.w_name
     ) SEARCH BREADTH FIRST BY w_name SET w_seq
       CYCLE w_name SET w_cycle TO 'T' DEFAULT 'F'
SELECT w_name, w_mgr, w_seq, w_cycle
  FROM w_emp;
Table created.
```

The following is an example of using *with clause* in INSERT statement.

```
gSQL>
INSERT INTO new_emp
WITH w_emp( w_name, w_mgr ) AS
     (
          SELECT name, mgr
            FROM emp
           WHERE mgr = 'Bill'
          UNION ALL
          SELECT name, mgr
            FROM emp, w_emp
           WHERE mgr = w_emp.w_name
     ) SEARCH BREADTH FIRST BY w_name SET w_seq
       CYCLE w_name SET w_cycle TO 'T' DEFAULT 'F'
SELECT w_name, w_mgr, w_seq, w_cycle
  FROM w_emp;
6 rows created.
```

The following is an example of using *with clause* in UPDATE statement.

```
gSQL>
UPDATE new_emp SET w_name = NULL
 WHERE ( w_name, w_mgr )
```

```
       IN ( WITH w_emp( w_name, w_mgr ) AS
                (
                    SELECT name, mgr
                      FROM emp
                     WHERE mgr = 'Bill'
                    UNION ALL
                    SELECT name, mgr
                      FROM emp, w_emp
                     WHERE mgr = w_emp.w_name
                ) SEARCH BREADTH FIRST BY w_name SET w_seq
                  CYCLE w_name SET w_cycle TO 'T' DEFAULT 'F'
             SELECT w_name, w_mgr
               FROM w_emp );
9 rows updated.
```

The following is an example of using *with clause* in DELETE statement.

```
gSQL>
DELETE FROM new_emp
WHERE ( w_mgr )
      IN ( WITH w_emp( w_name, w_mgr ) AS
                (
                    SELECT name, mgr
                      FROM emp
                     WHERE mgr = 'Bill'
                    UNION ALL
                    SELECT name, mgr
                      FROM emp, w_emp
                     WHERE mgr = w_emp.w_name
                ) SEARCH BREADTH FIRST BY w_name SET w_seq
                  CYCLE w_name SET w_cycle TO 'T' DEFAULT 'F'
           SELECT w_mgr
             FROM w_emp );
9 rows deleted.
```

The following is an example of using *with clause* in CREATE VIEW statement.

```
gSQL>
CREATE VIEW v_emp AS
WITH w_emp( w_name, w_mgr ) AS
     (
          SELECT name, mgr
```

```
            FROM emp
           WHERE mgr IS NULL
          UNION ALL
          SELECT name, mgr
            FROM emp, w_emp
           WHERE mgr = w_emp.w_name
     ) SEARCH BREADTH FIRST BY w_name SET w_seq
       CYCLE w_name SET w_cycle TO 'T' DEFAULT 'F'
SELECT w_name, w_mgr, w_seq, w_cycle
  FROM w_emp;
View created.
```

query specification

Function

It specifies the table which is derived from the result of <table expression>.

Syntax

```
<query specification> ::=
    SELECT [ <hint clause> ] [ <set quantifier> ] <select list> <table expression>
<set quantifier> ::=
      ALL
    ¦ DISTINCT
<table expression> ::=
      <from clause> [ <where clause> ] [ <hierarchical query clause> ] [ <group by clause> ] [
<having clause> ] [ <window clause> ]
```

Invocation and Access Rules

The user should satisfy one of the following conditions to perform <query specification>.

- The owner of that table
- SELECT privilege for the table
- The user owns one of SELECT TABLE, CONTROL TABLE, CONTROL privileges for the schema to which the table belongs
- The user owns the SELECT TABLE privilege for the database

Syntax Rules and Parameters

<hint clause>

It specifies the hint for query execution.
For more information, refer to **SQL Hint**.

<set quantifier>

It specifies whether to remove a duplicate of the query result.
If it is omitted, it operates in the same way as ALL.

〈select list〉

It specifies the column to be retrieved among query results.
For more information, refer to **select list**.

〈from clause〉

It specifies the tables to be retrieved.
For more information, refer to **from clause**.

〈where clause〉

It specifies conditions for retrieving.
For more information, refer to **where clause**.

〈hierarchical query clause〉

It specifies to retrieve the hierarchical model data in a hierarchy.
For more information, refer to **hierarchical query clause**.

〈group by clause〉

It specifies grouping of the query result.
For more information, refer to **group by clause**.

〈having clause〉

It specifies conditions for the grouping result.
For more information, refer to **having clause**.

〈window clause〉

It defines the execution range of the window function.
For more information, refer to **window clause**.

Description

〈hint clause〉

〈hint clause〉 is a comment which the user uses to directly command an optimizer how to execute SQL st
atement.

The optimizer of GOLDILOCKS preferentially applies 〈hint clause〉 specified by a user.
If it is not applicable, the optimizer selects the best execution plan through the cost calculation.

Even when a syntactic error occurs in 〈hint clause〉, GOLDILOCKS is set to ignore and perform it by defaul t. Set **HINT_ERROR** property to *on*, then execute the query to check if a syntactic error exist in 〈hint claus e〉.

〈set quantifier〉

〈set quantifier〉 sets whether to remove duplicates from the result set consisting of the 〈select list〉 expre ssions.

- ALL: It does not remove the duplicates from the result set.
- DISTINCT: It removes the duplicates from the result set.
- If it is omitted, it is operated by default which is as same as ALL.

〈select list〉

It specifies columns to be retrieved from the query result.
They are listed by separating by a comma (,).
An asterisk (*) is used to specify all columns in 〈from clause〉.

〈from clause〉

〈from clause〉 specifies the tables or views to be retrieved.

〈where clause〉

〈where clause〉 specifies the conditions to get only the desired results from the result obtained from 〈fro m clause〉.

〈hierarchical query clause〉

It specifies to retrieve the hierarchical model data in a hierarchy.
It returns table records in a hierarchy of depth-first sequence by using the launch condition and sub-conn ectivity condition.

〈group by clause〉

〈group by clause〉 specifies the method of grouping the result set to which 〈where clause〉 was applied.

When 〈group by clause〉 is specified, the following expressions can be used in 〈select list〉.

- Constant number

- Expression specified in *group by*
- Operation expression specified in *group by*
- Aggregation function for an expression belonging to a group

〈having clause〉

〈having clause〉 specifies the retrieving condition for the grouped result set.
It is generally used together with 〈group by clause〉.

〈window clause〉

It specifies the execution range of the window function described in *select list* and *order by* clause

Examples

The following is an example of SELECT statement which uses 〈hint clause〉.

```
gSQL> SELECT /*+ INDEX_DESC(supplier, supplier_pk_index) */ s_name, s_nation FROM supplier;
S_NAME                    S_NATION
------------------------- -------------
Supplier#5                CANADA
Supplier#4                UNITED STATES
Supplier#3                GERMANY
Supplier#2                KOREA
Supplier#1                FRANCE
5 rows selected.
```

The following is an example of SELECT statement which uses 〈set quantifier〉.

```
gSQL> SELECT ALL p_type FROM part;
P_TYPE
------
COPPER
NICKEL
STEEL
NICKEL
STEEL
5 rows selected.
gSQL> SELECT DISTINCT p_type FROM part;
P_TYPE
------
COPPER
```

```
STEEL
NICKEL
3 rows selected.
```

The following is an example of SELECT statement which uses <where clause>.

```
gSQL> SELECT p_name, p_brand, p_type, p_size FROM part where p_size < 10;
P_NAME P_BRAND    P_TYPE P_SIZE
------ ---------- ------ ------
Part#1 Brand#1    COPPER      7
Part#2 Brand#1    NICKEL      1
2 rows selected.
```

The following is an example of retrieving the hierarchy data of SELECT statement by using <hierarchical query clause>.

```
gSQL> SELECT *
        FROM emp
      START WITH mgr IS NULL
      CONNECT BY NOCYCLE mgr = PRIOR name
      ORDER SIBLINGS BY name;
NAME    MGR
------- -------
Kelly   null
Bill    Kelly
Larry   Bill
Scott   Bill
Jackson Kelly
Paul    Jackson
Joe     Kelly
7 rows selected.
```

The following is an example of SELECT statement which uses <group by clause>.

```
gSQL> SELECT ps_partkey, SUM(ps_availqty) FROM partsupp GROUP BY ps_partkey;
PS_PARTKEY SUM(PS_AVAILQTY)
---------- ----------------
         1            11401
         2             8025
         3            13864
         4            11564
         5             8744
```

```
5 rows selected.
```

The following is an example of SELECT statement which uses <having clause>.

```
gSQL> SELECT ps_partkey, SUM(ps_availqty) FROM partsupp GROUP BY ps_partkey having
SUM(ps_availqty) > 10000;
PS_PARTKEY SUM(PS_AVAILQTY)
---------- ----------------
         1            11401
         3            13864
         4            11564
3 rows selected.
```

The following is an example of SELECT statement which uses <window clause>.

```
gSQL>
SELECT item_no,
       sales_date,
       sales,
       SUM( sales ) OVER W1 cumulative_sales,
       AVG( sales ) OVER w1 avg_sales
  FROM store
WINDOW w1 AS ( PARTITION BY item_no
               ORDER BY sales_date
               ROWS BETWEEN UNBOUNDED PRECEDING
                        AND CURRENT ROW );
ITEM_NO SALES_DATE SALES CUMULATIVE_SALES AVG_SALES
------- ---------- ----- ---------------- ---------
    100 2001-01-01   150              150       150
    100 2001-01-02   100              250       125
    100 2001-01-03   170              420       140
    100 2001-01-04    90              510     127.5
    100 2001-01-05   200              710       142
    235 2001-01-01    70               70        70
    235 2001-01-02   130              200       100
    235 2001-01-03   190              390       130
    235 2001-01-04   150              540       135
    235 2001-01-05    50              590       118
10 rows selected.
```

Compatibility

Table 20-10 SQL standard compatibility

Feature ID	Description	Compatibility
F801	Full set function	X
T051	Row types	X
T301	Functional dependencies	X
T325	Qualified SQL parameter references	X
T053	Explicit aliases for all-fields reference	O
T285	Enhanced derived column names	O

For More Information

Refer to **query expression**.

select list

Function

It specifies the columns to be retrieved from the query result.

Syntax

```
<select list> ::=
      <asterisk>
    ¦ <select sublist> [ { <comma> <select sublist> } ... ]
<select sublist> ::=
      <derived column>
    ¦ <qualified asterisk>
<qualified asterisk> ::=
      <asterisked identifier chain> <period> <asterisk>
<asterisked identifier chain> ::=
    <asterisked identifier> [ { <period> <asterisked identifier> } ... ]
<derived column> ::=
    <value expression> [ <as clause> ]
<as clause> ::=
    [ AS ] <column name>
```

Invocation and Access Rules

If columns or subqueries exist in <select list> statement, the user should satisfy the followings.

- The access privileges for the columns
- The access privileges for the table and columns of the subqueries

Syntax Rules and Parameters

<select list>

It has <asterisk> or <select sublist>.

〈asterisk〉

- 〈asterisk〉 can be used only alone in 〈select list〉.
  - (O) SELECT * FROM t1;
  - (X) SELECT *, c1 FROM t1;

〈select sublist〉

- It has 〈derived column〉 or 〈qualified asterisk〉.
  - SELECT c1, c2 FROM t1;
  - SELECT t1.* FROM t1;
- 〈derived column〉 can change the output name by using *AS*, and *AS* can be omitted.
  - SELECT c1 AS col1, c2 AS col2 AS FROM t1;
  - SELECT c1 col1, c2 col2 FROM t1;
- If two or more 〈select sublist〉 are specified, each 〈select sublist〉 should be separated by a comma (,).
  - (O) SELECT c1, c2 FROM t1;
  - (O) SELECT c1, c2, t1.* FROM t1;
  - (X) SELECT c1 c2 FROM t1;
    - c2 is processed as ALIAS.
  - (X) SELECT c1 c2 c3 FROM t1;

Description

〈select list〉

〈select list〉 specifies the columns to be included in the result set.

〈asterisk〉

〈asterisk〉 sets all columns in 〈from clause〉 as a select list.

〈select sublist〉

〈select sublist〉 has 〈derived column〉 or 〈qualified asterisk〉.

- 〈qualified asterisk〉
  - It sets all columns belonging to a specific table or a view as a select list.
- 〈derived column〉
  - It can specifies a column or 〈value expression〉.
  - The column name can be updated by using 〈as clause〉, and *AS* can be omitted.
  - If 〈from clause〉 has tables with the same column name, then the table name or the table alias sh ould be specified to refer to that columns.

    - SELECT T1.C1, T2.C1 FROM T1, T2;
    - SELECT A.C1, B.C1 FROM T1 A, T2 B;

If two or more <select sublist> are specified, each <select sublist> should be separated by a comma (',').

Names to Be Set in select list

- When <column name> is specified in <derived column>, that name is set as a select list name
  - SELECT i1 AS name FROM t1;
- When <column name> is not specified in <derived column>
  - When <derived column> is a single column reference
    - The column name of a single column is set as a select list name.
    - SELECT i1 FROM t1;
  - If <derived column> is not a column but it is an expression
    - The select list name is not set.
    - SELECT i1 + 100 FROM t1;
    - If it is written in CREATE TABLE AS SELECT clause, the column name should be specified.
    - CREATE TABLE t2 AS SELECT i1 + 100 AS sum_i1 FROM t1;

Examples

The following is an example of SELECT statement which uses <asterisk>.

```
gSQL> SELECT * FROM supplier;
S_SUPPKEY S_NAME                    S_NATION      S_PHONE
--------- ------------------------- ------------- ---------------
        1 Supplier#1                FRANCE        27-918-335-1736
        2 Supplier#2                KOREA         15-679-861-2259
        3 Supplier#3                GERMANY       11-383-516-1199
        4 Supplier#4                UNITED STATES 25-843-787-7479
        5 Supplier#5                CANADA        21-151-690-3663
5 rows selected.
```

The following is an example of SELECT statement which uses <select sublist>.

```
gSQL> SELECT revenue.* FROM revenue;
SUPPLIER_NO TOTAL_REVENUE
----------- -------------
          1      11978.64
          2       20321.5
          3      41844.68
3 rows selected.
```

```
gSQL> SELECT supplier_no suppno, total_revenue AS TOTAL FROM revenue;
SUPPNO    TOTAL
------ --------
     1 11978.64
     2  20321.5
     3 41844.68
3 rows selected.
gSQL> SELECT 1, revenue.*, CAST( total_revenue AS NATIVE_INTEGER ) TOTAL FROM revenue;
1 SUPPLIER_NO TOTAL_REVENUE TOTAL
- ----------- ------------- -----
1           1      11978.64 11979
1           2       20321.5 20322
1           3      41844.68 41845
3 rows selected.
```

For More Information

Refer to **query specification**.

from clause

Function

It specifies the table which is derived from one or more tables.

Syntax

```
<from clause> ::=
    FROM <table reference list>
<table reference list> ::=
    <table reference> [ { , <table reference> } ... ]
<table reference> ::=
      <table factor>
    ¦ <joined table>
<table factor> ::=
    <table primary>
<table primary> ::=
      <table name> [ <cluster domain> ] [ [ AS ] <correlation name> ]
    ¦ <derived table> [ <cluster domain> ] [ [ AS ] <correlation name> [ <left paren> <derived
column list> <right paren> ] ]
    ¦ <lateral derived table> [ <cluster domain> ] [ [ AS ] <correlation name> [ <left paren>
<derived column list> <right paren> ] ]
    ¦ <table function derived table> [ [ AS ] <correlation name> ]
    ¦ <parenthesized joined table>
<derived table> ::=
    <table subquery>
<lateral derived table> ::=
    LATERAL <table subquery>
<parenthesized joined table> ::=
      <left paren> <parenthesized joined table> <right paren>
    ¦ <left paren> <joined table> <right paren>
<derived column list> ::=
    <column name list>
<cluster domain> ::=
    @ <cluster domain name>
<cluster domain name> ::=
      GLOBAL
    ¦ LOCAL
    ¦ LOCAL_OFFLINE
```

```
    ¦ <identifier>
<table function derived table> ::=
      TABLE <left paren> <table function expression> <right paren>
<table function expression> ::=
      <table function name> <left paren> [ <table function argument list> ] <right paren>
<table function argument list> ::=
      <value expression> [ <comma> ... ]
```

Invocation and Access Rules

The access privilege for the table or view specified in <table reference list> is required.

Syntax Rules and Parameters

<table reference list>

- One or more tables can be specified in <table reference list> by using a comma (,).
- When two or more tables are specified
  - The evaluation order for the tables is from left to right.
  - When * is specified in <select list>, the columns are sequentially mapped in <select list> from the l eft table to the right table.

<table primary>

- An alias name can be specified by using <correlation name>.
  - SELECT * FROM t1 AS a, t2 AS b;
  - SELECT * FROM (SELECT i1 FROM t1) AS a;

- <derived table> as known as <table subquery>
  - can specify an alias name by using <correlation name>.
    - SELECT * FROM (SELECT i1, i2, i3 FROM t1) AS a;
  - can specify <derived column list>.
    - SELECT * FROM (SELECT i1, i2, i3 FROM t1) AS a(col1, col2, col3);
    - The number of <column name> in <derived column list> should be same as the number of tar gets in <select list> specified in <table subquery>.
    - It is sequentially mapped 1 :1 to the target in <select list> specified in <table subquery>.
    - It should use <column name> specified in <derived column list> to refer to <select list> of <ta ble subquery> in that <derived table>.

```
SELECT col1, col2
FROM ( SELECT i1, i2 FROM t1 ) AS a( col1, col2 )
```

```
WHERE col1 = 1 AND col2 = 1;
```

- 〈lateral derived table〉
  - If LATERAL is specified in front of 〈table subquery〉, then it becomes the lateral inline view.
  - The lateral inline view can refer to tables listed in FROM clause of the main query in all clauses wit hin 〈table subquery〉.
    - However, it can refer to only the table which was specified before the lateral inline view. If it i s RIGHT OUTER JOIN or FULL OUTER JOIN, then it can not refer to the table even when it was specified beforehand.

- 〈table function derived table〉
  - table function derived table is a logical table which consists of the result sets of executing the tab le function. For more information about the table function derived table, refer to **Table Function Derived Table**.
  - table function derived table specifies TABLE and the table function name to execute.
  - 〈table function argument list〉 of 〈table function expression〉 can refer to the columns of the tabl e listed before 〈table function derived table〉 in FROM clause.
    - However, if it is RIGHT OUTER JOIN or FULL OUTER JOIN, then it can not be referred even wh en the table is specified beforehand.

```
SELECT t1.col1, ft.rf2
FROM t1, TABLE( tablefunc( t1.col1 ) );
```

〈correlation name〉

- The same 〈correlation name〉 should not exist two or more in 〈table reference list〉.
- When 〈correlation name〉 is specified, 〈correlation name〉 should be used to refer to 〈table name〉 o r 〈derived table〉.
  - SELECT a.i1 FROM t1 AS a WHERE a.i1 > 3;
  - (X) SELECT t1.i1 FROM t1 AS a WHERE t1.i1 > 3;
- When specifying 〈correlation name〉, AS can be omitted.
  - SELECT a.i1 FROM t1 a;

〈derived column list〉

The same 〈column name〉 should not exist two or more in 〈derived column list〉.

〈cluster domain〉

- 〈cluster domain〉 can be specified in a table, a view, or a table subquery.
  - SELECT * FROM t1@G1;
  - It can not be specified in 〈parenthesized joined table〉.
    - (X) SELECT * FROM (t1 INNER JOIN t2 ON t1.sk = t2.sk)@G2;

- <cluster domain> can not be specified in a table or a view whose structure or data is to be altered.
  - (X) DELETE FROM t2@GLOBAL;
  - (X) UPDATE t1@GLOBAL SET i1 = 1;
  - (X) INSERT INTO t1@GLOBAL VALUES (1, 10);
  - (X) SELECT * FROM t1@GLOBAL FOR UPDATE;
  - (X) CREATE INDEX t1_idx ON t1@GLOBAL(i1);

<cluster domain name>

Only a cluster group name or a cluster member name can be <identifier> of <cluster domain name>.

- cluster group name
  - SELECT * FROM t1@G1;
- cluster member name
  - SELECT * FROM t1@G1N1;

Description

<table reference list>

Two or more tables can be specified in <table reference list> by using a comma (,).

- If two or more tables are specified, it operates in the same way as cross join each table from left to ri ght.
  - SELECT * FROM t1, t2;
  - <=> SELECT * FROM t1 CROSS JOIN t2;
- If the conditions to join two tables exist in <where clause>, the two tables operate in the same way as inner join which has <where clause> as a join condition.
  - SELECT * FROM t1, t2 WHERE t1.I1 = t2.I1;
  - <=> SELECT * FROM t1 INNER JOIN t2 ON t1.i1 = t2.i1;
- If outer join operator (+) is used in <where clause>, it operates in the same way as outer join.
  - For more information about outer join operator (+), refer to **OUTER JOIN**.

<table reference>

A single table, or view, table subquery, joined table can be <table reference>. Others except for the joine d table can have a correlation name.

For more information about joined table, refer to **joined table**.

〈table primary〉

The tables, views, table subqueries and 〈parenthesized joined table〉 can be 〈table primary〉.

The table, view, table subquery can have a correlation name, and AS can be omitted. If the correlation na me is specified, it should be used in everywhere referring to the table, view, table subquery such as 〈selec t list〉, 〈where clause〉.

The table subquery can specify 〈derived column list〉. The name specified in 〈derived column list〉 should be used in everywhere referring to table subquery column as like correlation name. To use 〈derived colu mn list〉 in the table subquery, the correlation name should be specified.

〈parenthesized joined table〉 specifies the logical join order for the table participating in the join operatio n. At this time, if all join operations for the tables enclosed with parentheses are cross join, inner join, the join order can be changed by an optimizer.

〈cluster domain〉

When 〈cluster domain〉 is omitted, it means the same as using GLOBAL as 〈cluster domain name〉.
For more information, refer to **Cluster Domain**.

〈cluster domain name〉

The reserved words defined in 〈cluster domain name〉 mean as follows.

- GLOBAL
  - It selects all cluster groups as a cluster domain.
- LOCAL
  - It selects only the server performing the user query as a cluster domain.
    - It brings data of G2N1 when performing the following query in G2N1.
    - SELECT * FROM t1@LOCAL;
- LOCAL_OFFLINE
  - It selects only the server performing the user query as a cluster domain to retrieve the offline tabl e data.
    - It brings data of offline table T1 in G2N1 when performing the following query in G2N1.
    - SELECT * FROM t1@LOCAL_OFFLINE;
  - If LOCAL_OFFLINE domain is specified in an online table, then an error occurs.

If 〈identifier〉 is specified in 〈cluster domain name〉, a cluster group or a cluster member with the corresp onding name is selected as **Cluster Domain**.

Examples

The following is an example of SELECT statement to query a single table by using <table name>.

```
gSQL> SELECT c_name, c_nation FROM customer;
C_NAME     C_NATION
---------- -------------
Customer#1 KOREA
Customer#2 CANADA
Customer#3 KOREA
Customer#4 GERMANY
Customer#5 UNITED STATES
5 rows selected.
```

The following is an example of SELECT statement which uses <derived table>.

```
gSQL> SELECT * FROM (SELECT c_name, c_nation FROM customer);
C_NAME     C_NATION
---------- -------------
Customer#1 KOREA
Customer#2 CANADA
Customer#3 KOREA
Customer#4 GERMANY
Customer#5 UNITED STATES
5 rows selected.
gSQL> SELECT * FROM (SELECT c_name, c_nation FROM customer) AS CUST ("CUSTOMER_NAME",
"CUSTOMER_NATION");
CUSTOMER_NAME CUSTOMER_NATION
------------- ---------------
Customer#1    KOREA
Customer#2    CANADA
Customer#3    KOREA
Customer#4    GERMANY
Customer#5    UNITED STATES
5 rows selected.
```

The following is an example of SELECT statement using <lateral derived table>.

```
gSQL> SELECT r_name, n_name
  FROM region, (  SELECT n_name
                    FROM nation
                   WHERE n_regionkey = r_regionkey
```

```
               ) v_nation
 WHERE r_name = 'ASIA';
ERR-42000(16036): 'R_REGIONKEY': invalid identifier :
                  WHERE n_regionkey = r_regionkey
                                      *
gSQL> SELECT r_name, n_name
  FROM region, LATERAL (  SELECT n_name
                            FROM nation
                           WHERE n_regionkey = r_regionkey
                       ) v_nation
 WHERE r_name = 'ASIA';
R_NAME                    N_NAME
------------------------- -------------------------
ASIA                      INDIA
ASIA                      INDONESIA
ASIA                      JAPAN
ASIA                      CHINA
ASIA                      VIETNAM
5 rows selected.
```

The following is an example of SELECT statement for a joined table which uses parentheses.

```
gSQL> SELECT customer.c_name, o_totalprice FROM (customer INNER JOIN orders ON
customer.c_custkey = orders.o_custkey);
C_NAME     O_TOTALPRICE
---------- ------------
Customer#1    173665.47
Customer#2     46929.18
Customer#4    193846.25
Customer#3     32151.78
Customer#5     144659.2
5 rows selected.
```

The following is an example of SELECT statement which uses two table separated by a comma (,).

```
gSQL> SELECT c_name, o_totalprice FROM customer, orders;
C_NAME     O_TOTALPRICE
---------- ------------
Customer#1    173665.47
Customer#1     46929.18
Customer#1    193846.25
Customer#1     32151.78
```

```
Customer#1     144659.2
Customer#2    173665.47
Customer#2     46929.18
Customer#2    193846.25
Customer#2     32151.78
Customer#2     144659.2
Customer#3    173665.47
Customer#3     46929.18
Customer#3    193846.25
Customer#3     32151.78
Customer#3     144659.2
Customer#4    173665.47
Customer#4     46929.18
Customer#4    193846.25
Customer#4     32151.78
Customer#4     144659.2
C_NAME     O_TOTALPRICE
---------- ------------
Customer#5    173665.47
Customer#5     46929.18
Customer#5    193846.25
Customer#5     32151.78
Customer#5     144659.2
25 rows selected.
```

The following is an example of SELECT statement which uses <cluster domain>.

- Using the reserved word GLOBAL

```
gSQL> SELECT * FROM (SELECT c_name, c_nation FROM customer@GLOBAL);
C_NAME     C_NATION
---------- -------------
Customer#1 KOREA
Customer#2 CANADA
Customer#3 KOREA
Customer#4 GERMANY
Customer#5 UNITED STATES
5 rows selected.
```

- Using the reserved word LOCAL

```
gSQL> SELECT * FROM (SELECT c_name, c_nation FROM customer)@LOCAL;
C_NAME     C_NATION
---------- -------------
Customer#1 KOREA
Customer#2 CANADA
2 rows selected.
```

- Using the cluster group name G1

```
gSQL> SELECT * FROM (SELECT c_name, c_nation FROM customer@G1);
C_NAME     C_NATION
---------- -------------
Customer#1 KOREA
Customer#2 CANADA
2 rows selected.
```

- Using the cluster member name G2N1

```
gSQL> SELECT * FROM (SELECT c_name, c_nation FROM customer@G2N1);
C_NAME     C_NATION
---------- -------------
Customer#3 KOREA
Customer#4 GERMANY
2 rows selected.
```

For More Information

Refer to **subquery**.

joined table

Function

It specifies the table derived from a cartesian product, inner join, outer join.

Syntax

```
<joined table> ::=
      <cross join>
    ¦ <qualified join>
    ¦ <natural join>
<cross join> ::=
    <table reference> CROSS JOIN <table factor>
<qualified join> ::=
    <table reference> [ <join type> ] JOIN <table reference> <join specification>
<natural join> ::=
    <table reference> NATURAL [ <join type> ] JOIN <table factor>
<join specification> ::=
      <join condition>
    ¦ <named columns join>
<join condition> ::=
    ON <search condition>
<named columns join> ::=
    USING ( <join column list> )
<join type> ::=
      INNER
    ¦ { LEFT ¦ RIGHT ¦ FULL } [ OUTER ]
<join column list> ::=
    <column name list>
```

Invocation and Access Rules

The access privilege for all tables and views specified in a joined table is required.

Syntax Rules and Parameters

〈cross join〉

〈join specification〉 specifying the join condition does not appear at the location of 〈cross join〉.
A single table, 〈table subquery〉 or 〈parenthesized joined table〉 can appear on the right of 〈cross join〉.

〈qualified join〉

- 〈join specification〉 specifying the join condition should be specified.
  - SELECT * FROM t1 INNER JOIN t2 ON t1.i1 = t2.i1;
  - SELECT * FROM t1 INNER JOIN t2 USING (i1);
- 〈join type〉 can be omitted, and it is processed as INNER when omitted.
  - SELECT * FROM t1 JOIN t2 ON t1.i1 = t2.i1;
  - 〈=〉 SELECT * FROM t1 INNER JOIN t2 ON t1.i1 = t2.i1;
- OUTER can be omitted in 〈join type〉.
  - SELECT * FROM t1 LEFT JOIN t2 ON t1.i1 = t2.i1;
  - 〈=〉 SELECT * FROM t1 LEFT OUTER JOIN t2 ON t1.i1 = t2.i1;
- If 〈join type〉 is OUTER JOIN, then only 〈join condition〉 can appear on 〈join specification〉.
  - SELECT * FROM t1 FULL OUTER JOIN t2 ON t1.i1 = t2.i1;
  - (X) SELECT * FROM t1 FULL OUTER JOIN t2 USING (i1);

〈natural join〉

- 〈join specification〉 specifying the join condition does not appear at the location of 〈natural join〉.
- A single table, 〈table subquery〉 or 〈parenthesized joined table〉 can appear on the right of 〈natural j oin〉.
- 〈join type〉 can be omitted. When it is omitted, it performs INNER.
  - SELECT * FROM t1 NATURAL JOIN t2;
  - 〈=〉 SELECT * FROM t1 NATURAL INNER JOIN t2;
- It does not allow OUTER in 〈join type〉.
  - (X) SELECT * FROM t1 NATURAL LEFT OUTER JOIN t2;
- If the same 〈column name〉 does not exist between the left row and right row of NATURAL JOIN, it p erforms 〈cross join〉.
  - t1(c1 INTEGER, c2 INTEGER);
  - t2(c3 INTEGER, c4 INTEGER);
  - SELECT * FROM t1 NATURAL INNER JOIN t2;
  - 〈=〉 SELECT * FROM t1 CROSS JOIN t2;
- If the same 〈column name〉 exists between the left row and right row of NATURAL JOIN, it is perfor med as if USING clause is specified.
  - t1(c1 INTEGER, c2 INTEGER);
  - t2(c2 INTEGER, c3 INTEGER);
  - SELECT * FROM t1 NATURAL INNER JOIN t2;
  - 〈=〉 SELECT * FROM t1 INNER JOIN t2 USING(c2);

〈join specification〉

- Only one of 〈join condition〉 or 〈named columns join〉 can be specified.
  - 〈join condition〉
    - SELECT * FROM t1 INNER JOIN t2 ON t1.i1 = t2.i1;
  - 〈named columns join〉
    - SELECT * FROM t1 INNER JOIN t2 USING (i1);
- When 〈named columns join〉 is specified
  - One or more column name should be specified in 〈join column list〉.
    - SELECT * FROM t1 INNER JOIN t2 USING (i1);
  - The column name can not be specified such as 〈table name〉.〈column name〉.
    - (X) SELECT * FROM t1 INNER JOIN t2 USING (t1.i1);
  - The listed columns in 〈join column list〉 should be on the left row and right row of JOIN, and they should be able to be compared.
    - t1(c1 INTEGER, c2 INTEGER);
    - t2(c2 INTEGER, c3 INTEGER);
    - SELECT * FROM t1 INNER JOIN t2 USING (c2);
  - If * is used in 〈select list〉, then the records are configured as follows.

    1) Columns specified in 〈join column list〉

    2) Columns which does not correspond to 〈join column list〉 among left rows.

    3) Columns which does not correspond to 〈join column list〉 among right rows.
    - t1(c1 INTEGER, c2 INTEGER);
    - t2(c2 INTEGER, c3 INTEGER);
    - SELECT * FROM t1 INNER JOIN t2 USING (c2);
    - Record configuration: C2, C1, C3
  - 〈column name〉 specified in 〈join column list〉 can not be referenced together with 〈table name〉.〈column name〉, but it can be referenced only by the 〈column name〉.
    - SELECT c2 FROM t1 INNER JOIN t2 USING (c2) WHERE c2 > 3;
    - (X) SELECT t1.c2 FROM t1 INNER JOIN t2 USING (c2);
    - (X) SELECT * FROM t1 INNER JOIN t2 USING (c2) WHERE t1.c2 > 3;
  - Processing the join condition of 〈join column list〉
    - For each column listed in 〈join column list〉
    - The condition 〈left table name〉.〈column name〉 = 〈right table name〉.〈column name〉 is generated
    - and the conditions to process each 〈column name〉 condition using AND are generated.
    - t1(c1 INTEGER, c2 INTEGER);
    - t2(c1 INTEGER, c2 INTEGER);
    - SELECT * FROM t1 INNER JOIN t2 USING (c1, c2);
    - Join condition: t1.c1 = t2.c1 AND t1.c2 = t2.c2
  - 〈table name〉.* statement which returns all the row for a particular table can not be used in 〈select list〉.
    - (X) SELECT t1.*, t2.* FROM t1 INNER JOIN t2 USING (c1, c2);

Description

〈cross join〉

〈cross join〉 returns a result which combines each left row with all right rows.

```
T1 ( 1, 1 ), ( 2, 2 )
T2 ( 2, 2 ), ( 3, 3 )
gSQL> SELECT * FROM t1 CROSS JOIN t2;
C1 C2 C1 C2
-- -- -- --
 1  1  2  2
 1  1  3  3
 2  2  2  2
 2  2  3  3
4 rows selected.
```

The explicit join condition can not be specified in 〈cross join〉, but the join condition for the two tables ca n be specified in 〈where clause〉. In this case, it performs inner join.

- SELECT * FROM t1 CROSS JOIN t2 WHERE t1.c1 = t2.c1;
- 〈=〉 SELECT * FROM t1 INNER JOIN t2 ON t1.c1 = t2.c1;

```
T1 ( 1, 1 ), ( 2, 2 )
T2 ( 2, 2 ), ( 3, 3 )
gSQL> SELECT * FROM t1 CROSS JOIN t2 WHERE t1.c1 = t2.c1;
C1 C2 C1 C2
-- -- -- --
 2  2  2  2
1 row selected.
```

〈qualified join〉

〈qualified join〉 combines each left row with all right rows, then returns only the rows satisfying the join c ondition as a result.

If 〈where clause〉 exists in 〈table expression〉, then conditions in 〈where clause〉 are applied to the result set of 〈qualified join〉.

The result is same, even when inner join processes the conditions in 〈where clause〉 as join conditions. Bu t result differs when outer join processes the conditions in 〈where clause〉 as join conditions.

INNER JOIN

```
t1 ( 1, 1 ), ( 2, 2 ), ( 3, 3 ), ( 4, 4 ), ( 5, 5 )
t2 ( 2, 2 ), ( 3, 3 )
```

- When a condition exists only on ON clause

```
gSQL> SELECT * FROM t1 INNER JOIN t2 ON t1.c1 = t2.c1 AND t1.c2 = t2.c2;
C1 C2 C1 C2
-- -- -- --
 2  2  2  2
 3  3  3  3
2 rows selected.
```

- When a condition exists on ON clause and WHERE clause
  - It applies WHERE condition t1.c2 = t2.c2 to the result set to which JOIN condition ON t1.c1 = t2.c1 is applied.

```
gSQL> SELECT * FROM t1 INNER JOIN t2 ON t1.c1 = t2.c1 WHERE t1.c2 = t2.c2;
C1 C2 C1 C2
-- -- -- --
 2  2  2  2
 3  3  3  3
2 rows selected.
```

  - The result set to which JOIN condition ON t1.c1 = t2.c1 is applied → Apply WHERE condition t1.c2 = t2.c2

```
( 2,  2,    2,    2 )                    ( 2,  2,    2,    2 )
  ( 3,  3,    3,    3 )                →   ( 3,  3,    3,    3 )
```

OUTER JOIN

```
t1 ( 1, 1 ), ( 2, 2 ), ( 3, 3 ), ( 4, 4 ), ( 5, 5 )
t2 ( 2, 2 ), ( 3, 3 )
```

- When a condition exists only on ON clause

```
gSQL> SELECT * FROM t1 LEFT OUTER JOIN t2 ON t1.c1 = t2.c1 AND t1.c2 = t2.c2;
C1 C2   C1   C2
-- -- ---- ----
 1  1 null null
 2  2    2    2
```

```
 3  3    3    3
 4  4 null null
 5  5 null null
5 rows selected.
```

- When a condition exists on ON clause and WHERE clause
  - It applies WHERE condition t1.c2 = t2.c2 to the result set to which JOIN condition ON t1.c1 = t2.c1 is applied.

```
gSQL> SELECT * FROM t1 LEFT OUTER JOIN t2 ON t1.c1 = t2.c1 WHERE t1.c2 = t2.c2;
C1 C2 C1 C2
-- -- -- --
 2  2  2  2
 3  3  3  3
2 rows selected.
```

  - The result set to which JOIN condition ON t1.c1 = t2.c1 is applied → Apply WHERE condition t1.c2 = t2.c2

```
( 1,  1, null, null )
  ( 2,  2,    2,    2 )                    ( 2,  2,    2,    2 )
  ( 3,  3,    3,    3 )          →         ( 3,  3,    3,    3 )
  ( 4,  4, null, null )
  ( 5,  5, null, null )
```

Left outer join combines right rows satisfying the join condition for the left rows, then returns the combined rows as a result. If right rows satisfying the join condition does not exist, then it returns the result whose left row values are as they are and whose right row values are filled with NULL.

LEFT OUTER JOIN

```
t1 ( 1, 1 ), ( 2, 2 )
t2 ( 2, 2 ), ( 3, 3 )
gSQL> SELECT * FROM t1 LEFT OUTER JOIN t2 ON t1.c1 = t2.c1;
C1 C2   C1   C2
-- -- ---- ----
 1  1 null null
 2  2    2    2
2 rows selected.
```

Right outer join is operated in an opposite way of left outer join.

```
RIGHT OUTER JOIN
t1 ( 1, 1 ), ( 2, 2 )
t2 ( 2, 2 ), ( 3, 3 )
gSQL> SELECT * FROM t1 RIGHT OUTER JOIN t2 ON t1.c1 = t2.c1;
  C1   C2 C1 C2
---- ---- -- --
   2    2  2  2
null null  3  3
2 rows selected.
```

Full outer join returns the left rows filled with NULL for all right rows which do not satisfy the join conditi on together with left outer join results.

```
FULL OUTER JOIN
t1 ( 1, 1 ), ( 2, 2 )
t2 ( 2, 2 ), ( 3, 3 )
gSQL> SELECT * FROM t1 FULL OUTER JOIN t2 ON t1.c1 = t2.c1;
  C1   C2   C1   C2
---- ---- ---- ----
   1    1 null null
   2    2    2    2
null null    3    3
3 rows selected.
```

〈natural join〉

〈natural join〉 joins all columns with same names in two tables participating in join as equal. In other wor ds, it is as same as specifying all columns with same names of two tables participating in join in USING cla use of inner join.

```
t1 ( C1 INTEGER, C2 INTEGER )
t2 ( C1 INTEGER, C3 INTEGER )
t1 ( 1, 10 ), ( 2, 20 ), ( 3, 30 )
t2 ( 1, 100 ), ( 2, 200 ), ( 3, 300 )
gSQL> SELECT * FROM t1 NATURAL JOIN t2;
C1 C2  C3
-- -- ---
 1 10 100
 2 20 200
 3 30 300
3 rows selected.
gSQL> SELECT * FROM t1 INNER JOIN t2 USING ( c1 );
```

```
C1 C2  C3
-- -- ---
 1 10 100
 2 20 200
 3 30 300
3 rows selected.
```

〈join specification〉

It specifies the join condition.
〈join condition〉 specifies the condition for joining left rows and right rows of a join statement.
〈named columns join〉 specifies the join condition by listing that 〈column name〉, if the same 〈column name〉 exist in left rows and right rows.

```
t1 ( C1 INTEGER, C2 INTEGER )
t2 ( C1 INTEGER, C3 INTEGER )
t1 ( 1, 10 ), ( 2, 20 ), ( 3, 30 )
t2 ( 1, 100 ), ( 2, 200 ), ( 3, 300 )
• 〈join condition〉
gSQL> SELECT * FROM t1 INNER JOIN t2 ON t1.c1 = t2.c1;
C1 C2 C1  C3
-- -- -- ---
 1 10  1 100
 2 20  2 200
 3 30  3 300
3 rows selected.
• 〈named columns join〉
gSQL> SELECT * FROM t1 INNER JOIN t2 USING ( c1 );
C1 C2  C3
-- -- ---
 1 10 100
 2 20 200
 3 30 300
3 rows selected.
```

Examples

The following is an example of SELECT statement which uses 〈cross join〉.

```
gSQL> SELECT c_name, o_totalprice FROM customer CROSS JOIN orders;
C_NAME     O_TOTALPRICE
```

```
---------- ------------
Customer#1    173665.47
Customer#1     46929.18
Customer#1    193846.25
Customer#1     32151.78
Customer#1     144659.2
Customer#2    173665.47
Customer#2     46929.18
Customer#2    193846.25
Customer#2     32151.78
Customer#2     144659.2
Customer#3    173665.47
Customer#3     46929.18
Customer#3    193846.25
Customer#3     32151.78
Customer#3     144659.2
Customer#4    173665.47
Customer#4     46929.18
Customer#4    193846.25
Customer#4     32151.78
Customer#4     144659.2
C_NAME     O_TOTALPRICE
---------- ------------
Customer#5    173665.47
Customer#5     46929.18
Customer#5    193846.25
Customer#5     32151.78
Customer#5     144659.2
25 rows selected.
```

The following is an example of SELECT statement which uses inner join.

```
gSQL> SELECT c_name, o_totalprice FROM customer INNER JOIN orders ON c_custkey = o_custkey;
C_NAME     O_TOTALPRICE
---------- ------------
Customer#1    173665.47
Customer#2     46929.18
Customer#4    193846.25
Customer#3     32151.78
Customer#5     144659.2
5 rows selected.
```

The following is an example of SELECT statement which uses outer join.

```
gSQL> SELECT c_name, o_totalprice FROM customer LEFT OUTER JOIN orders ON c_custkey =
o_custkey AND o_orderdate < '1996-01-01';
C_NAME     O_TOTALPRICE
---------- ------------
Customer#1         null
Customer#2         null
Customer#3     32151.78
Customer#4    193846.25
Customer#5     144659.2
5 rows selected.
gSQL> SELECT c_name, o_totalprice FROM customer RIGHT OUTER JOIN orders ON c_custkey =
o_custkey AND c_nation = 'KOREA';
C_NAME     O_TOTALPRICE
---------- ------------
Customer#1    173665.47
null           46929.18
null          193846.25
Customer#3     32151.78
null           144659.2
5 rows selected.
gSQL> SELECT c_name, o_totalprice FROM customer FULL OUTER JOIN orders ON c_custkey =
o_custkey AND c_nation = 'KOREA' AND o_orderdate < '1996-01-01';
C_NAME     O_TOTALPRICE
---------- ------------
Customer#1         null
Customer#2         null
Customer#3     32151.78
Customer#4         null
Customer#5         null
null          173665.47
null           46929.18
null          193846.25
null           144659.2
9 rows selected.
```

The following is an example of SELECT statement which uses natural join.

```
gSQL> SELECT c_name, o_totalprice FROM (SELECT c_custkey custkey, c_name FROM customer)
NATURAL JOIN (SELECT o_custkey custkey, o_totalprice FROM orders);
```

```
C_NAME     O_TOTALPRICE
---------- ------------
Customer#1    173665.47
Customer#2     46929.18
Customer#4    193846.25
Customer#3     32151.78
Customer#5     144659.2
5 rows selected.
```

Compatibility

Table 20-11 SQL standard compatibility

Feature ID	Description	Compatibility
F401	Extended joined table	O
F402	Named column joins for LOBs, arrays, and multisets	X
F403	Partitioned join tables	X

For More Information

Refer to **from clause**.

where clause

Function

It applies <search condition> to the result of <from clause>.

Syntax

```
<where clause> ::=
    WHERE <search condition>
```

Syntax Rules and Parameters

<where clause>

<search condition> which returns a boolean type is required after WHERE keyword.

Description

For more information about <where clause>, refer to **Conditions**.

Example

The following is an example of SELECT statement which uses <where clause>.

```
gSQL> SELECT s_name, s_nation FROM supplier WHERE s_nation = 'KOREA';
S_NAME                    S_NATION
------------------------- --------
Supplier#2                KOREA
1 row selected.
gSQL> SELECT s_name, ps_availqty, ps_supplycost FROM supplier, partsupp WHERE s_nation =
'KOREA' AND s_suppkey = ps_suppkey;
S_NAME                    PS_AVAILQTY PS_SUPPLYCOST
------------------------- ----------- -------------
Supplier#2                       8076        993.49
Supplier#2                       4069        357.84
2 rows selected.
```

Compatibility

Table 20-12 SQL standard compatibility

Feature ID	Description	Compatibility
F441	Extended set function support	O

For More Information

Refer to **query specification**.

hierarchical query clause

Function

It specifies to retrieve the hierarchical model data in a hierarchy.
It returns table records in a hierarchy of depth-first sequence by using the launch condition and sub-connectivity condition.

Syntax

```
<hierarchical query clause> ::=
    <start with connect by clause> [ <order siblings by clause> ]
<start with connect by clause> ::=
    <start with clause> <connect by clause>
    | <connect by clause> <start with clause>
    | <connect by clause>
<start with clause> ::=
    START WITH <start_with_condition>
<connect by clause> ::=
    CONNECT BY [NOCYCLE] <connect_by_condition>
<order siblings by clause> ::=
    ORDER SIBLINGS BY <ordering element> [ { <comma> <ordering element> }... ]
<ordering element> ::=
    <value expression> [ASC | DESC] [NULLS FIRST | NULLS LAST]
<hierarchy expression> ::=
    LEVEL
    | CONNECT_BY_ISCYCLE
    | CONNECT_BY_ISLEAF
    | PRIOR <value expression>
    | CONNECT_BY_ROOT <value expression>
    | SYS_CONNECT_BY_PATH <left paren> <value expression> <comma> <character string literal>
<right paren>
```

Invocation and Access Rules

It is supported in <query specification> statement, and the user should satisfy the access privilege of <query specification> to perform it.
For more information, refer to **query specification**.

Syntax Rules and Parameters

〈hierarchical query clause〉

〈connect by clause〉 should be described.
〈start with clause〉 or 〈order siblings by clause〉 is described when it is required.

〈start with clause〉

It specifies the condition of root record in data hierarchy.
When it is not specified, then all records of from clause become targets of root records.
It can be described only once within SELECT statement.

〈connect by clause〉

It describes the relation between the parent record and the child record.
It expresses the relation between the parent record and the child record by using PRIOR operator which r epresenting the column value of the parent record.
If the condition of connecting the parent record and the child record is not specified by using PRIOR oper ator, then an infinite loop may occur.
It can be described only once within SELECT statement.

- NOCYCLE
  - When NOCYCLE option is not specified
    - When cycle occurs, then that query causes an error and the execution stops.
  - When NOCYCLE option is specified
    - When cycle occurs, then that query does not cause an error.
    - The record which caused cycle stops retrieving the child record and it is not included in the re sult record.
    - The query continuously proceeds for sibling rows where cycle does not occur.
    - 1 is stored in CONNECT_BY_ISCYCLE of the parent record of the record which caused cycle.
    - 0 is stored in CONNECT_BY_ISCYCLE of the parent record of the record which has not cause d cycle.

〈order siblings by clause〉

It specifies the order of fetching sibling records of the same parent records.

- Sort order
  - ASC
  - DESC
  - If not specified, the default value is ASC.

- Null ordering
  - NULLS FIRST
  - NULLS LAST
  - If not specified, the default value is NULLS LAST.

〈hierarchy expression〉

- The following hierarchy information is acquired when configuring the hierarchy query.
  - LEVEL
  - CONNECT_BY_ISCYCLE
  - CONNECT_BY_ISLEAF
  - PRIOR
  - CONNECT_BY_ROOT
  - SYS_CONNECT_BY_PATH

Table 20-13 Result type of 〈hierarchy expression〉

Expression	Result DataType
LEVEL	NATIVE_BIGINT
CONNECT_BY_ISCYCLE	NATIVE_BIGINT
CONNECT_BY_ISLEAF	NATIVE_BIGINT
PRIOR expr	expr의 DataType
CONNECT_BY_ROOT expr	expr의 DataType
SYS_CONNECT_BY_PATH(expr, literal)	VARCHAR(4000 characters)

〈hierarchy expression〉 can be described in the following statements.

Expression₩clause	FROM	START WITH	CONNECT BY	ORDER SIBLINGS BY	WHERE/ GROUP BY/ HAVING	ORDER BY/ SELECT TARGET
LEVEL	X	O	O	X	O	O
CONNECT_BY_ISCYCLE	X	X	X	X	O	O
CONNECT_BY_ISLEAF	X	X	X	X	O	O
PRIOR	X	X	O	X	O	O
CONNECT_BY_ROOT	X	X	X	X	O	O
SYS_CONNECT_BY_PATH	X	X	X	X	O	O

Whether 〈hierarchy expression〉 can be used as an argument of 〈hierarchy expression〉 is described in the following table.

Expression₩Argument(expr)	LEVEL	CONNECT_BY_ISCYCLE	CONNECT_BY_ISLEAF	PRIOR	CONNECT_BY_ROOT	SYS_CONNECT_BY_PARTH
PRIOR expr	X	X	X	X	X	X
CONNECT_BY_ROOT expr	X	X	X	X	X	X
SYS_CONNECT_BY_PARTH (expr,literal)	O	O	O	O	O	O

Description

〈hierarchical query clause〉 retrieves the hierarchical model data in a hierarchy.
It returns table records in a hierarchy of depth-first sequence by using the launch condition and sub-connectivity condition.

When 〈hierarchical query clause〉 is described in SELECT, then it is processed in the following order.

1. ON condition in FROM clause
2. START WITH
3. CONNECT BY
4. WHERE

- When only a single table exists in *from* clause

```
SELECT *
 FROM r_region
WHERE r_population > 10000000            ❸ Condition in WHERE clause
START WITH r_name = 'EARTH'              ❶ START WITH
CONNECT BY r_domain = PRIOR r_name       ❷ CONNECT BY
```

- When join condition configured with multiple tables exists in *from* clause
    - When join condition is described in ON clause of FROM clause

```
SELECT *
  FROM r_region INNER JOIN s_region
       ON r_id = s_id                     ❶ Join condition in ON clause
 WHERE r_population > 10000000            ❹ Condition in WHERE clause
START WITH r_name = 'EARTH'              ❷ START WITH
CONNECT BY r_domain = PRIOR r_name       ❸ CONNECT BY
```

    - When join condition is described in WHERE clause

```
SELECT *
  FROM r_region, s_region
 WHERE r_population > 10000000          ❸ Condition in WHERE clause
   AND r_id = s_id                      ❸ Join condition in WHERE clause
START WITH r_name = 'EARTH'             ❶ START WITH
CONNECT BY r_domain = PRIOR r_name      ❷ CONNECT BY
```

- When join condition is described in both ON clause of FROM and in WHERE clause

```
SELECT *
  FROM r_region INNER JOIN s_region
       ON r_name = s_name               ❶ Join condition in ON clause
 WHERE r_population > 10000000          ❹ Condition in WHERE clause
   AND r_id = s_id                      ❹ Join condition in WHERE clause
START WITH r_name = 'EARTH'             ❷ START WITH
CONNECT BY r_domain = PRIOR r_name      ❸ CONNECT BY
```

〈order siblings by clause〉

It specifies the order of fetching sibling records of the same parent records within 〈hierarchical query clause〉.
〈order siblings by clause〉 is a statement distinct from 〈order by clause〉.

```
gSQL>
SELECT * FROM t1;
I1  I2
--- ----
A   null
AA  A
AB  A
fAA AA
eAA AA
bAA AA
dAB AB
cAB AB
aAB AB
9 rows selected.
```

- The following is an example of defining the order of fetching sibling records of the same parent records.

```
gSQL>
SELECT LEVEL, i1, i2
  FROM t1
START WITH i1 = 'A'
CONNECT BY i2 = PRIOR i1
ORDER SIBLINGS BY i1;
LEVEL I1  I2
----- --- ----
    1 A   null
    2 AA  A
    3 bAA AA
    3 eAA AA
    3 fAA AA
    2 AB  A
    3 aAB AB
    3 cAB AB
    3 dAB AB
9 rows selected.
```

- The following is an example of sorting all results retrieved in a hierarchy by using ORDER BY claus e in an order of LEVEL.

```
gSQL>
SELECT LEVEL, i1, i2
  FROM t1
START WITH i1 = 'A'
CONNECT BY i2 = PRIOR i1
ORDER SIBLINGS BY i1
ORDER BY LEVEL;
LEVEL I1  I2
----- --- ----
    1 A   null
    2 AA  A
    2 AB  A
    3 bAA AA
    3 eAA AA
    3 fAA AA
    3 aAB AB
    3 cAB AB
    3 dAB AB
9 rows selected.
```

〈hierarchy expression〉

The followings are features of the hierarchy expression.

- PRIOR
  - It acquires the information based on the parent record of the current record.
  - PRIOR is a unary operator, and its priority is as same as that of a unary operator +,-.

```
gSQL>
SELECT i1, i2
  FROM t1
START WITH i1 = 'X'
CONNECT BY i2 = PRIOR i1;
I1    I2
----- ----
X     null
XA    X
XXA   XA
XXXA  XXA
XXXXA XXXA
5 rows selected.
```

- LEVEL
  - It is the hierarchical value in which the record belongs.
  - LEVEL of the root record is 1, and LEVEL of the child record of the root is 2.
  - LEVEL increases by 1 as it goes down to the child record.

```
gSQL>
SELECT LEVEL, i1, i2
  FROM t1
START WITH i1 = 'X'
CONNECT BY i2 = prior i1;
LEVEL I1    I2
----- ----- ----
    1 X     null
    2 XA    X
    3 XXA   XA
    4 XXXA  XXA
    5 XXXXA XXXA
5 rows selected.
```

- CONNECT_BY_ISCYCLE

- It acquires the information about whether the record causing cycle exists among child records related to the current record.
- It is available only when NOCYCLE exists in CONNECT BY statement.

```
gSQL>
SELECT * FROM t1;
I1 I2
-- ----
A  null
AA A
AB A
AC A
AA AA
AB AA
6 rows selected.
gSQL>
SELECT i1, i2, CONNECT_BY_ISCYCLE
  FROM t1
START WITH i1 = 'A'
CONNECT BY NOCYCLE i2 = prior i1;
I1 I2   CONNECT_BY_ISCYCLE
-- ---- ------------------
A  null                 0
AA A                    1
AB AA                   0
AB A                    0
AC A                    0
5 rows selected.
```

- CONNECT_BY_ISLEAF
  - It acquires the information about whether the child record related to the current record exists.
  - If the child record related to the current record does not exist, then it returns 1.

```
gSQL>
SELECT i1, i2, CONNECT_BY_ISLEAF
  FROM t1
START WITH i1 = 'X'
CONNECT BY i2 = prior i1;
I1    I2   CONNECT_BY_ISLEAF
----- ---- -----------------
X     null                0
XA    X                   0
```

```
XXA   XA                  0
XXXA  XXA                 0
XXXXA XXXA                1
5 rows selected.
```

- CONNECT_BY_ROOT
  - It acquires the information based on the root record of the current record.

```
gSQL>
SELECT i1, i2, CONNECT_BY_ROOT i1
  FROM t1
START WITH i1 = 'X'
CONNECT BY i2 = prior i1;
I1    I2   CONNECT_BY_ROOT I1
----- ---- ------------------
X     null X
XA    X    X
XXA   XA   X
XXXA  XXA  X
XXXXA XXXA X
5 rows selected.
```

- SYS_CONNECT_BY_PATH
  - It acquires the information by recursively searching along the parent record of the current record.

```
gSQL>
SELECT i1, i2, SYS_CONNECT_BY_PATH( i1, '/' )
  FROM t1
START WITH i1 = 'X'
CONNECT BY i2 = prior i1;
I1    I2   SYS_CONNECT_BY_PATH( I1, '/' )
----- ---- ------------------------------
X     null /X
XA    X    /X/XA
XXA   XA   /X/XA/XXA
XXXA  XXA  /X/XA/XXA/XXXA
XXXXA XXXA /X/XA/XXA/XXXA/XXXXA
5 rows selected.
```

Examples

The following is the result of retrieving record in emp table which will be used in the example of hierarchi cal query clause.

```
gSQL>
SELECT * FROM emp;
NAME    MGR
------- -------
Kelly   null
Bill    Kelly
Jackson Kelly
Joe     Kelly
Scott   Bill
Larry   Bill
Paul    Jackson
Bill    Bill
8 rows selected.
```

The following is an example of when cycle occurs.

```
gSQL>
SELECT *
  FROM emp
START WITH mgr IS NULL
CONNECT BY mgr = PRIOR name
ORDER SIBLINGS BY name;
ERR-42000(16511): cycle detected while executing recursive WITH query
```

The following is an example of executing the query by using CONNECT BY NOCYCLE statement.

```
gSQL>
SELECT *
  FROM emp
START WITH mgr IS NULL
CONNECT BY NOCYCLE mgr = PRIOR name
ORDER SIBLINGS BY name;
NAME    MGR
------- -------
Kelly   null
Bill    Kelly
Larry   Bill
```

```
Scott   Bill
Jackson Kelly
Paul    Jackson
Joe     Kelly
7 rows selected.
```

The following is an example of retrieving the information about the hierarchical data by using the hierarchy expression.

```
gSQL>
SELECT name,
       mgr,
       PRIOR name AS prior_mgr,
       LEVEL,
       CONNECT_BY_ISCYCLE AS iscycle,
       CONNECT_BY_ISLEAF AS isleaf,
       CONNECT_BY_ROOT mgr AS root_mgr,
       SYS_CONNECT_BY_PATH( mgr, '/' ) AS path
  FROM emp
START WITH mgr IS NULL
CONNECT BY NOCYCLE mgr = PRIOR name
ORDER SIBLINGS BY name;
NAME    MGR     PRIOR_MGR LEVEL ISCYCLE ISLEAF ROOT_MGR PATH
------- ------- --------- ----- ------- ------ -------- ---------------
Kelly   null    null          1       0      0 null     /
Bill    Kelly   Kelly         2       1      0 null     //Kelly
Larry   Bill    Bill          3       0      1 null     //Kelly/Bill
Scott   Bill    Bill          3       0      1 null     //Kelly/Bill
Jackson Kelly   Kelly         2       0      0 null     //Kelly
Paul    Jackson Jackson       3       0      1 null     //Kelly/Jackson
Joe     Kelly   Kelly         2       0      1 null     //Kelly
7 rows selected.
```

group by clause

Function

It specifies the grouped table of which <group by clause> was applied to the result processed by the previous statements.

Syntax

```
<group by clause> ::=
    GROUP BY <grouping element list>
<grouping element list> ::=
    <grouping element> [ { , <grouping element> } ... ]
<grouping element> ::=
      <ordinary grouping set>
    ¦ <empty grouping set>
<ordinary grouping set> ::=
      <grouping column reference>
<grouping column reference> ::=
    <column reference>
    ¦ <value_expression>
<empty grouping set> ::=
    <left paren> <right paren>
```

Invocation and Access Rules

Any separate access privilege is not required for a user to perform <group by clause>.

Syntax Rules and Parameters

<ordinary grouping set>

It consists of one or more <grouping column reference>.
It does not support LONG type (LONG VARCHAR, LONG VARBINARY).

- SELECT c1, sum(c2) FROM t1 GROUP BY c1;
- SELECT sum(c1) FROM t1 GROUP BY NULL;

〈empty grouping set〉

It can be specified by using only parentheses.

- SELECT sum(c1) FROM t1 GROUP BY ();

Description

〈grouping element list〉

It groups 〈grouping element list〉 specified in 〈group by clause〉 into a GROUPING SET. If all values of 〈grouping element〉 in GROUPING SET are matched, it is processed as the same group.

- If 〈group by clause〉 is specified, the following expressions can appear in 〈select list〉.
  - Constant number
  - 〈grouping column reference〉 specified in 〈group by clause〉
  - Operation expression including 〈grouping column reference〉 specified in 〈group by clause〉
  - Aggregation function of a column which is not specified in 〈group by clause〉
  - SELECT c1, sum(c2) FROM t1 GROUP BY c1;

〈grouping column reference〉

〈column reference〉 or 〈value expression〉 can appear in 〈grouping column reference〉.

- 〈column reference〉
  - Only the columns belonging to 〈from clause〉 of 〈query specification〉 can be referenced.
    - SELECT c1 FROM t1 GROUP BY c1;
  - If same column names exist, then clearly specify the column name by using a table name.
    - SELECT t1.c1, t2.c1 FROM t1, t2 GROUP BY t1.c1, t2.c1;

- 〈value expression〉
  - It is an expression which includes 〈column reference〉.
    - It can be divided into several groups by using 〈column reference〉
    - SELECT sum(c2) FROM t1 GROUP BY c1 + 10;
  - It is an expression which does not include 〈column reference〉.
    - Values in 〈value expression〉 are all same constants, so all records are configured into a single group.
    - If null is specified in 〈value expression〉, the null values are treated as the same value, so all records are configured into a single group.
    - SELECT sum(c1), sum(c2) FROM t1 GROUP BY NULL;

〈empty grouping set〉

All records in 〈empty grouping set〉 are configured into a single group.

- SELECT sum(c1), sum(c2) FROM t1 GROUP BY ();

Example

The following is an example of SELECT statement which uses GROUP BY clause.

```
gSQL> SELECT c_nation, COUNT(c_name) FROM customer GROUP BY c_nation;
C_NATION      COUNT(C_NAME)
------------- -------------
UNITED STATES             1
CANADA                    1
KOREA                     2
GERMANY                   1
4 rows selected.
gSQL> SELECT COUNT(c_name) FROM customer GROUP BY NULL;
COUNT(C_NAME)
-------------
            5
1 row selected.
gSQL> SELECT COUNT(c_name) FROM customer GROUP BY ();
COUNT(C_NAME)
-------------
            5
1 row selected.
```

Compatibility

Table 20-14 SQL standard compatibility

Feature ID	Description	Compatibility
T431	Extended grouping capabilities	X
T432	Nested and concatenated GROUPING SETS	X
T434	GROUP BY DISTINCT	X

For More Information

Refer to the followings.

- having clause
- query specification

having clause

Function

It specifies grouped tables having removed groups which do not satisfy <search condition>.

Syntax

```
<having clause> ::=
    HAVING <search condition>
```

Invocation and Access Rules

Any separate access privilege is not required for a user to perform <having clause>.

Syntax Rules and Parameters

<having clause>

- What can be used without aggregate functions in <search condition> is only <grouping column reference> specified in <group by clause>.
  - SELECT c1, sum(c2) FROM t1 GROUP BY c1 HAVING c1 > 3;
- The columns which are not specified in <group by clause> can be specified by using aggregate functions.
  - SELECT c1, sum(c2) FROM t1 GROUP BY c1 HAVING sum(c2) > 100;

Description

<having clause>

<having clause> specifies search conditions for the grouped data.

Generally, it is used together with <group by clause>. When <having clause> is used without <group by clause>, it is considered as if <empty grouping set> exists.

- SELECT sum(c1), sum(c2) FROM t1 HAVING sum(c1) > 0;
- <=> SELECT sum(c1), sum(c2) FROM t1 GROUP BY () HAVING sum(c1) > 0;

<grouping column reference> specified in <group by clause> can be specified in <having clause>.
The columns which are not specified in <group by clause> can be specified by using aggregate functions.

- SELECT c1, sum(c2) FROM t1 GROUP BY c1 HAVING c1 > 3 AND sum(c2) > 100;

Example

The following is an example of SELECT statement which uses <having clause>.

```
gSQL> SELECT c_nation, COUNT(c_name) FROM customer GROUP BY c_nation HAVING COUNT(c_name) > 1;
C_NATION COUNT(C_NAME)
-------- -------------
KOREA                2
1 row selected.
gSQL> SELECT COUNT(c_name) FROM customer HAVING COUNT(c_name) > 1;
COUNT(C_NAME)
-------------
            5
1 row selected.
```

Compatibility

Table 20-15 SQL standard compatibility

Feature ID	Description	Compatibility
T301	Functional dependencies	O

For More Information

Refer to the followings.

- **group by clause**
- **Conditions**

window clause

Function

It defines the execution range of the window function described in select list and order by clause.

Syntax

```
<window clause> ::=
     WINDOW <window definition list>
<window definition list> ::=
     <window definition> [ { <comma> <window definition> }... ]
<window definition> ::=
     <new window name> AS <window specification>
<new window name> ::=
     <window name>
<window specification> ::=
     <left paren> <window specification details> <right paren>
<window specification details> ::=
     [ <existing window name> ]
          [ <window partition clause> ]
          [ <window order clause> ]
          [ <window frame clause> ]
<existing window name> ::=
     <window name>
<window partition clause> ::=
     PARTITION BY <window partition column reference list>
<window partition column reference list> ::=
     <window partition column reference>
          [ { <comma> <window partition column reference> }... ]
<window partition column reference> ::=
     <column reference>
<window order clause> ::=
     ORDER BY <sort specification list>
<sort specification list> ::=
     <sort specification> [ { <comma> <sort specification> }... ]
<sort specification> ::=
     <sort key> [ <ordering specification> ] [ <null ordering> ]
<sort key> ::=
     <value expression>
```

```
<ordering specification> ::=
       ASC
     | DESC
<null ordering> ::=
      NULLS FIRST
    | NULLS LAST
<window frame clause> ::=
     <window frame units> <window frame extent>
          [ <window frame exclusion> ]
<window frame units> ::=
       ROWS
     | RANGE
     | GROUPS
<window frame extent> ::=
       <window frame start>
     | <window frame between>
<window frame start> ::=
       UNBOUNDED PRECEDING
     | <window frame preceding>
     | CURRENT ROW
<window frame preceding> ::=
     <unsigned value specification> PRECEDING
<window frame between> ::=
     BETWEEN <window frame bound 1> AND <window frame bound 2>
<window frame bound 1> ::=
     <window frame bound>
<window frame bound 2> ::=
     <window frame bound>
<window frame bound> ::=
       <window frame start>
     | UNBOUNDED FOLLOWING
     | <window frame following>
<window frame following> ::=
     <unsigned value specification> FOLLOWING
<window frame exclusion> ::=
       EXCLUDE CURRENT ROW
     | EXCLUDE GROUP
     | EXCLUDE TIES
     | EXCLUDE NO OTHERS
```

Invocation and Access Rules

The access privilege for a column is required if the column exist in a window clause.

Syntax Rules and Parameters

〈window clause〉

The window function is not allowed in a window clause.

〈window definition list〉

It can define multiple 〈window definition〉.

〈window definition〉

It describes the execution range of window function with 〈new window name〉.

〈new window name〉 should not be a duplicate in 〈window clause〉.
〈new window name〉 can be referred in *over* clause in the window function.

```
SELECT SUM(i2) OVER w1
  FROM t1
WINDOW w1 AS ( PARTITION BY i1 ORDER BY i2 );
```

〈window specification〉

It defines the execution range of window function.

It can redefine 〈window specification〉 by referring to 〈existing window name〉 and adding it to the pred efined information.
〈existing window name〉 can refer to 〈new window name〉 which was predefined in 〈window definition list〉 only.

```
• Referred in window clause
SELECT SUM(i2) OVER w2
  FROM t1
WINDOW w1 AS ( PARTITION BY i1
               ORDER BY i2 ),
       w2 AS ( w1 ROWS BETWEEN UNBOUNDED PRECEDING  <---
                          AND CURRENT ROW );
• Referred in over clause of window function
SELECT SUM(i2) OVER ( w1 ROWS BETWEEN UNBOUNDED PRECEDING  <---
```

```
                                    AND CURRENT ROW )
  FROM t1
WINDOW w1 AS ( PARTITION BY i1
               ORDER BY i2 );
```

When redefining <window specification>by referring to <existing window name>

```
• <window partition clause> is not allowed in the redefinition.
SELECT SUM(i2) OVER ( w1 PARTITION BY i1 )  <--- ( X )
  FROM t1
WINDOW w1 AS ( );
• If order by clause is described in <existing window name>,
  order by clause is not allowed in the redefinition.
SELECT SUM(i2) OVER ( w1 ORDER BY i3 ) <--- ( X )
  FROM t1
WINDOW w1 AS ( PARTITION BY i1
               ORDER BY i2 );
• window frame clause is not allowed in <existing window name>.
SELECT SUM(i2) OVER ( w1 )
  FROM t1
WINDOW w1 AS ( PARTITION BY i1
               ORDER BY i2
               ROWS BETWEEN UNBOUNDED PRECEDING  <--- ( X )
                        AND CURRENT ROW );
```

<window frame start>

If frame end is omitted, then <window frame start> is as same as <window frame start> AND CURRENT R OW.

- UNBOUNDED PRECEDING
  - UNBOUNDED PRECEDING AND CURRENT ROW
- *offset* PRECEDING
  - *offset* PRECEDING AND CURRENT ROW
    - 3 PRECEDING AND CURRENT ROW
- CURRENT ROW
  - CURRENT ROW AND CURRENT ROW

<window frame between>

- UNBOUNDED FOLLOWING is not allowed in frame start.
  - BETWEEN *UNBOUNDED FOLLOWING* AND UNBOUNDED FOLLOWING (X)

- UNBOUNDED PRECEDING is not allowed in frame end.
  - BETWEEN UNBOUNDED PRECEDING AND *UNBOUNDED PRECEDING* (X)
- If frame start is CURRENT ROW, then <window frame preceding> is not allowed in frame end.
  - BETWEEN CURRENT ROW AND *1 PRECEDING* (X)
- If frame start is <window frame following>, then neither <window frame preceding> nor CURRENT ROW is allowed in frame end.
  - BETWEEN 3 FOLLOWING AND *1 PRECEDING* (X)
  - BETWEEN 3 FOLLOWING AND *CURRENT ROW* (X)

<window frame following> / <window frame preceding>

offset PRECEDING / *offset* FOLLOWING

- Neither a negative number nor NULL is allowed in offset.
  - *NULL* PRECEDING (X)
  - *-1* PRECEDING (X)
  - *NULL* FOLLOWING (X)
  - *-1* FOLLOWING (X)

- When frame unit is RANGE
  - If the sort key of window ORDER BY is a numeric type, then describe a numeric type in offset.
    - ORDER BY orderkey RANGE BETWEEN 3 PRECEDING AND 5 FOLLOWING
  - If the sort key of window ORDER BY is a datetime type or an interval type, then describe an interval type in offset.
    - ORDER BY orderdate RANGE BETWEEN INTERVAL'3'DAY PRECEDING AND INTERVAL'5'day FOLLOWING
- When frame unit is ROWS/GROUPS, then describe an integer type number in offset.
  - ORDER BY orderkey ROWS BETWEEN 3 PRECEDING AND 5 FOLLOWING
  - ORDER BY orderkey GROUPS BETWEEN 3 PRECEDING AND 5 FOLLOWING

Description

WINDOW clause describes the execution range of window function described in select list and order by clause.

<window partition clause> divides groups.
<window order clause> sorts the records in the group.
<window frame clause> defines the range of window function's target record for the sorted records in the group.

WINDOW clause is executed for the result sets after FROM, WHERE, GROUP BY, HAVING clauses are executed.
When using an aggregate, GROUP BY, HAVING clauses in a query, then describe the group column inste

ad of the column from the original table in WINDOW clause.

〈window specification〉

It describes the execution range of window function for each record.

It can redefine 〈window specification〉 by referring to 〈existing window name〉 and adding it to the pred efined information.

```
SELECT SUM(i2) OVER ( w1
                      ORDER BY i2
                      ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW )
  FROM t1
WINDOW w1 AS ( PARTITION BY i1 ),
       w2 AS ( w1 ORDER BY i3 );
   → It is the same statement.
SELECT SUM(i2) OVER ( PARTITION BY i1
                      ORDER BY i2
                      ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW )
  FROM t1
WINDOW w1 AS ( PARTITION BY i1 ),
       w2 AS ( PARTITION BY i1
               ORDER BY i3 );
```

When referring to 〈window name〉 wname in window function OVER clause, then OVER wname and OV ER (wname) is not same.

- OVER wname

```
* Referring to 〈window specification〉 information defined with wname
ex) Referring to w1
SELECT SUM(i2) OVER w1
  FROM t1
WINDOW w1 AS ( PARTITION BY i1
               ORDER BY i2
               ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW );
```

- OVER(wname)

```
* It redefines 〈window specification〉 by adding it to the predefined information,
  and it can not define 〈window frame clause〉 in the existing information.
ex) Referring to w1
```

```
SELECT SUM(i2) OVER ( w1 ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW )
  FROM t1
WINDOW w1 AS ( PARTITION BY i1
               ORDER BY i2 );
ex) Referring to w1 ( Error : It can not define <window frame clause> in the existing
information. )
SELECT SUM(i2) OVER ( w1 )
  FROM t1
WINDOW w1 AS ( PARTITION BY i1
               ORDER BY i2
               ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW );  <---
```

<window partition clause>

It divides the query result set into groups based on <window partition column reference list> by using PA RTITION BY.
If this clause is omitted, then the function processes all rows in the result set as a single group.

- When <window partition clause> is described

```
gSQL>
SELECT orderdate,
       orderkey,
       totalprice,
       SUM( totalprice ) OVER( PARTITION BY orderdate ) AS SUM_OVER_RESULT
FROM orders;
ORDERDATE  ORDERKEY TOTALPRICE SUM_OVER_RESULT
---------- -------- ---------- ---------------
1998-07-24     1730     204656          520630
1998-07-24    17056     289620          520630
1998-07-24    19937      26354          520630      partition ❶
-------------------------------------------------------------------------
1998-07-25     2400     150304          368523
1998-07-25    11204      27165          368523
1998-07-25    11938     191054          368523      partition ❷
-------------------------------------------------------------------------
1998-07-26    35655      13698          362691
1998-07-26    53377     185930          362691
1998-07-26    55010     163063          362691      partition ❸
-------------------------------------------------------------------------
9 rows selected.
```

- When <window partition clause> is omitted

```
gSQL>
SELECT orderdate,
       orderkey,
       totalprice,
       SUM( totalprice ) OVER() AS SUM_OVER_RESULT
  FROM orders;
ORDERDATE  ORDERKEY TOTALPRICE SUM_OVER_RESULT
---------- -------- ---------- ---------------
1998-07-24     1730     204656         1251844
1998-07-24    17056     289620         1251844
1998-07-24    19937      26354         1251844
1998-07-25     2400     150304         1251844
1998-07-25    11204      27165         1251844
1998-07-25    11938     191054         1251844
1998-07-26    35655      13698         1251844
1998-07-26    53377     185930         1251844
1998-07-26    55010     163063         1251844      partition ❶
--------------------------------------------------------------------------
9 rows selected.
```

<window order clause>

It specifies the method of sorting the data in the partition based on <sort specification list> by using ORD ER BY.

- <ordering specification>
  - It specifies whether to sort in ascending order or in descending order.
    - ASC
    - DESC
    - If it is omitted, the default value is ASC.

- <null ordering>
  - It specifies an order between the NULL values and non-NULL values.
    - NULLS FIRST
    - NULLS LAST
    - If it is omitted, the default value is NULLS LAST.

- When <window frame units> of <window frame clause> is RANGE,
  - If offset PRECEDING or offset FOLLOWING is described, then it can specify only a single sort key.
    - ORDER BY I1 RANGE 3 PRECEDING

        - ORDER BY I1 RANGE BETWEEN CURRENT ROW AND 3 FOLLOWING
    - Otherwise, it can specify multiple sort keys.
        - ORDER BY I1, I2 RANGE UNBOUNDED PRECEDING
        - ORDER BY I1, I2 RANGE CURRENT ROW
        - ORDER BY I1, i2 RANGE BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING
        - ORDER BY I1, i2 RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWIN G

〈window frame clause〉

It specifies window frame which is the record range of window function's target.

window frame is the record range related to each row (current row) of the query.
The target of window frame is records sorted in the current partition.
window frame can specify the scope (ROWS/ RANGE/ GROUPS), starting point and ending point, and ex cluded records.

If it is omitted, then it applies RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW.

- RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
    - Starting point: From the record at the beginning of the partition
    - Ending point: Until all peer records of the current record

- peer: The records whose sorting order of window ORDER BY clause is same

```
gSQL>
SELECT orderdate AS O_DATE,
       orderkey AS O_KEY,
       custkey,
       totalprice,
       SUM( totalprice ) OVER( PARTITION BY orderdate
                               ORDER BY custkey ) AS SUM_OVER_RESULT
  FROM orders;
O_DATE     O_KEY CUSTKEY TOTALPRICE SUM_OVER_RESULT
---------- ----- ------- ---------- ---------------
2000-01-01   101 3088161     180000          180000
2000-01-01   102 3088163      42000          269000   <- peer ( custkey values are same. )
2000-01-01   103 3088163      47000          269000      peer
2000-01-01   104 3088165     217000          486000
2000-01-01   105 3088167     108000          734000   <- peer ( custkey values are same. )
2000-01-01   106 3088167      60000          734000      peer
2000-01-01   107 3088167      80000          734000      peer
...
```

```
15 rows selected.
```

- When <window frame clause> is omitted

```
gSQL>
SELECT orderdate AS O_DATE,
       orderkey AS O_KEY,
       custkey,
       totalprice,
       SUM( totalprice ) OVER( PARTITION BY orderdate
                               ORDER BY custkey ) AS SUM_OVER_RESULT
  FROM orders;
O_DATE     O_KEY CUSTKEY TOTALPRICE    SUM_OVER_RESULT
---------- ----- ------- ----------    ---------------
2000-01-01   101 3088161     180000 ❶     180000 ❶
2000-01-01   102 3088163      42000 ❷     269000 ❶+❷+❸
2000-01-01   103 3088163      47000 ❸     269000 ❶+❷+❸
2000-01-01   104 3088165     217000 ❹     486000 ❶+❷+❸+❹
2000-01-01   105 3088167     108000 ❺     734000 ❶+❷+❸+❹+❺+❻+❼
2000-01-01   106 3088167      60000 ❻     734000 ❶+❷+❸+❹+❺+❻+❼
2000-01-01   107 3088167      80000 ❼     734000 ❶+❷+❸+❹+❺+❻+❼
2000-01-01   108 3088169      32000 ❽     766000 ❶+❷+❸+❹+❺+❻+❼+❽
2000-01-01   109 3088170      30000 ❾     816000 ❶+❷+❸+❹+❺+❻+❼+❽+❾+❿
2000-01-01   110 3088170      20000 ❿     816000 ❶+❷+❸+❹+❺+❻+❼+❽+❾+❿
--------------------------------------------------------------------------
2000-03-03   301 3088161     180000 ❶     222000 ❶+❷
2000-03-03   302 3088161      42000 ❷     222000 ❶+❷
2000-03-03   303 3088165      47000 ❸     269000 ❶+❷+❸
2000-03-03   304 3088167     217000 ❹     594000 ❶+❷+❸+❹+❺
2000-03-03   305 3088167     108000 ❺     594000 ❶+❷+❸+❹+❺
15 rows selected.
```

<window frame units>

ROWS/ RANGE/ GROUPS are units of window frame scope.

<window frame extent>

It defines window frame start (the starting point) and window frame end (ending point).

- UNBOUNDED PRECEDING
  - From the record at the beginning of the partition
- UNBOUNDED FOLLOWING

    - Until the record at the end of the partition
- CURRENT ROW
    - When it is <window frame units> ROWS,
        - it is the current record.
    - When it is <window frame units> RANGE/ GROUPS
        - they are all peer records in the current record.
- offset PRECEDING/ offset FOLLOWING
    - When it is <window frame units> ROWS
        - it is the range of the number of offset records before and after the current record.
    - When it is <window frame units> GROUPS
        - it is the range of the number of offset groups before and after the current record group.
    - When it is <window frame units> RANGE
        - it is the range of offset values before and after the current record.
        - When sorting ORDER BY column in ASC order
        - ··· offset PRECEDING → The record whose value is above (sort key value of the current recor d - offset)
        - ··· offset FOLLOWING → The record whose value is below (sort key value of the current recor d + offset)
        - When sorting ORDER BY column in DESC order
        - ··· offset PRECEDING → The record whose value is below (sort key value of the current recor d + offset)
        - ··· offset FOLLOWING → The record whose value is above (sort key value of the current recor d - offset)
- For more information, refer to **Example of Using <window frame extent>**.

<window frame exclusion>

It defines records to exclude from window frame.

- EXCLUDE CURRENT ROW: It excludes the current record.
- EXCLUDE GROUP: It excludes the current record and all peer records.
- EXCLUDE TIES: It maintains the current record but excludes all peer records.
- EXCLUDE NO OTHERS: It does not exclude any record.
- If it is not specified, the default value is EXCLUDE NO OTHERS.
- For more information, refer to **Example of Using <window frame exclusion>**.

Example of Using <window frame extent>

- BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW

```
# ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
gSQL>
SELECT orderdate AS O_DATE,
```

```
       orderkey AS O_KEY,
       custkey,
       totalprice,
       SUM( totalprice ) OVER( PARTITION BY orderdate
                               ORDER BY custkey
                               ROWS BETWEEN UNBOUNDED PRECEDING
                                        AND CURRENT ROW ) AS SUM_OVER_RESULT
  FROM orders;
O_DATE     O_KEY CUSTKEY TOTALPRICE    SUM_OVER_RESULT
---------- ----- ------- ----------    ---------------
2000-01-01   101 3088161     180000 ❶     180000 ❶
2000-01-01   102 3088163      42000 ❷     222000 ❶+❷
2000-01-01   103 3088163      47000 ❸     269000 ❶+❷+❸
2000-01-01   104 3088165     217000 ❹     486000 ❶+❷+❸+❹
2000-01-01   105 3088167     108000 ❺     594000 ❶+❷+❸+❹+❺
2000-01-01   106 3088167      60000 ❻     654000 ❶+❷+❸+❹+❺+❻
2000-01-01   107 3088167      80000 ❼     734000 ❶+❷+❸+❹+❺+❻+❼
2000-01-01   108 3088169      32000 ❽     766000 ❶+❷+❸+❹+❺+❻+❼+❽
2000-01-01   109 3088170      30000 ❾     796000 ❶+❷+❸+❹+❺+❻+❼+❽+❾
2000-01-01   110 3088170      20000 ❿     816000 ❶+❷+❸+❹+❺+❻+❼+❽+❾+❿
-------------------------------------------------------------------------
2000-03-03   301 3088161     180000 ❶     180000 ❶
2000-03-03   302 3088161      42000 ❷     222000 ❶+❷
2000-03-03   303 3088165      47000 ❸     269000 ❶+❷+❸
2000-03-03   304 3088167     217000 ❹     486000 ❶+❷+❸+❹
2000-03-03   305 3088167     108000 ❺     594000 ❶+❷+❸+❹+❺
15 rows selected.
```

```
# RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
gSQL>
SELECT orderdate AS O_DATE,
       orderkey AS O_KEY,
       custkey,
       totalprice,
       SUM( totalprice ) OVER( PARTITION BY orderdate
                               ORDER BY custkey
                               RANGE BETWEEN UNBOUNDED PRECEDING
                                         AND CURRENT ROW ) AS SUM_OVER_RESULT
  FROM orders;
O_DATE     O_KEY CUSTKEY TOTALPRICE    SUM_OVER_RESULT
---------- ----- ------- ----------    ---------------
```

```
2000-01-01   101 3088161     180000 ❶     180000 ❶
2000-01-01   102 3088163      42000 ❷     269000 ❶+❷+❸
2000-01-01   103 3088163      47000 ❸     269000 ❶+❷+❸
2000-01-01   104 3088165     217000 ❹     486000 ❶+❷+❸+❹
2000-01-01   105 3088167     108000 ❺     734000 ❶+❷+❸+❹+❺+❻+❼
2000-01-01   106 3088167      60000 ❻     734000 ❶+❷+❸+❹+❺+❻+❼
2000-01-01   107 3088167      80000 ❼     734000 ❶+❷+❸+❹+❺+❻+❼
2000-01-01   108 3088169      32000 ❽     766000 ❶+❷+❸+❹+❺+❻+❼+❽
2000-01-01   109 3088170      30000 ❾     816000 ❶+❷+❸+❹+❺+❻+❼+❽+❾+❿
2000-01-01   110 3088170      20000 ❿     816000 ❶+❷+❸+❹+❺+❻+❼+❽+❾+❿
-------------------------------------------------------------------------
2000-03-03   301 3088161     180000 ❶     222000 ❶+❷
2000-03-03   302 3088161      42000 ❷     222000 ❶+❷
2000-03-03   303 3088165      47000 ❸     269000 ❶+❷+❸
2000-03-03   304 3088167     217000 ❹     594000 ❶+❷+❸+❹+❺
2000-03-03   305 3088167     108000 ❺     594000 ❶+❷+❸+❹+❺
15 rows selected.
```

```
# GROUPS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
gSQL>
SELECT orderdate AS O_DATE,
       orderkey AS O_KEY,
       custkey,
       totalprice,
       SUM( totalprice ) OVER( PARTITION BY orderdate
                               ORDER BY custkey
                               GROUPS BETWEEN UNBOUNDED PRECEDING
                                         AND CURRENT ROW ) AS SUM_OVER_RESULT
  FROM orders;
O_DATE     O_KEY CUSTKEY TOTALPRICE    SUM_OVER_RESULT
---------- ----- ------- ----------    ---------------
2000-01-01   101 3088161     180000 ❶     180000 ❶
2000-01-01   102 3088163      42000 ❷     269000 ❶+❷+❸
2000-01-01   103 3088163      47000 ❸     269000 ❶+❷+❸
2000-01-01   104 3088165     217000 ❹     486000 ❶+❷+❸+❹
2000-01-01   105 3088167     108000 ❺     734000 ❶+❷+❸+❹+❺+❻+❼
2000-01-01   106 3088167      60000 ❻     734000 ❶+❷+❸+❹+❺+❻+❼
2000-01-01   107 3088167      80000 ❼     734000 ❶+❷+❸+❹+❺+❻+❼
2000-01-01   108 3088169      32000 ❽     766000 ❶+❷+❸+❹+❺+❻+❼+❽
2000-01-01   109 3088170      30000 ❾     816000 ❶+❷+❸+❹+❺+❻+❼+❽+❾+❿
2000-01-01   110 3088170      20000 ❿     816000 ❶+❷+❸+❹+❺+❻+❼+❽+❾+❿
```

```
------------------------------------------------------------------------
2000-03-03   301 3088161     180000 ❶     222000 ❶+❷
2000-03-03   302 3088161      42000 ❷     222000 ❶+❷
2000-03-03   303 3088165      47000 ❸     269000 ❶+❷+❸
2000-03-03   304 3088167     217000 ❹     594000 ❶+❷+❸+❹+❺
2000-03-03   305 3088167     108000 ❺     594000 ❶+❷+❸+❹+❺
15 rows selected.
```

- BETWEEN offset PRECEDING AND offset FOLLOWING

```
# ROWS BETWEEN 1 PRECEDING AND 2 FOLLOWING
gSQL>
SELECT orderdate AS O_DATE,
       orderkey AS O_KEY,
       custkey,
       totalprice,
       SUM( totalprice ) OVER( PARTITION BY orderdate
                               ORDER BY custkey
                               ROWS BETWEEN 1 PRECEDING
                                        AND 2 FOLLOWING ) AS SUM_OVER_RESULT
  FROM orders;
O_DATE     O_KEY CUSTKEY TOTALPRICE    SUM_OVER_RESULT
---------- ----- ------- ----------    ---------------
2000-01-01   101 3088161     180000 ❶     269000 ❶+❷+❸
2000-01-01   102 3088163      42000 ❷     486000 ❶+❷+❸+❹
2000-01-01   103 3088163      47000 ❸     414000 ❷+❸+❹+❺
2000-01-01   104 3088165     217000 ❹     432000 ❸+❹+❺+❻
2000-01-01   105 3088167     108000 ❺     465000 ❹+❺+❻+❼
2000-01-01   106 3088167      60000 ❻     280000 ❺+❻+❼+❽
2000-01-01   107 3088167      80000 ❼     202000 ❻+❼+❽+❾
2000-01-01   108 3088169      32000 ❽     162000 ❼+❽+❾+❿
2000-01-01   109 3088170      30000 ❾      82000 ❽+❾+❿
2000-01-01   110 3088170      20000 ❿      50000 ❾+❿
------------------------------------------------------------------------
2000-03-03   301 3088161     180000 ❶     269000 ❶+❷+❸
2000-03-03   302 3088161      42000 ❷     486000 ❶+❷+❸+❹
2000-03-03   303 3088165      47000 ❸     414000 ❷+❸+❹+❺
2000-03-03   304 3088167     217000 ❹     372000 ❸+❹+❺
2000-03-03   305 3088167     108000 ❺     325000 ❹+❺
15 rows selected.
```

```
# RANGE BETWEEN 1 PRECEDING AND 2 FOLLOWING
#####################################################
# When sorting ORDER BY column in ASC order
#####################################################
  • 1 PRECEDING
    -->   The value above ( sortkey value of the current row - 1 )
          = The value above ( custkey - 1 )
  • 2 FOLLOWING
    -->   The value below ( sortkey value of the current row + 2 )
          = The value below ( custkey + 2 )
gSQL>
SELECT orderdate AS O_DATE,
       orderkey AS O_KEY,
       custkey,
       totalprice,
       SUM( totalprice ) OVER( PARTITION BY orderdate
                               ORDER BY custkey
                               RANGE BETWEEN 1 PRECEDING
                                         AND 2 FOLLOWING ) AS SUM_OVER_RESULT
  FROM orders;
O_DATE     O_KEY CUSTKEY TOTALPRICE    SUM_OVER_RESULT
---------- ----- ------- ----------    ---------------
2000-01-01   101 3088161     180000 ❶     269000 ❶+❷+❸
2000-01-01   102 3088163      42000 ❷     306000 ❷+❸+❹
2000-01-01   103 3088163      47000 ❸     306000 ❷+❸+❹
2000-01-01   104 3088165     217000 ❹     465000 ❹+❺+❻+❼
2000-01-01   105 3088167     108000 ❺     280000 ❺+❻+❼+❽
2000-01-01   106 3088167      60000 ❻     280000 ❺+❻+❼+❽
2000-01-01   107 3088167      80000 ❼     280000 ❺+❻+❼+❽
2000-01-01   108 3088169      32000 ❽      82000 ❽+❾+❿
2000-01-01   109 3088170      30000 ❾      82000 ❽+❾+❿
2000-01-01   110 3088170      20000 ❿      82000 ❽+❾+❿
-------------------------------------------------------------------------
2000-03-03   301 3088161     180000 ❶     222000 ❶+❷
2000-03-03   302 3088161      42000 ❷     222000 ❶+❷
2000-03-03   303 3088165      47000 ❸     372000 ❸+❹+❺
2000-03-03   304 3088167     217000 ❹     325000 ❹+❺
2000-03-03   305 3088167     108000 ❺     325000 ❹+❺
15 rows selected.
#####################################################
# When sorting ORDER BY column in DESC order
```

```
################################################
  • 1 PRECEDING
    -->   The value below ( sortkey value of the current row + 1 )
        = The value below ( custkey + 1 )
  • 2 FOLLOWING
    -->   The value above ( sortkey value of the current row - 2 )
        = The value above ( custkey - 2 )
gSQL>
SELECT orderdate AS O_DATE,
       orderkey AS O_KEY,
       custkey,
       totalprice,
       SUM( totalprice ) OVER( PARTITION BY orderdate
                               ORDER BY custkey DESC
                               RANGE BETWEEN 1 PRECEDING
                                         AND 2 FOLLOWING ) AS SUM_OVER_RESULT
  FROM orders;
O_DATE     O_KEY CUSTKEY TOTALPRICE    SUM_OVER_RESULT
---------- ----- ------- ----------    ---------------
2000-01-01   109 3088170      30000 ❶      82000 ❶+❷+❸
2000-01-01   110 3088170      20000 ❷      82000 ❶+❷+❸
2000-01-01   108 3088169      32000 ❸     330000 ❶+❷+❸+❹+❺+❻
2000-01-01   105 3088167     108000 ❹     465000 ❹+❺+❻+❼
2000-01-01   106 3088167      60000 ❺     465000 ❹+❺+❻+❼
2000-01-01   107 3088167      80000 ❻     465000 ❹+❺+❻+❼
2000-01-01   104 3088165     217000 ❼     306000 ❼+❽+❾
2000-01-01   102 3088163      42000 ❽     269000 ❽+❾+❿
2000-01-01   103 3088163      47000 ❾     269000 ❽+❾+❿
2000-01-01   101 3088161     180000 ❿     180000 ❿
-------------------------------------------------------------------------
2000-03-03   304 3088167     217000 ❶     372000 ❶+❷+❸
2000-03-03   305 3088167     108000 ❷     372000 ❶+❷+❸
2000-03-03   303 3088165      47000 ❸      47000 ❸
2000-03-03   301 3088161     180000 ❹     222000 ❹+❺
2000-03-03   302 3088161      42000 ❺     222000 ❹+❺
15 rows selected.
```

```
# GROUPS BETWEEN 1 PRECEDING AND 2 FOLLOWING
gSQL>
SELECT orderdate AS O_DATE,
       orderkey AS O_KEY,
```

```
       custkey,
       totalprice,
       SUM( totalprice ) OVER( PARTITION BY orderdate
                               ORDER BY custkey
                               GROUPS BETWEEN 1 PRECEDING
                                         AND 2 FOLLOWING ) AS SUM_OVER_RESULT
  FROM orders;
O_DATE     O_KEY CUSTKEY TOTALPRICE    SUM_OVER_RESULT
---------- ----- ------- ----------    ---------------
2000-01-01   101 3088161     180000 ❶     486000 ❶+❷+❸+❹
2000-01-01   102 3088163      42000 ❷     734000 ❶+❷+❸+❹+❺+❻+❼
2000-01-01   103 3088163      47000 ❸     734000 ❶+❷+❸+❹+❺+❻+❼
2000-01-01   104 3088165     217000 ❹     586000 ❷+❸+❹+❺+❻+❼+❽
2000-01-01   105 3088167     108000 ❺     547000 ❹+❺+❻+❼+❽+❾+❿
2000-01-01   106 3088167      60000 ❻     547000 ❹+❺+❻+❼+❽+❾+❿
2000-01-01   107 3088167      80000 ❼     547000 ❹+❺+❻+❼+❽+❾+❿
2000-01-01   108 3088169      32000 ❽     330000 ❺+❻+❼+❽+❾+❿
2000-01-01   109 3088170      30000 ❾      82000 ❽+❾+❿
2000-01-01   110 3088170      20000 ❿      82000 ❽+❾+❿
------------------------------------------------------------------------
2000-03-03   301 3088161     180000 ❶     594000 ❶+❷+❸+❹+❺
2000-03-03   302 3088161      42000 ❷     594000 ❶+❷+❸+❹+❺
2000-03-03   303 3088165      47000 ❸     594000 ❶+❷+❸+❹+❺
2000-03-03   304 3088167     217000 ❹     372000 ❸+❹+❺
2000-03-03   305 3088167     108000 ❺     372000 ❸+❹+❺
15 rows selected.
```

- BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING

```
# ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING
gSQL>
SELECT orderdate AS O_DATE,
       orderkey AS O_KEY,
       custkey,
       totalprice,
       SUM( totalprice ) OVER( PARTITION BY orderdate
                               ORDER BY custkey
                               ROWS BETWEEN CURRENT ROW
                                        AND UNBOUNDED FOLLOWING ) AS SUM_OVER_RESULT
  FROM orders;
O_DATE     O_KEY CUSTKEY TOTALPRICE SUM_OVER_RESULT
```

```
---------- ----- ------- ---------- ---------------
2000-01-01   101 3088161     180000          816000
2000-01-01   102 3088163      42000          636000
2000-01-01   103 3088163      47000          594000
2000-01-01   104 3088165     217000          547000
2000-01-01   105 3088167     108000          330000
2000-01-01   106 3088167      60000          222000
2000-01-01   107 3088167      80000          162000
2000-01-01   108 3088169      32000           82000
2000-01-01   109 3088170      30000           50000
2000-01-01   110 3088170      20000           20000
2000-03-03   301 3088161     180000          594000
2000-03-03   302 3088161      42000          414000
2000-03-03   303 3088165      47000          372000
2000-03-03   304 3088167     217000          325000
2000-03-03   305 3088167     108000          108000
15 rows selected.
```

```
# RANGE BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING
gSQL>
SELECT orderdate AS O_DATE,
       orderkey AS O_KEY,
       custkey,
       totalprice,
       SUM( totalprice ) OVER( PARTITION BY orderdate
                               ORDER BY custkey
                               RANGE BETWEEN CURRENT ROW
                                         AND UNBOUNDED FOLLOWING ) AS SUM_OVER_RESULT
  FROM orders;
O_DATE     O_KEY CUSTKEY TOTALPRICE SUM_OVER_RESULT
---------- ----- ------- ---------- ---------------
2000-01-01   101 3088161     180000          816000
2000-01-01   102 3088163      42000          636000
2000-01-01   103 3088163      47000          636000
2000-01-01   104 3088165     217000          547000
2000-01-01   105 3088167     108000          330000
2000-01-01   106 3088167      60000          330000
2000-01-01   107 3088167      80000          330000
2000-01-01   108 3088169      32000           82000
2000-01-01   109 3088170      30000           50000
2000-01-01   110 3088170      20000           50000
```

```
2000-03-03   301 3088161     180000          594000
2000-03-03   302 3088161      42000          594000
2000-03-03   303 3088165      47000          372000
2000-03-03   304 3088167     217000          325000
2000-03-03   305 3088167     108000          325000
15 rows selected.
```

```
# GROUPS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING
gSQL>
SELECT orderdate AS O_DATE,
       orderkey AS O_KEY,
       custkey,
       totalprice,
       SUM( totalprice ) OVER( PARTITION BY orderdate
                               ORDER BY custkey
                               GROUPS BETWEEN CURRENT ROW
                                          AND UNBOUNDED FOLLOWING ) AS SUM_OVER_RESULT
  FROM orders;
O_DATE     O_KEY CUSTKEY TOTALPRICE SUM_OVER_RESULT
---------- ----- ------- ---------- ---------------
2000-01-01   101 3088161     180000          816000
2000-01-01   102 3088163      42000          636000
2000-01-01   103 3088163      47000          636000
2000-01-01   104 3088165     217000          547000
2000-01-01   105 3088167     108000          330000
2000-01-01   106 3088167      60000          330000
2000-01-01   107 3088167      80000          330000
2000-01-01   108 3088169      32000           82000
2000-01-01   109 3088170      30000           50000
2000-01-01   110 3088170      20000           50000
2000-03-03   301 3088161     180000          594000
2000-03-03   302 3088161      42000          594000
2000-03-03   303 3088165      47000          372000
2000-03-03   304 3088167     217000          325000
2000-03-03   305 3088167     108000          325000
15 rows selected.
```

Example of Using <window frame exclusion>

- EXCLUDE CURRENT ROW

```
# ROWS
gSQL>
SELECT orderdate AS O_DATE,
       orderkey AS O_KEY,
       custkey,
       totalprice,
       SUM( totalprice ) OVER( PARTITION BY orderdate
                               ORDER BY custkey
                               ROWS BETWEEN UNBOUNDED PRECEDING
                                        AND CURRENT ROW
                               EXCLUDE CURRENT ROW ) AS SUM_OVER_RESULT
  FROM orders;
O_DATE     O_KEY CUSTKEY TOTALPRICE    SUM_OVER_RESULT
---------- ----- ------- ----------    ---------------
2000-01-01   101 3088161     180000 ❶       null
2000-01-01   102 3088163      42000 ❷     180000 ❶
2000-01-01   103 3088163      47000 ❸     222000 ❶+❷
2000-01-01   104 3088165     217000 ❹     269000 ❶+❷+❸
2000-01-01   105 3088167     108000 ❺     486000 ❶+❷+❸+❹
2000-01-01   106 3088167      60000 ❻     594000 ❶+❷+❸+❹+❺
2000-01-01   107 3088167      80000 ❼     654000 ❶+❷+❸+❹+❺+❻
2000-01-01   108 3088169      32000 ❽     734000 ❶+❷+❸+❹+❺+❻+❼
2000-01-01   109 3088170      30000 ❾     766000 ❶+❷+❸+❹+❺+❻+❼+❽
2000-01-01   110 3088170      20000 ❿     796000 ❶+❷+❸+❹+❺+❻+❼+❽+❾
------------------------------------------------------------------------
2000-03-03   301 3088161     180000 ❶       null
2000-03-03   302 3088161      42000 ❷     180000 ❶
2000-03-03   303 3088165      47000 ❸     222000 ❶+❷
2000-03-03   304 3088167     217000 ❹     269000 ❶+❷+❸
2000-03-03   305 3088167     108000 ❺     486000 ❶+❷+❸+❹
15 rows selected.
```

```
# RANGE
gSQL>
SELECT orderdate AS O_DATE,
       orderkey AS O_KEY,
       custkey,
       totalprice,
       SUM( totalprice ) OVER( PARTITION BY orderdate
                               ORDER BY custkey
                               RANGE BETWEEN UNBOUNDED PRECEDING
```

```
                                    AND CURRENT ROW
                                EXCLUDE CURRENT ROW ) AS SUM_OVER_RESULT
  FROM orders;
O_DATE     O_KEY CUSTKEY TOTALPRICE    SUM_OVER_RESULT
---------- ----- ------- ----------    ---------------
2000-01-01   101 3088161     180000 ❶       null
2000-01-01   102 3088163      42000 ❷     227000 ❶+❸
2000-01-01   103 3088163      47000 ❸     222000 ❶+❷
2000-01-01   104 3088165     217000 ❹     269000 ❶+❷+❸
2000-01-01   105 3088167     108000 ❺     626000 ❶+❷+❸+❹+❻+❼
2000-01-01   106 3088167      60000 ❻     674000 ❶+❷+❸+❹+❺+❼
2000-01-01   107 3088167      80000 ❼     654000 ❶+❷+❸+❹+❺+❻
2000-01-01   108 3088169      32000 ❽     734000 ❶+❷+❸+❹+❺+❻+❼
2000-01-01   109 3088170      30000 ❾     786000 ❶+❷+❸+❹+❺+❻+❼+❽+❿
2000-01-01   110 3088170      20000 ❿     796000 ❶+❷+❸+❹+❺+❻+❼+❽+❾
-------------------------------------------------------------------------
2000-03-03   301 3088161     180000 ❶      42000 ❷
2000-03-03   302 3088161      42000 ❷     180000 ❶
2000-03-03   303 3088165      47000 ❸     222000 ❶+❷
2000-03-03   304 3088167     217000 ❹     377000 ❶+❷+❸+❺
2000-03-03   305 3088167     108000 ❺     486000 ❶+❷+❸+❹
15 rows selected.
```

```
# GROUPS
gSQL>
SELECT orderdate AS O_DATE,
       orderkey AS O_KEY,
       custkey,
       totalprice,
       SUM( totalprice ) OVER( PARTITION BY orderdate
                               ORDER BY custkey
                               GROUPS BETWEEN UNBOUNDED PRECEDING
                                         AND CURRENT ROW
                               EXCLUDE CURRENT ROW ) AS SUM_OVER_RESULT
  FROM orders;
O_DATE     O_KEY CUSTKEY TOTALPRICE    SUM_OVER_RESULT
---------- ----- ------- ----------    ---------------
2000-01-01   101 3088161     180000 ❶       null
2000-01-01   102 3088163      42000 ❷     227000 ❶+❸
2000-01-01   103 3088163      47000 ❸     222000 ❶+❷
2000-01-01   104 3088165     217000 ❹     269000 ❶+❷+❸
```

```
2000-01-01   105 3088167     108000 ❺     626000 ❶+❷+❸+❹+❻+❼
2000-01-01   106 3088167      60000 ❻     674000 ❶+❷+❸+❹+❺+❼
2000-01-01   107 3088167      80000 ❼     654000 ❶+❷+❸+❹+❺+❻
2000-01-01   108 3088169      32000 ❽     734000 ❶+❷+❸+❹+❺+❻+❼
2000-01-01   109 3088170      30000 ❾     786000 ❶+❷+❸+❹+❺+❻+❼+❽+❿
2000-01-01   110 3088170      20000 ❿     796000 ❶+❷+❸+❹+❺+❻+❼+❽+❾
-------------------------------------------------------------------------
2000-03-03   301 3088161     180000 ❶      42000 ❷
2000-03-03   302 3088161      42000 ❷     180000 ❶
2000-03-03   303 3088165      47000 ❸     222000 ❶+❷
2000-03-03   304 3088167     217000 ❹     377000 ❶+❷+❸+❺
2000-03-03   305 3088167     108000 ❺     486000 ❶+❷+❸+❹
15 rows selected.
```

- EXCLUDE GROUP

```
# ROWS
gSQL>
SELECT orderdate AS O_DATE,
       orderkey AS O_KEY,
       custkey,
       totalprice,
       SUM( totalprice ) OVER( PARTITION BY orderdate
                               ORDER BY custkey
                               ROWS BETWEEN UNBOUNDED PRECEDING
                                        AND CURRENT ROW
                               EXCLUDE GROUP ) AS SUM_OVER_RESULT
  FROM orders;
O_DATE     O_KEY CUSTKEY TOTALPRICE    SUM_OVER_RESULT
---------- ----- ------- ----------    ---------------
2000-01-01   101 3088161     180000 ❶       null
2000-01-01   102 3088163      42000 ❷     180000 ❶
2000-01-01   103 3088163      47000 ❸     180000 ❶
2000-01-01   104 3088165     217000 ❹     269000 ❶+❷+❸
2000-01-01   105 3088167     108000 ❺     486000 ❶+❷+❸+❹
2000-01-01   106 3088167      60000 ❻     486000 ❶+❷+❸+❹
2000-01-01   107 3088167      80000 ❼     486000 ❶+❷+❸+❹
2000-01-01   108 3088169      32000 ❽     734000 ❶+❷+❸+❹+❺+❻+❼
2000-01-01   109 3088170      30000 ❾     766000 ❶+❷+❸+❹+❺+❻+❼+❽
2000-01-01   110 3088170      20000 ❿     766000 ❶+❷+❸+❹+❺+❻+❼+❽
-------------------------------------------------------------------------
```

```
2000-03-03   301 3088161     180000 ❶       null
2000-03-03   302 3088161      42000 ❷       null
2000-03-03   303 3088165      47000 ❸     222000 ❶+❷
2000-03-03   304 3088167     217000 ❹     269000 ❶+❷+❸
2000-03-03   305 3088167     108000 ❺     269000 ❶+❷+❸
15 rows selected.
```

```
# RANGE
gSQL>
SELECT orderdate AS O_DATE,
       orderkey AS O_KEY,
       custkey,
       totalprice,
       SUM( totalprice ) OVER( PARTITION BY orderdate
                               ORDER BY custkey
                               RANGE BETWEEN UNBOUNDED PRECEDING
                                         AND CURRENT ROW
                               EXCLUDE GROUP ) AS SUM_OVER_RESULT
  FROM orders;
O_DATE     O_KEY CUSTKEY TOTALPRICE    SUM_OVER_RESULT
---------- ----- ------- ----------    ---------------
2000-01-01   101 3088161     180000 ❶       null
2000-01-01   102 3088163      42000 ❷     180000 ❶
2000-01-01   103 3088163      47000 ❸     180000 ❶
2000-01-01   104 3088165     217000 ❹     269000 ❶+❷+❸
2000-01-01   105 3088167     108000 ❺     486000 ❶+❷+❸+❹
2000-01-01   106 3088167      60000 ❻     486000 ❶+❷+❸+❹
2000-01-01   107 3088167      80000 ❼     486000 ❶+❷+❸+❹
2000-01-01   108 3088169      32000 ❽     734000 ❶+❷+❸+❹+❺+❻+❼
2000-01-01   109 3088170      30000 ❾     766000 ❶+❷+❸+❹+❺+❻+❼+❽
2000-01-01   110 3088170      20000 ❿     766000 ❶+❷+❸+❹+❺+❻+❼+❽
-------------------------------------------------------------------------
2000-03-03   301 3088161     180000 ❶       null
2000-03-03   302 3088161      42000 ❷       null
2000-03-03   303 3088165      47000 ❸     222000 ❶+❷
2000-03-03   304 3088167     217000 ❹     269000 ❶+❷+❸
2000-03-03   305 3088167     108000 ❺     269000 ❶+❷+❸
15 rows selected.
```

```
# GROUPS
gSQL> SELECT orderdate AS O_DATE,
```

```
       orderkey AS O_KEY,
       custkey,
       totalprice,
       SUM( totalprice ) OVER( PARTITION BY orderdate
                               ORDER BY custkey
                               GROUPS BETWEEN UNBOUNDED PRECEDING
                                          AND CURRENT ROW
                               EXCLUDE GROUP ) AS SUM_OVER_RESULT
  FROM orders;
O_DATE     O_KEY CUSTKEY TOTALPRICE    SUM_OVER_RESULT
---------- ----- ------- ----------    ---------------
2000-01-01   101 3088161     180000 ❶       null
2000-01-01   102 3088163      42000 ❷     180000 ❶
2000-01-01   103 3088163      47000 ❸     180000 ❶
2000-01-01   104 3088165     217000 ❹     269000 ❶+❷+❸
2000-01-01   105 3088167     108000 ❺     486000 ❶+❷+❸+❹
2000-01-01   106 3088167      60000 ❻     486000 ❶+❷+❸+❹
2000-01-01   107 3088167      80000 ❼     486000 ❶+❷+❸+❹
2000-01-01   108 3088169      32000 ❽     734000 ❶+❷+❸+❹+❺+❻+❼
2000-01-01   109 3088170      30000 ❾     766000 ❶+❷+❸+❹+❺+❻+❼+❽
2000-01-01   110 3088170      20000 ❿     766000 ❶+❷+❸+❹+❺+❻+❼+❽
------------------------------------------------------------------------
2000-03-03   301 3088161     180000 ❶       null
2000-03-03   302 3088161      42000 ❷       null
2000-03-03   303 3088165      47000 ❸     222000 ❶+❷
2000-03-03   304 3088167     217000 ❹     269000 ❶+❷+❸
2000-03-03   305 3088167     108000 ❺     269000 ❶+❷+❸
15 rows selected.
```

- EXCLUDE TIES

```
# ROWS
gSQL>
SELECT orderdate AS O_DATE,
       orderkey AS O_KEY,
       custkey,
       totalprice,
       SUM( totalprice ) OVER( PARTITION BY orderdate
                               ORDER BY custkey
                               ROWS BETWEEN UNBOUNDED PRECEDING
                                        AND CURRENT ROW
```

```
                              EXCLUDE TIES ) AS SUM_OVER_RESULT
  FROM orders;
O_DATE     O_KEY CUSTKEY TOTALPRICE    SUM_OVER_RESULT
---------- ----- ------- ----------    ---------------
2000-01-01   101 3088161     180000 ❶     180000 ❶
2000-01-01   102 3088163      42000 ❷     222000 ❶+❷
2000-01-01   103 3088163      47000 ❸     227000 ❶+❸
2000-01-01   104 3088165     217000 ❹     486000 ❶+❷+❸+❹
2000-01-01   105 3088167     108000 ❺     594000 ❶+❷+❸+❹+❺
2000-01-01   106 3088167      60000 ❻     546000 ❶+❷+❸+❹+❻
2000-01-01   107 3088167      80000 ❼     566000 ❶+❷+❸+❹+❼
2000-01-01   108 3088169      32000 ❽     766000 ❶+❷+❸+❹+❺+❻+❼+❽
2000-01-01   109 3088170      30000 ❾     796000 ❶+❷+❸+❹+❺+❻+❼+❽+❾
2000-01-01   110 3088170      20000 ❿     786000 ❶+❷+❸+❹+❺+❻+❼+❽+❿
-------------------------------------------------------------------------
2000-03-03   301 3088161     180000 ❶     180000 ❶
2000-03-03   302 3088161      42000 ❷      42000 ❷
2000-03-03   303 3088165      47000 ❸     269000 ❶+❷+❸
2000-03-03   304 3088167     217000 ❹     486000 ❶+❷+❸+❹
2000-03-03   305 3088167     108000 ❺     377000 ❶+❷+❸+❺
15 rows selected.
```

```
# RANGE
gSQL>
SELECT orderdate AS O_DATE,
       orderkey AS O_KEY,
       custkey,
       totalprice,
       SUM( totalprice ) OVER( PARTITION BY orderdate
                               ORDER BY custkey
                               RANGE BETWEEN UNBOUNDED PRECEDING
                                         AND CURRENT ROW
                               EXCLUDE TIES ) AS SUM_OVER_RESULT
  FROM orders;
O_DATE     O_KEY CUSTKEY TOTALPRICE    SUM_OVER_RESULT
---------- ----- ------- ----------    ---------------
2000-01-01   101 3088161     180000 ❶     180000 ❶
2000-01-01   102 3088163      42000 ❷     222000 ❶+❷
2000-01-01   103 3088163      47000 ❸     227000 ❶+❸
2000-01-01   104 3088165     217000 ❹     486000 ❶+❷+❸+❹
2000-01-01   105 3088167     108000 ❺     594000 ❶+❷+❸+❹+❺
```

```
2000-01-01   106 3088167      60000 ❻     546000 ❶+❷+❸+❹+❻
2000-01-01   107 3088167      80000 ❼     566000 ❶+❷+❸+❹+❼
2000-01-01   108 3088169      32000 ❽     766000 ❶+❷+❸+❹+❺+❻+❼+❽
2000-01-01   109 3088170      30000 ❾     796000 ❶+❷+❸+❹+❺+❻+❼+❽+❾
2000-01-01   110 3088170      20000 ❿     786000 ❶+❷+❸+❹+❺+❻+❼+❽+❿
-------------------------------------------------------------------------
2000-03-03   301 3088161     180000 ❶     180000 ❶
2000-03-03   302 3088161      42000 ❷      42000 ❷
2000-03-03   303 3088165      47000 ❸     269000 ❶+❷+❸
2000-03-03   304 3088167     217000 ❹     486000 ❶+❷+❸+❹
2000-03-03   305 3088167     108000 ❺     377000 ❶+❷+❸+❺
15 rows selected.
```

```
# GROUPS
gSQL>
SELECT orderdate AS O_DATE,
       orderkey AS O_KEY,
       custkey,
       totalprice,
       SUM( totalprice ) OVER( PARTITION BY orderdate
                               ORDER BY custkey
                               GROUPS BETWEEN UNBOUNDED PRECEDING
                                          AND CURRENT ROW
                               EXCLUDE TIES ) AS SUM_OVER_RESULT
  FROM orders;
O_DATE     O_KEY CUSTKEY TOTALPRICE    SUM_OVER_RESULT
---------- ----- ------- ----------    ---------------
2000-01-01   101 3088161     180000 ❶     180000 ❶
2000-01-01   102 3088163      42000 ❷     222000 ❶+❷
2000-01-01   103 3088163      47000 ❸     227000 ❶+❸
2000-01-01   104 3088165     217000 ❹     486000 ❶+❷+❸+❹
2000-01-01   105 3088167     108000 ❺     594000 ❶+❷+❸+❹+❺
2000-01-01   106 3088167      60000 ❻     546000 ❶+❷+❸+❹+❻
2000-01-01   107 3088167      80000 ❼     566000 ❶+❷+❸+❹+❼
2000-01-01   108 3088169      32000 ❽     766000 ❶+❷+❸+❹+❺+❻+❼+❽
2000-01-01   109 3088170      30000 ❾     796000 ❶+❷+❸+❹+❺+❻+❼+❽+❾
2000-01-01   110 3088170      20000 ❿     786000 ❶+❷+❸+❹+❺+❻+❼+❽+❿
-------------------------------------------------------------------------
2000-03-03   301 3088161     180000 ❶     180000 ❶
2000-03-03   302 3088161      42000 ❷      42000 ❷
2000-03-03   303 3088165      47000 ❸     269000 ❶+❷+❸
```

```
2000-03-03   304 3088167     217000 ❹      486000 ❶+❷+❸+❹
2000-03-03   305 3088167     108000 ❺      377000 ❶+❷+❸+❺
15 rows selected.
```

Example

```
gSQL>
SELECT item_no,
       sales_date,
       sales,
       SUM( sales ) OVER W1 cumulative_sales,
       AVG( sales ) OVER w1 avg_sales
  FROM store
WINDOW w1 AS ( PARTITION BY item_no
               ORDER BY sales_date
               ROWS BETWEEN UNBOUNDED PRECEDING
                        AND CURRENT ROW );
ITEM_NO SALES_DATE SALES CUMULATIVE_SALES AVG_SALES
------- ---------- ----- ---------------- ---------
    100 2001-01-01   150              150       150
    100 2001-01-02   100              250       125
    100 2001-01-03   170              420       140
    100 2001-01-04    90              510     127.5
    100 2001-01-05   200              710       142
    235 2001-01-01    70               70        70
    235 2001-01-02   130              200       100
    235 2001-01-03   190              390       130
    235 2001-01-04   150              540       135
    235 2001-01-05    50              590       118
10 rows selected.
```

Compatibility

Table 20-16 SQL standard compatibility

Feature ID	Description	Compatibility
T611	Elementary OLAP operations	X
T612	Advanced OLAP operations	X
T301	Functional dependencies	X
T620	WINDOW clause: GROUPS option	O

For More Information

Refer to the followings.

- group by clause
- order by clause
- Window Function

order by clause

Function

It specifies the sorting order of the query results.

Syntax

```
<order by clause> ::=
    ORDER BY <sort specification list>
<sort specification list> ::=
    <sort specification> [ { <comma> <sort specification> }... ]
<sort specification> ::=
    <sort key> [ <ordering specification> ] [ <null ordering> ]
<sort key> ::=
    <value expression>
<ordering specification> ::=
      ASC
    ¦ DESC
<null ordering> ::=
      NULLS FIRST
    ¦ NULLS LAST
```

Invocation and Access Rules

The access privilege for a column is required if the column exist in a sort key specified for sorting.

Syntax Rules and Parameters

<order by clause>

- When <set quantifier> DISTINCT is specified in <query specification>, then only the expression specifie d in <select list> can appear in <sort key>.
  - SELECT DISTINCT c1, c2 FROM t1 ORDER BY c1;
  - (X) SELECT DISTINCT c1, c2 FROM t1 ORDER BY c5;
- When one or more <set function specification> are specified in <select list> of <query specification>, t hen only the expression specified in <select list> can appear in <sort key>.
  - SELECT c1, sum(c2) FROM t1 GROUP BY c1 ORDER BY c1;
  - (X) SELECT c1, sum(c2) FROM t1 GROUP BY c1 ORDER BY c5;

- If <order by clause> is specified in <set operator>, <sort key> is analysed based on the firstly specified <query specification>.
  - SELECT c1, c2 FROM t1 UNION SELECT i1, i2 FROM t3 ORDER BY c1, c2;
  - (X) SELECT c1, c2 FROM t1 UNION SELECT i1, i2 FROM t3 ORDER BY i1, i2;

<sort specification list>

- <ordering specification>
  - ASC
  - DESC
  - If it is not specified, the default value is ASC.
- <null ordering>
  - NULLS FIRST
  - NULLS LAST
  - If it is not specified, the default value is NULLS LAST.

<sort key>

- If <value expression> of <sort key> is the positive integer value, the value is used as a sort key index.
  - The i-th <select sublist> of <query specification> which corresponds to the value is used as a sort key.
    - SELECT c1, c2 FROM t1 ORDER BY 1;
    - C1 is sorted by the sort key.
  - If the i-th <select sublist> of <query specification> >which corresponds to the value does not exist, it returns an error.
    - (X) SELECT c1, c2 FROM t1 ORDER BY 3;
- A row subquery or relation subquery is not supported as <value expression>.
  - (X) SELECT c1, c2 FROM t1 ORDER BY (SELECT i1, i2 FROM t2 FETCH FIRST ROW ONLY);
  - Multiple records exist in T2.
    - (X) SELECT c1, c2 FROM t1 ORDER BY (SELECT i1 FROM t2);
- Other <value expression> are used as sort keys.

Description

<order by clause>

<order by clause> specifies a method to sort the query results.

<sort key> can be listed in <order by clause> by using a comma (,), and <sort key> of each records are co mpared and listed in order.

```
SELECT c1, c2 FROM t1 ORDER BY c1, c2;
```

<ordering specification> which specifies an ascending order or an descending order can be specified in <sort key>. If it is omitted, then they are sorted in an ascending order.

```
gSQL> SELECT c1 FROM t1;
C1
--
 2
 3
 1
3 rows selected.
```

- Ascending order (ASC)

```
gSQL> SELECT c1 FROM t1 ORDER BY c1;
C1
--
 1
 2
 3
3 rows selected.
gSQL> SELECT c1 FROM t1 ORDER BY c1 ASC;
C1
--
 1
 2
 3
3 rows selected.
```

- Descending order (DESC)

```
gSQL> SELECT c1 FROM t1 ORDER BY c1 DESC;
C1
--
 3
 2
 1
3 rows selected.
```

<null ordering> specifies an order between the non-NULL values and NULL values in <sort key>. If it is omi tted, then they are sorted as NULLS LAST.

```
gSQL> SELECT c1 FROM t1;
  C1
----
   2
null
   1
3 rows selected.
```

- NULLS LAST

```
gSQL> SELECT c1 FROM t1 ORDER BY c1;
  C1
----
   1
   2
null
3 rows selected.
gSQL> SELECT c1 FROM t1 ORDER BY c1 NULLS LAST;
  C1
----
   1
   2
null
3 rows selected.
```

- NULLS FIRST

```
gSQL> SELECT c1 FROM t1 ORDER BY c1 NULLS FIRST;
  C1
----
null
   1
   2
3 rows selected.
```

If a constant value is specified in <sort key>, the expression positioned in the location corresponding to th e order of the corresponding value in <select list> is regarded as <sort key>. In this case, the constant is an integer bigger than 0, and it should be equal or smaller than the total number of expression in <select list> .

```
gSQL> SELECT c1 FROM t1 ORDER BY 1;
  C1
----
   1
   2
null
3 rows selected.
```

LONG (LONG VARCHAR, LONG VARBINARY) type can not be specified in <sort key>.

Comparison of Null Value

- The comparison between null values is regarded as the same value.
- The comparison between non-null value and null value is subject to the following rules.
  - When it is NULLS FIRST and ASC: null value < not null value
  - When it is NULLS LAST and ASC: null value > not null value
  - When it is NULLS FIRST and DESC: null value > not null value
  - When it is NULLS LAST and DESC: null value < not null value
- If the comparison result between null values is UNKNOWN, it is sorted according to the scan order.

Sorting Rows Which Have Same Sort Key Value

Peers are rows which can not be distinguished by a sort key, and the peers are sorted according to the sc an order.

<aggregation function> Which Is Used As <sort key>

If <aggregation function> is used in <query specification>, or <group by clause> is specified, <aggregation function> can be used as <sort key>.
However, the nested aggregation function can be used as <sort key> only when <group by clause> is spec ified.

```
gSQL> SELECT c1, c2 FROM t1;
C1 C2
-- --
 2  1
 3  5
 1  2
 2 10
 3 10
5 rows selected.
gSQL> SELECT sum(c1) FROM t1 ORDER BY sum(c1);
```

```
SUM(C1)
-------
     11
1 row selected.
gSQL> SELECT c1, sum(c2) FROM t1 GROUP BY c1 ORDER BY sum(c2);
C1 SUM(C2)
-- -------
 1       2
 2      11
 3      15
3 rows selected.
gSQL> SELECT sum(c1) FROM t1 GROUP BY c1 ORDER BY sum(sum(c1));
SUM(C1)
-------
      6
1 row selected.
```

Example

The following is an example of SELECT statement which uses ORDER BY clause.

```
gSQL> SELECT c_name, c_nation FROM customer ORDER BY c_nation;
C_NAME     C_NATION
---------- -------------
Customer#2 CANADA
Customer#4 GERMANY
Customer#1 KOREA
Customer#3 KOREA
Customer#5 UNITED STATES
5 rows selected.
gSQL> SELECT c_name, c_nation FROM customer ORDER BY c_nation DESC;
C_NAME     C_NATION
---------- -------------
Customer#5 UNITED STATES
Customer#1 KOREA
Customer#3 KOREA
Customer#4 GERMANY
Customer#2 CANADA
5 rows selected.
gSQL> SELECT c_name, c_nation FROM customer ORDER BY 2 DESC;
C_NAME     C_NATION
```

```
---------- -------------
Customer#5 UNITED STATES
Customer#1 KOREA
Customer#3 KOREA
Customer#4 GERMANY
Customer#2 CANADA
5 rows selected.
```

Compatibility

Table 20-17 SQL standard compatibility

Feature ID	Description	Compatibility
F850	Top-level <order by clause> in <query expression>	O
F851	<order by clause> in subqueries	O
F852	Top-level <order by clause> in views	O
F855	Nested <order by clause> in <query expression>	O

For More Information

Refer to **query expression**.

offset limit clause

Function

It specifies the number of rows to skip and the number of rows to fetch from the query results.

Syntax

```
<offset limit clause> ::=
      <result offset clause>
    ¦ <fetch limit clause>
    ¦ <result offset clause> <fetch limit clause>
<result offset clause> ::=
    OFFSET <offset row count> [ { ROW ¦ ROWS } ]
<fetch limit clause> ::=
      <fetch first clause>
    ¦ <limit clause>
<fetch first clause> ::=
    FETCH [ FIRST ¦ NEXT ] [ <fetch row count> ] [ ROW ONLY ¦ ROWS ONLY ]
<limit clause> ::=
    LIMIT { <fetch row count> ¦ <offset row count> , <fetch row count> ¦ ALL }
```

Invocation and Access Rules

The access privilege for <offset limit clause> is not required.

Syntax Rules and Parameters

<result offset clause>

- <offset row count> value should be a positive integer which is equal to or bigger than zero.
- ROW and ROWS are keywords with the same meaning and they can be omitted.
- If the statement is omitted, it means as same as *OFFSET 0 ROWS*.

<fetch limit clause>

- It specifies the number of rows to skip in the query result.
- If the statement is omitted, it means as same as *LIMIT ALL*.

〈fetch first clause〉

- It specifies the number of rows to fetch from the query result.
- It can not be used together with 〈limit clause〉.
- FIRST and NEXT are keywords with the same meaning and they can be omitted.
- ROW ONLY and ROWS ONLY are keywords with the same meaning and they can be omitted.
- 〈fetch row count〉
  - It should be a positive integer which is bigger than zero.
  - It can be omitted and if it is omitted, its value is one.

〈limit clause〉

- It specifies the number of rows to fetch.
- It can simultaneously specify both the number of rows to fetch and the number of rows to be skippe d from query results .
- It can not be used together with 〈fetch first clause〉.
- When it is used as LIMIT 〈fetch row count〉
  - 〈fetch row count〉 should be a positive integer which is bigger than zero.
  - The statement means same as *FETCH FIRST 〈fetch row count〉 ROWS ONLY*.
- When it is used as LIMIT 〈offset row count〉, 〈fetch row count〉
  - It can not be used together with 〈result offset clause〉.
  - 〈offset row count〉 should be a positive integer which is equal to or bigger than zero.
  - 〈fetch row count〉 should be a positive integer which is bigger than zero.
  - The statement means same as *OFFSET 〈offset row count〉 ROWS FETCH FIRST 〈fetch row count〉 ROWS ONLY*.
- When it is used as LIMIT ALL
  - It does not limit the number of rows to fetch.

Description

〈result offset clause〉

It fetches rows from the 〈offset row count〉th of the query results. If the result which 〈offset row count〉 queried is equal to or greater than the number of rows, the number of fetch rows is 0.

```
gSQL> SELECT c1 FROM t1;
C1
--
 1
 2
 3
3 rows selected.
```

```
gSQL> SELECT c1 FROM t1 OFFSET 1;
C1
--
 2
 3
2 rows selected.
gSQL> SELECT c1 FROM t1 OFFSET 3;
no rows selected.
```

〈fetch first clause〉

It fetches the query results as many as the number of 〈fetch row count〉.

```
gSQL> SELECT c1 FROM t1;
C1
--
 1
 2
 3
3 rows selected.
gSQL> SELECT c1 FROM t1 FETCH FIRST 2 ROWS ONLY;
C1
--
 1
 2
2 rows selected.
```

〈limit clause〉

When LIMIT 〈fetch_row_count〉 is used, it fetches the query results as many as the number of 〈fetch row count〉.

When LIMIT 〈offset row count〉 is used, 〈fetch row count〉, it fetches the query results as many as the nu mber of 〈fetch row count〉 from the 〈offset row count〉th row.

When LIMIT ALL is used, it returns the query results to a user without limit of the number.

```
gSQL> SELECT c1 FROM t1;
C1
--
 1
 2
 3
```

```
3 rows selected.
• LIMIT <fetch_row_count>
gSQL> SELECT c1 FROM t1 LIMIT 2;
C1
--
 1
 2
2 rows selected.
• LIMIT <offset row count>, <fetch_row_count>
gSQL> SELECT c1 FROM t1 LIMIT 1, 1;
C1
--
 2
1 row selected.
• LIMIT ALL
gSQL> SELECT c1 FROM t1 LIMIT ALL;
C1
--
 1
 2
 3
3 rows selected.
```

Examples

The following is an example of SELECT statement which uses <result offset clause>.

```
gSQL> SELECT c_name, c_nation FROM customer OFFSET 1;
C_NAME     C_NATION
---------- -------------
Customer#2 CANADA
Customer#3 KOREA
Customer#4 GERMANY
Customer#5 UNITED STATES
4 rows selected.
```

The following is an example of SELECT statement which uses <fetch first clause>.

```
gSQL> SELECT c_name, c_nation FROM customer FETCH FIRST ROW ONLY;
C_NAME     C_NATION
---------- --------
```

```
Customer#1 KOREA
1 row selected.
gSQL> SELECT c_name, c_nation FROM customer FETCH FIRST 2 ROW ONLY;
C_NAME     C_NATION
---------- --------
Customer#1 KOREA
Customer#2 CANADA
2 rows selected.
```

The following is an example of SELECT statement which uses <limit clause>.

```
gSQL> SELECT c_name, c_nation FROM customer LIMIT 1;
C_NAME     C_NATION
---------- --------
Customer#1 KOREA
1 row selected.
gSQL> SELECT c_name, c_nation FROM customer LIMIT 1, 2;
C_NAME     C_NATION
---------- --------
Customer#2 CANADA
Customer#3 KOREA
2 rows selected.
gSQL> SELECT c_name, c_nation FROM customer LIMIT ALL;
C_NAME     C_NATION
---------- -------------
Customer#1 KOREA
Customer#2 CANADA
Customer#3 KOREA
Customer#4 GERMANY
Customer#5 UNITED STATES
5 rows selected.
```

The following is an example of SELECT statement which uses <result offset clause> and <fetch limit clause>
.

```
gSQL> SELECT c_name, c_nation FROM customer OFFSET 1 FETCH 2;
C_NAME     C_NATION
---------- --------
Customer#2 CANADA
Customer#3 KOREA
2 rows selected.
gSQL> SELECT c_name, c_nation FROM customer OFFSET 1 LIMIT 2;
```

```
C_NAME     C_NATION
---------- --------
Customer#2 CANADA
Customer#3 KOREA
2 rows selected.
```

Compatibility

Table 20-18 SQL standard compatibility

Feature ID	Description	Compatibility
F861	Top-level <result offset clause> in <query expression>	O
F862	<result offset clause> in subqueries	O
F863	Nested <result offset clause> in <query expression>	O
F864	Top-level <result offset clause> in views	O
F865	dynamic <offset row count> in <result offset clause>	X

set operator

Function

It performs a set operation for results of the subquery.

Syntax

```
<set operator> ::=
      <set operator term>
    ¦ <query expression body> UNION [ ALL ¦ DISTINCT ] <set operator term>
    ¦ <query expression body> EXCEPT [ ALL ¦ DISTINCT ] <set operator term>
    ¦ <query expression body> MINUS [ ALL ¦ DISTINCT ] <set operator term>
<set operator term> ::=
      <query term>
    ¦ <set operator term> INTERSECT [ ALL ¦ DISTINCT ] <set operator term>
```

Invocation and Access Rules

All access privileges for the <query expression> in each <set operator term> are required for using <set operator> statement.

Syntax Rules and Parameters

<set operator>

- It specifies the set operations among the subqueries.
- The number of the target in <select list> of each subquery should be same, and all matched targets should belong to the same data type group.
- The representative name of the result target of <set operator> is the target name of <select list> of the first subquery.
  - gSQL> SELECT c1 AS NAME FROM t1 UNION SELECT i1 FROM t2;
    NAME

    1
    2
    2 rows selected.
- If the processing order is not explicitly specified by using parentheses it processes by evaluating specified subqueries from the left to the the right.
- The meaning of each operator in <set operator> is as follows.

- UNION
  - UNION ALL: It is a union of all subquery results without removing the duplicates.
  - UNION DISTINCT: It is a union of all subquery results, which removed the duplicates.
  - If at least one of ALL and DISTINCT is not specified, it is operated as same as when DISTINCT i s specified.
- EXCEPT
  - EXCEPT ALL: It returns the difference of all rows for the subquery result including all duplicat es.
  - EXCEPT DISTINCT: It returns the difference of all rows for the subquery result excluding all du plicates.
  - If at least one of ALL and DISTINCT is not specified, it is operated as same as when DISTINCT i s specified.
- MINUS
  - It is an alias of EXCEPT and, it is operated as same as EXCEPT.
- INTERSECT
  - INTERSECT ALL: It is a intersection of all subquery results without removing the duplicates.
  - INTERSECT DISTINCT: It is a intersection of all subquery results, which removed the duplicates.
  - If at least one of ALL and DISTINCT is not specified, it is operated as same as when DISTINCT i s specified.

〈query term〉

It specifies the single subquery.
For more information, refer to **query expression**.

Description

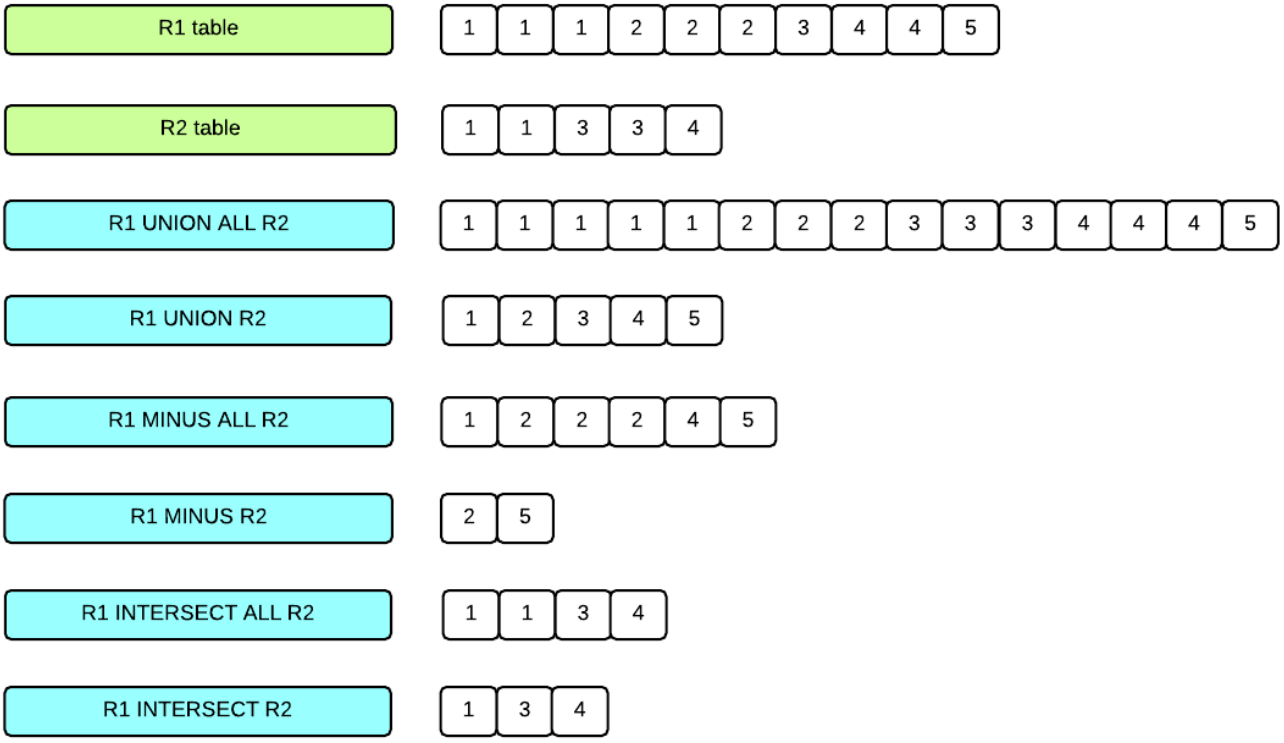
The Differences between ALL and DISTINCT in 〈set operator〉

For example, if the data of the table R1 and R2 is given as follows, the result of each 〈set operator 〉 is as follows.

- TABLE data
  - R1 TABLE = {1, 1, 1, 2, 2, 2, 3, 4, 4, 5}
  - R2 TABLE = {1, 1, 3, 3, 4}
- SELECT * FROM R1 UNION ALL SELECT * FROM R2;
  - result = {1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5}
- SELECT * FROM R1 UNION DISTINCT SELECT * FROM R2;
  - result = {1, 2, 3, 4, 5}
- SELECT * FROM R1 MINUS ALL SELECT * FROM R2;
  - result = {1, 2, 2, 2, 4, 5}
- SELECT * FROM R1 MINUS DISTINCT SELECT * FROM R2;

    - result = {2, 5}
- SELECT * FROM R1 INTERSECT ALL SELECT * FROM R2;
    - result = {1, 1, 3, 4}
- SELECT * FROM R1 INTERSECT DISTINCT SELECT * FROM R2;
    - result = {1, 3, 4}

Figure 1 SET operation results



Operator Precedence

The operator precedence of <set operator> is as follows.

- Parentheses ( ) has a priority.
- INTERSECT has a priority.
- For UNION and EXCEPT, the precedence is according to an order listed from left to right within an ex pression.

Result Type of <set operator>

The i-th column of all subqueries in <set operator> should be a data type of the same family, and its result type is determined by **Result Type Combination Rule**.
However, LONG VARCHAR and LONG VARBINARY types can only use UNION ALL.

ORDER BY Clause

When <set operator> is used together with ORDER BY, and the column names are different among subq ueries, then it can be used as follows.

- ORDER BY indicator
  - It specifies the order of result columns.
    SELECT c1 FROM t1
    UNION ALL
    SELECT c2 FROM t2
    ORDER BY 1;
- ORDER BY left_column_name
  - It specifies the column name of the first subquery.
    SELECT c1 FROM t1
    UNION ALL
    SELECT c2 FROM t2
    ORDER BY c1;

Examples

The following is an example of SELECT statement which uses UNION operator.

```
gSQL> SELECT s_nation nation FROM supplier UNION ALL SELECT c_nation FROM customer;
NATION
-------------
FRANCE
KOREA
GERMANY
UNITED STATES
CANADA
KOREA
CANADA
KOREA
GERMANY
UNITED STATES
10 rows selected.
gSQL> SELECT s_nation nation FROM supplier UNION DISTINCT SELECT c_nation FROM customer;
NATION
-------------
UNITED STATES
CANADA
KOREA
```

```
GERMANY
FRANCE
5 rows selected.
```

The following is an example of SELECT statement which uses EXCEPT operator.

```
gSQL> SELECT c_nation nation FROM customer EXCEPT ALL SELECT s_nation FROM supplier;
NATION
------
KOREA
1 row selected.
gSQL> SELECT c_nation nation FROM customer EXCEPT DISTINCT SELECT s_nation FROM supplier;
no rows selected.
```

The following is an example of SELECT statement which uses INTERSECT operator.

```
gSQL> SELECT c_nation nation FROM customer INTERSECT ALL SELECT s_nation FROM supplier;
NATION
--------------
UNITED STATES
CANADA
KOREA
GERMANY
4 rows selected.
gSQL> SELECT c_nation nation FROM customer INTERSECT DISTINCT SELECT s_nation FROM supplier;
NATION
--------------
UNITED STATES
CANADA
KOREA
GERMANY
4 rows selected.
```

Compatibility

Table 20-19 SQL standard compatibility

Feature ID	Description	Compatibility
F302	INTERSECT table operator	O
F301	CORRESPONDING	X
T551	Optional key words for default syntax	O
F304	EXCEPT ALL table operator	O

For More Information

Refer to **query expression**.

subquery

Function

It specifies the scalar value, row, table which are derived from <query expression>.

Syntax

```
<scalar subquery> ::=
    <subquery>
<row subquery> ::=
    <subquery>
<table subquery> ::=
    <subquery>
<subquery> ::=
    ( <query expression> )
```

Invocation and Access Rules

The access privilege for <query expression> in <subquery> is required.

Syntax Rules and Parameters

<scalar subquery>

- The number of targets in <query expression> should be one.
- The result value according to the number of rows returned from <query expression> is as follows.
  - If the number of returned rows is zero, the result value is NULL.
  - If the number of returned rows is one, the result value is a value contained in the row.
  - If the number of returned rows is two or more, an exception error occurs.

<row subquery>

- The number of target in <query expression> should be two or more.
- The result value according to the number of rows returned from <query expression> is as follows.
  - If the number of returned rows is zero, the result value is a row all of whose columns are NULL.
  - If the number of returned rows is one, the result value is that row.
  - If the number of returned rows is two or more, an exception error occurs.

〈table subquery〉

- The number of target in 〈query expression〉 should be one or more.
- The result according to the number of rows returned from 〈query expression〉 is as follows.
  - If the number of returned rows is zero, the result value is *no rows*.
  - If the number of returned rows is one or more, the result value is that row.

Description

〈scalar subquery〉

〈scalar subquery〉 returns one row which has one column as a result. The target of 〈scalar subquery〉 should be only one, and the result data type depends on the data type of the target.

〈scalar subquery〉 can be used alone in the target of 〈select list〉, and it can be used in the operator which has a single column.

〈row subquery〉

〈row subquery〉 returns one row which has two or more columns as a result. The targets of 〈row subquery〉 should be two or more, and the result data type depends on the data type of each target.

〈row subquery〉 can not be used alone in the target of 〈select list〉, and it can only be used in the row operator which has two or more columns.

〈table subquery〉

〈table subquery〉 returns one or more rows which have one or more columns as a result. The targets of 〈table subquery〉 should be one or more, and the result data type depends on the data type of each target.

〈table subquery〉 can not be used alone in the target of 〈select list〉, but it can be used in the operators such as IN, NOT IN, EXISTS, NOT EXISTS, quantify operator.

Examples

The following is an example of SELECT statement which uses 〈scalar subquery〉.

```
gSQL> SELECT (SELECT c_name FROM dual)  FROM customer;
(SELECT C_NAME FROM DUAL)
-------------------------
Customer#1
Customer#2
Customer#3
Customer#4
```

```
Customer#5
5 rows selected.
gSQL> SELECT c_name, c_nation FROM customer WHERE c_nation = (SELECT 'CANADA' FROM dual);
C_NAME     C_NATION
---------- --------
Customer#2 CANADA
1 row selected.
```

The following is an example of SELECT statement which uses <row subquery>.

```
gSQL> SELECT p_name, p_brand, p_type FROM part WHERE (p_brand, p_type) = (SELECT 'Brand#1',
'NICKEL' FROM dual);
P_NAME P_BRAND    P_TYPE
------ ---------- ------
Part#2 Brand#1    NICKEL
1 row selected.
```

The following is an example of SELECT statement which uses <table subquery>.

```
gSQL> SELECT s_name, s_nation FROM supplier WHERE s_nation IN (SELECT c_nation FROM customer);
S_NAME                    S_NATION
------------------------- -------------
Supplier#2                KOREA
Supplier#3                GERMANY
Supplier#4                UNITED STATES
Supplier#5                CANADA
4 rows selected.
gSQL> SELECT * FROM (SELECT s_name, s_nation FROM supplier);
S_NAME                    S_NATION
------------------------- -------------
Supplier#1                FRANCE
Supplier#2                KOREA
Supplier#3                GERMANY
Supplier#4                UNITED STATES
Supplier#5                CANADA
5 rows selected.
```

Compatibility

Table 20-20 SQL standard compatibility

Feature ID	Description	Compatibility
F471	Scalar subquery values	O
F641	Row and table constructors	X
T501	Enhanced EXISTS predicate	O
E061-11	Subqueries in IN predicate	O
E061-12	Subqueries in quantified comparison predicate	O
E061-12	Correlated subqueries	O

For More Information

Refer to the followings.

- **from clause**
- **where clause**

hint clause

It specifies a hint to be used for a query execution.
For more information, refer to **SQL Hint**.

20.17 SELECT .. FOR UPDATE

Function

It sets whether or not to update the result set of SELECT statement.

Syntax

```
<select for update statement> ::=
    <query expression>  <updatability clause>
    ;
<updatability clause> ::=
      FOR READ ONLY
    ¦ FOR UPDATE [ OF <column name list> ] [ <lock wait mode> ]
<lock wait mode> ::=
    ¦ WAIT
    ¦ WAIT second
    ¦ NOWAIT
```

Invocation and Access Rules

The user should satisfy the following conditions to perform <select for update statement>.

- One of the following privileges for all tables used in the statement is required for a user to perform <query expression>.
  - SELECT(columns) ON TABLE for all columns used in the statement among the table columns
  - (SELECT or CONTROL TABLE) ON TABLE for that table
  - (SELECT TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - SELECT ANY TABLE ON DATABASE

- If FOR UPDATE clause is used, one of the following privileges for the tables to be locked is required.
  - (LOCK or CONTROL TABLE) ON TABLE for that table
  - (LOCK TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - LOCK ANY TABLE ON DATABASE

Syntax Rules and Parameters

〈query expression〉

INTO clause should not exist in SELECT statement.

To use FOR UPDATE, the query should identify the row updates of the base table, or it should be an upda table query which can acquire the lock into the row.

The updatable query should satisfy all of following conditions.

- DISTINCT should not exist in the top-level query.
  - (X) SELECT DISTINCT * FROM t1;
- GROUP BY, HAVING, aggregation function should not exist in the top-level query.
  - (X) SELECT MAX(c1) FROM t1;
- Set operators should not exist.
  - (X) SELECT * FROM t1 UNION ALL SELECT * FROM t2;
- There should be at least one updatable column in the table listed in FROM clause.
  - The column of the table which is not for cross join among the tables included in join is not an up datable column.
    - OUTER JOIN is not the cross join.
    - NATURAL JOIN is not the cross join.
    - If USING clause is is used in INNER JOIN, it is not the cross join.
  - The column of the following tables is not an updatable column.
    - Dictionary table, fixed table, performance view
  - The column of a view is not an updatable table.

For more information about SELECT statement, refer to **query expression**.

〈updatability clause〉

It specifies whether or not to update the row for the result set.

- FOR READ ONLY
  - The read-only query is declared.
- FOR UPDATE
  - The writable query is declared.
  - x lock is acquired for the rows until the end of the transaction to prevent other transactions from updating the rows when executing the query.
  - 〈query expression〉 should be an updatable query.

FOR UPDATE OF …

It lists the columns relating to acquiring lock when executing the query.

- The column listed in FOR UPDATE OF statement.
  - It should be updatable columns of the table listed in the FROM clause of <query expression>.
  - It acquires a lock for the table of the listed column.
- Only FOR UPDATE is used
  - It means the same as listing all updatable columns of the table in FROM clause of <query expression>.
  - It acquires a lock for the table of all columns.

<lock wait mode>

It is used together with FOR UPDATE statement, and it specifies the lock acquisition method.

- WAIT
  - It acquires a lock for all rows of the query result before obtaining the query result.
  - It waits until acquiring a lock.
- WAIT second
  - It acquires a lock for all rows of the query result before acquiring the query result.
  - If the lock is not acquired for a specified time, an error occurs.
  - The wait time is in seconds and it can use the value between 0 and 1,000,000,000.
- NOWAIT
  - It acquires a lock for all rows of the query result before acquiring the query result.
  - If the lock is not immediately acquired, an error occurs.
- If it is not specified, the default value is WAIT.

Description

SELECT statement keep fetching the rows regardless of whether the transaction ends. However, SELECT .. FOR UPDATE statement can not fetch the rows when the transaction ends because the statement acquires the lock for the rows.

Cursor holdability

- WITH HOLD
  - It can keep fetching regardless of whether the transaction ends.
  - It is also known as fetch across commit.

- WITHOUT HOLD
  - When the transaction ends, it can not fetch.

Examples

The following is an example of acquiring a lock for the row by using FOR UPDATE statement.

```
gSQL> SELECT id, data FROM t1 WHERE id = 3 FOR UPDATE;
ID DATA
-- ------
 3 data_3
1 row selected.
```

The following uses join and ORDER BY clause but it is an updatable query, so FOR UPDATE statement can be used.

```
gSQL> SELECT t1.id, t1.name, t2.addr
        FROM t1, t2
       WHERE t1.id = t2.id
       ORDER BY 1
         FOR UPDATE;
ID NAME    ADDR
-- ------- -------------
 1 someone somewhere
 2 anyone  anywhere
 3 unknown N/A
 4 leekmo  leekmo's home
 5 mkkim   seoul
5 rows selected.
```

The following is a non-updatable query, so FOR UPDATE statement can not be used.

```
gSQL> SELECT id, COUNT(*)
        FROM t1
       GROUP BY id
         FOR UPDATE;
ERR-42000(16112): query expression is not updatable
```

Compatibility

In the SQL standard, <select for update statement> is not defined, but it can be defined by using **DECLARE cursor_name** statement.

20.18 SELECT .. INTO

Function

It retrieves a single row by using a query, then obtains the value of retrieved row into the host variable.

Syntax

```
<select statement: single row> ::=
    SELECT [ <hint clause> ] [ <set quantifier> ] <select list>
        INTO <select target list>
        <table expression>
    ;
<select target list> ::=
    variable_name [, ...]
```

Invocation and Access Rules

One of the following privileges for all tables used in the statement is required for a user to perform <select statement: single row>.

- SELECT(columns) ON TABLE for all columns used in the statement among the table columns
- (SELECT or CONTROL TABLE) ON TABLE for that table
- (SELECT TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- SELECT ANY TABLE ON DATABASE

Syntax Rules and Parameters

<hint clause>

It specifies hints for query execution.
For more information, refer to **hint clause** of **SELECT** statement.

〈set quantifier〉

It specifies whether to remove duplicates from the query result.
For more information, refer to **query specification** clause.

〈select list〉

It specifies the columns to be retrieved from the query result.
For more information, refer to **select list** clause.

INTO 〈select target list〉

The number of the variable specified in INTO clause should be equal to the number of the expression spe cified in 〈select list〉.

〈table expression〉

It specifies the query information such as a search condition.
For more information, refer to **query specification** clause.

Description

The rows to be retrieved should be one or less.
If two or more rows are retrieved, an error occurs.

Differences among SELECT-related Statements

- 〈select statement〉
  - It retrieves multiple rows which satisfy the condition, and the retrieved rows can be retrieved by u sing API such as SQLFetch ().
  - e.g. SELECT c1 FROM t1 WHERE c1 > 0;
- 〈select statement: single row〉
  - It can retrieve one or less row which satisfies the condition, then obtains the value into the host v ariable in INTO clause when the retrieved row is a single row.
  - e.g. SELECT c2 INTO :v1 FROM t1 WHERE c1 = 0;

Example

The following is an example of obtaining the value into the host variable by using interactive SQL (gsql).

```
gSQL> \var v_id   INTEGER
gSLQ> \var v_data VARCHAR(128)
gSQL> SELECT id, data INTO :v_id, :v_data FROM t1 WHERE id = 3;
V_ID V_DATA
---- ------
   3 data_3
1 row selected.
```

20.19 SELECT .. INTO .. FOR UPDATE

Function

It sets whether to update the row by retrieving a single row through the query, then obtains the value of retrieved row into the host variable.

Syntax

```
<select for update statement: single row> ::=
    SELECT [ <hint clause> ] [ <set quantifier> ] <select list>
        INTO <select target list>
        <table expression>  <updatability clause>
    ;
<select target list> ::=
    variable_name [, ...]
<updatability clause> ::=
      FOR READ ONLY
    | FOR UPDATE [ OF <column name list> ] [ <lock wait mode> ]
<lock wait mode> ::=
    | WAIT
    | WAIT second
    | NOWAIT
```

Invocation and Access Rules

One of the following privileges for all tables used in the statement is required for a user to perform <select statement: single row>.

- SELECT(columns) ON TABLE for all columns used in the statement among the table columns
- (SELECT or CONTROL TABLE) ON TABLE for that table
- (SELECT TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- SELECT ANY TABLE ON DATABASE

If FOR UPDATE clause is used, one of the following privileges for the tables to be locked is required.

- (LOCK or CONTROL TABLE) ON TABLE for that table
- (LOCK TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- LOCK ANY TABLE ON DATABASE

Syntax Rules and Parameters

〈select for update statement: single row〉

To use FOR UPDATE, the query should identify the row updates of the base table, or it should be an upda table query which can acquire the lock into the row.

The updatable query should satisfy all of following conditions.

- DISTINCT should not exist in the top-level query.
  - (X) SELECT DISTINCT * FROM t1;
- GROUP BY, HAVING, aggregation function should not exist in the top-level query.
  - (X) SELECT MAX(c1) FROM t1;
- Set operators should not exist.
  - (X) SELECT * FROM t1 UNION ALL SELECT * FROM t2;
- There should be at least one updatable column in the table listed in FROM clause.
  - The column of the table which is not for cross join among the tables included in join is not an up datable column.
    - FULL OUTER JOIN is not the cross join.
    - NATURAL JOIN is not the cross join.
    - If USING clause is is used in INNER JOIN, it is not the cross join.
  - The column of the following tables is not an updatable column.
    - Dictionary table, fixed table, performance view
  - The column of a view is not an updatable table.

〈updatability clause〉

It specifies whether or not to update the row for the result set.

- FOR READ ONLY
  - The read-only query is declared.
- FOR UPDATE
  - The writable query is declared.
  - x lock is acquired for the rows until the end of the transaction to prevent other transactions from updating the rows when executing the query.
  - 〈query expression〉 should be an updatable query.

FOR UPDATE OF …

It lists the columns relating to acquiring lock when executing the query.

- The column listed in FOR UPDATE OF statement.
  - It should be updatable columns of the table listed in the FROM clause of 〈query expression〉.
  - It acquires a lock for the table of the listed column.
- Only FOR UPDATE is used
  - It means the same as listing all updatable columns of the table in FROM clause of 〈query expression〉.
  - It acquires a lock for the table of all columns.

〈lock wait mode〉

It is used together with FOR UPDATE statement, and it specifies the lock acquisition method.

- WAIT
  - It acquires a lock for all rows of the query result before obtaining the query result.
  - It waits until acquiring a lock.
- WAIT second
  - It acquires a lock for all rows of the query result before acquiring the query result.
  - If the lock is not acquired for a specified time, an error occurs.
  - The wait time is in seconds and it can use the value between 0 and 1,000,000,000.
- NOWAIT
  - It acquires a lock for all rows of the query result before acquiring the query result.
  - If the lock is not immediately acquired, an error occurs.
- If it is not specified, the default value is WAIT.

〈hint clause〉

It specifies hints for query execution.
For more information, refer to **hint clause** of **SELECT** statement.

〈set quantifier〉

It specifies whether to remove duplicates from the query result.
For more information, refer to **query specification** clause.

〈select list〉

It specifies the columns to be retrieved from the query result.
For more information, refer to **select list** clause.

INTO 〈select target list〉

The number of the variable specified in INTO clause should be equal to the number of the expression spe cified in 〈select list〉.

〈table expression〉

It specifies the query information such as a search condition.
For more information, refer to **query specification** clause.

Description

The rows to be retrieved should be one or less.
If two or more rows are retrieved, an error occurs.

SELECT statement keep fetching the rows regardless of whether the transaction ends. However, SELECT .. FOR UPDATE statement can not fetch the rows when the transaction ends because the statement acquir es the lock for the rows.

Cursor holdability

- WITH HOLD
  - It can keep fetching regardless of whether the transaction ends.
  - It is also known as fetch across commit.
- WITHOUT HOLD
  - When the transaction ends, it can not fetch.

Differences among SELECT-related Statements

- 〈select for update statement〉
  - It retrieves multiple rows which satisfy the condition, sets whether to update them and the retriev ed rows can be retrieved by using API such as SQLFetch ().
  - e.g. SELECT c1 FROM t1 WHERE c1 > 0 FOR UPDATE;
- 〈select for update statement: single row〉
  - It can retrieve one or less row which satisfies the condition, sets whether to update them then ob tains the value into the host variable in INTO clause when the retrieved row is a single row.
  - e.g. SELECT c2 INTO :v1 FROM t1 WHERE c1 = 0 FOR UPDATE;

Examples

The following is an example of acquiring a lock for the row by using FOR UPDATE statement, and obtaini ng the value into the host variable by using interactive SQL (gsql).

```
gSQL> \var v_id   INTEGER
gSQL> \var v_data VARCHAR(128)
gSQL> SELECT id, data INTO :v_id, :v_data FROM t1 WHERE id = 3 FOR UPDATE;
V_ID V_DATA
---- ------
   3 data_3
1 row selected.
```

The following uses join and ORDER BY clause but it is an updatable query, so FOR UPDATE statement can be used.

```
gSQL> \var v_id   INTEGER
gSQL> \var v_name VARCHAR(128)
gSQL> \var v_addr VARCHAR(128)
gSQL> SELECT t1.id, t1.name, t2.addr
        INTO :v_id, :v_name, :v_addr
        FROM t1, t2
       WHERE t1.id = t2.id
       ORDER BY 1
       LIMIT 1
         FOR UPDATE;
ID NAME    ADDR
-- ------- -------------
 1 someone somewhere
```

```
1 row selected.
```

The following is a non-updatable query, so FOR UPDATE statement can not be used.

```
gSQL> \var v_id    INTEGER
gSQL> \var v_count INTEGER
gSQL> SELECT id, COUNT(*)
        INTO :v_id, :v_count
        FROM t1
       GROUP BY id
         FOR UPDATE;
ERR-42000(16112): query expression is not updatable
```

For More Information

Refer to the followings.

- SELECT .. FOR UPDATE
- SELECT .. INTO

20.20 SET CONSTRAINTS

Function

It sets the check point of deferrable constraint in a transaction to IMMEDIATE or DEFERRED.

Syntax

```
<set constraints mode statement> ::=
    SET { CONSTRAINT ¦ CONSTRAINTS } <constraint name list> { DEFERRED ¦ IMMEDIATE }
    ;
<constraint name list> ::=
      ALL
    ¦ <constraint name> [, ...]
```

Invocation and Access Rules

Any separate access privilege is not required for a user to perform SET CONSTRAINTS.


It is not supported in the cluster system.

Syntax Rules and Parameters

CONSTRAINT | CONSTRAINTS

CONSTRAINT and CONSTRAINTS are the keywords of the same meaning, and the SQL standard uses CONSTRAINTS.

<constraint name list>

It specifies the list of constraint names, or specifies ALL to set all deferrable constraints.
When specifying <constraint name>, it should be the name of the deferrable constraint.

ALL means all deferrable constraints.

DEFERRED | IMMEDIATE

It sets the check point of specified deferrable constraints.

- IMMEDIATE
  - It checks the specified constraints when executing the DML statement.
  - If the transaction violates the constraints, then an error occurs.
- DEFERRED
  - It checks the specified constraints when the transaction is committed.

If the transaction is in progress, the check point of the constraint is set in the current transaction. If the tr ansaction is not in progress, it is set in the next transaction.
After the transaction ends, it does not affect the next transaction.

Description

Deferrable Constraint

DEFERRABLE constraint can change its check point.
The following is an example of creating a table with a deferrable constraint, and inserting data to the tabl e.

```
gSQL> CREATE TABLE t1
(
    id   INTEGER,
    name VARCHAR(128) CONSTRAINT t1_uk UNIQUE
                      DEFERRABLE INITIALLY IMMEDIATE
);
Table created.
gSQL> COMMIT;
Commit complete.
gSQL> INSERT INTO t1 VALUES ( 1, 'leekmo' );
1 row created.
gSQL> INSERT INTO t1 VALUES ( 2, 'mkkim' );
1 row created.
gSQL> COMMIT;
Commit complete.
```

In the example above, UNIQUE constraint which is deferrable is created on a name column, and the initial check point is set as INITIALLY IMMEDIATE. Therefore, the constraint is checked whenever DML statemen t is executed.

In this case, if the user tries to exchange the name value of two rows as follows, it violates the constraint because the check point is IMMEDIATE.

```
gSQL> UPDATE t1 SET name = 'mkkim' WHERE id = 1;
ERR-23000(16057): unique constraint (PUBLIC.T1_UK) violated
gSQL> UPDATE t1 SET name = 'leekmo' WHERE id = 2;
ERR-23000(16057): unique constraint (PUBLIC.T1_UK) violated
```

If the check point is changed to DEFERRED as follows, the operation as same as above succeeds because the constraint is checked when the transaction is committed.

```
gSQL> SET CONSTRAINTS t1_uk DEFERRED;
Constraints set.
gSQL> UPDATE t1 SET name = 'mkkim' WHERE id = 1;
1 row updated.
gSQL> UPDATE t1 SET name = 'leekmo' WHERE id = 2;
1 row updated.
gSQL> COMMIT;
Commit complete.
```

If the check point is set to DEFERRED, then the constraint is checked when the transaction is committed. Therefore, if the transaction is committed when the constraint is violated, then the transaction fails and it is rolled back as follows.

```
gSQL> SET CONSTRAINTS t1_uk DEFERRED;
Constraints set.
gSQL> INSERT INTO t1 VALUES ( 3, 'leekmo' );
1 row created.
gSQL> COMMIT;
ERR-40002(16291): transaction rollback: integrity constraint violation : PUBLIC.T1_UK(1)
```

Violation of a Deferred Constraint

Executing the following statements when the transaction violates the constraints set to DEFFFERED, then an error occurs as follows.

- COMMIT
  - An error occurs and the transaction is rolled back.

- SET CONSTRAINTS ALL IMMEDIATE
  - An syntax error occurs.
- DDL
  - An syntax error occurs.

An unexpected ROLLBACK can occur when COMMIT, it is necessary to ensure whether the transaction vi olates the constraint by using SET CONSTRAINTS ALL IMMEDIATE statement.

```
gSQL> SET CONSTRAINTS t1_uk DEFERRED;
Constraints set.
gSQL> INSERT INTO t1 VALUES ( 3, 'leekmo' );
1 row created.
gSQL> SET CONSTRAINTS ALL IMMEDIATE;
ERR-23000(16038): integrity constraint violation : PUBLIC.T1_UK(1)
gSQL> SELECT * FROM t1 ORDER BY id;
ID NAME
-- ------
 1 mkkim
 2 leekmo
 3 leekmo
3 rows selected.
gSQL> UPDATE t1 SET name = 'xcom73' WHERE id = 3;
1 row updated.
gSQL> SET CONSTRAINTS ALL IMMEDIATE;
Constraints set.
gSQL> COMMIT;
Commit complete.
```

Transaction Control Language

SET CONSTRAINTS statement is a transaction control language which is used when the transaction is in p rogress such as **SAVEPOINT savepoint_specifier**.
The transaction control such as COMMIT, ROLLBACK, ROLLBACK TO SAVEPOINT statement is applied to SET CONSTRAINTS statement.

The following is an example of a table with multiple deferrable constraints.

```
CREATE TABLE t1
(
   id1 INTEGER CONSTRAINT t1_uk1 UNIQUE DEFERRABLE INITIALLY IMMEDIATE,
   id2 INTEGER CONSTRAINT t1_uk2 UNIQUE DEFERRABLE INITIALLY IMMEDIATE,
```

```
   id3 INTEGER CONSTRAINT t1_uk3 UNIQUE DEFERRABLE INITIALLY IMMEDIATE
);
```

If <set constraints mode statement> statement is performed when the transaction is in progress, the chec k point of deferrable constraints is changed depending on each point as follows.

- result: success

```
INSERT INTO t1 VALUES ( 1, 1, 1 );
1 row created.
COMMIT;
Commit complete.
```

- result: success

```
SAVEPOINT sp1;
Savepoint created.
```

- result: success
- t1_uk1 constraint is DEFERRED

```
SET CONSTRAINTS t1_uk1 DEFERRED;
Constraints set.
```

- result: success

```
SAVEPOINT sp2;
Savepoint created.
```

- result: success
- t1_uk1, t1_uk2 constraints are DEFERRED

```
SET CONSTRAINTS t1_uk2 DEFERRED;
Constraints set.
```

- result: success

```
SAVEPOINT sp3;
Savepoint created.
```

- result: success
- ALL constraints are DEFERRED

```
SET CONSTRAINTS ALL DEFERRED;
Constraints set.
```

- result: success

```
SAVEPOINT sp4;
Savepoint created.
```

- result: success
- ALL constraints are IMMEDIATE

```
SET CONSTRAINTS ALL IMMEDIATE;
Constraints set.
```

When the transaction is partially rolled back by using ROLLBACK TO SAVEPOINT statement as follows, SET CONSTRAINTS statement is also partially rolled back and the check point is changed.

- result: error

```
INSERT INTO t1 VALUES ( 1, 2, 2 );
ERR-23000(16057): unique constraint (PUBLIC.T1_UK1) violated
```

- result: error

```
INSERT INTO t1 VALUES ( 3, 1, 3 );
ERR-23000(16057): unique constraint (PUBLIC.T1_UK2) violated
```

- result: error

```
INSERT INTO t1 VALUES ( 4, 4, 1 );
ERR-23000(16057): unique constraint (PUBLIC.T1_UK3) violated
```

- result: success
- t1_uk1, t1_uk2 constraints are DEFERRED

```
ROLLBACK TO SAVEPOINT sp4;
Rollback complete.
```

- result: success

```
INSERT INTO t1 VALUES ( 1, 2, 2 );
1 row created.
```

- result: success

```
INSERT INTO t1 VALUES ( 3, 1, 3 );
1 row created.
```

- result: success

```
INSERT INTO t1 VALUES ( 4, 4, 1 );
1 row created.
```

- result: success
- t1_uk1, t1_uk2 constraints are DEFERRED

```
ROLLBACK TO SAVEPOINT sp3;
Rollback complete.
```

- result: success

```
INSERT INTO t1 VALUES ( 1, 2, 2 );
1 row created.
```

- result: success

```
INSERT INTO t1 VALUES ( 3, 1, 3 );
1 row created.
```

- result: success

```
INSERT INTO t1 VALUES ( 4, 4, 1 );
ERR-23000(16057): unique constraint (PUBLIC.T1_UK3) violated
```

- result: success
- t1_uk1 constraint is DEFERRED

```
ROLLBACK TO SAVEPOINT sp2;
Rollback complete.
```

- result: success

```
INSERT INTO t1 VALUES ( 1, 2, 2 );
1 row created.
```

- result: error

```
INSERT INTO t1 VALUES ( 3, 1, 3 );
ERR-23000(16057): unique constraint (PUBLIC.T1_UK2) violated
```

- result: error

```
INSERT INTO t1 VALUES ( 4, 4, 1 );
ERR-23000(16057): unique constraint (PUBLIC.T1_UK3) violated
```

- result: success
- all constraint are IMMEDIATE

```
ROLLBACK TO SAVEPOINT sp1;
Rollback complete.
```

- result: error

```
INSERT INTO t1 VALUES ( 1, 2, 2 );
ERR-23000(16057): unique constraint (PUBLIC.T1_UK1) violated
```

- result: error

```
INSERT INTO t1 VALUES ( 3, 1, 3 );
ERR-23000(16057): unique constraint (PUBLIC.T1_UK2) violated
```

- result: error

```
INSERT INTO t1 VALUES ( 4, 4, 1 );
ERR-23000(16057): unique constraint (PUBLIC.T1_UK3) violated
```

- result: 1 row
- 1 1 1

```
SELECT * FROM t1;
ID1 ID2 ID3
--- --- ---
  1   1   1
1 row selected.
```

When the transaction is committed or rolled back, the effects of SET CONSTRAINTS statement is also ter minated, and all deferrable constraints follows the constraints property which is INITIALLY IMMEDIATE or INITIALLY DEFERRED value.

Examples

The following is an example of changing the check point by specifying the constraint name.

```
gSQL> SET CONSTRAINTS t1_uk1 DEFERRED;
Constraints set.
```

The following is an example of changing the check point of all deferrable constraints.

```
gSQL> SET CONSTRAINTS ALL DEFERRED;
Constraints set.
```

Compatibility

The SQL standard does not define CONSTRAINT keyword clause.

Table 20-21 SQL standard compatibility

Feature ID	Description	Compatibility
F721	Deferrable constraints	O

For More Information

Refer to the followings.

- Adding constraints
  - **CREATE TABLE**
  - **ALTER TABLE name ADD CONSTRAINT**
  - **ALTER TABLE name ADD COLUMN**
  - **ALTER TABLE name ALTER COLUMN**
- Altering constraints: **ALTER TABLE name ALTER CONSTRAINT**
- Controlling the check point of constraints: **SET CONSTRAINTS**

20.21 SET SCHEMA schema_name

Function

It sets the default schema name to be used in the current session.

Syntax

```
<set schema statement> ::=
    SET SCHEMA schema_name
    ;
```

Invocation and Access Rules

N/A

Syntax Rules and Parameters

schema_name

It is the default schema name to be set in the current session.

Description

It sets the default schema name to be used in the current session.
If the schema name of the object is not specified, then it becomes the default schema name to be used in the current session.

- When accessing as the user u1, then retrieve R relation by using the schema path of user u1.

```
% gsql u1 u1
gsql> SELECT * FROM r;
```

- When SET SCHEMA statement is set, then then retrieve R relation by using NEW_SCHEMA.R.

```
gsql> SET SCHEMA new_schema;
gsql> SELECT * FROM r;
```

Examples

The following is an example of when user u1 has schema s1 and s2.

```
CREATE USER u1 IDENTIFIED BY u1 WITHOUT SCHEMA;
CREATE SCHEMA s1 AUTHORIZATION u1;
CREATE SCHEMA s2 AUTHORIZATION u1;
COMMIT;
ALTER USER u1 SCHEMA PATH ( s1, s2 );
GRANT ALL PRIVILEGES TO u1;
COMMIT;
CREATE TABLE s1.t1 ( c1 VARCHAR(32) );
INSERT INTO s1.t1 VALUES ( 'S1.T1' );
COMMIT;
CREATE TABLE s2.t1 ( c1 VARCHAR(32) );
INSERT INTO s2.t1 VALUES ( 'S2.T1' );
COMMIT;
```

It retrieves table S1.T1 by interpreting table t1 using schema path of user u1 when accessing for the first t ime.

```
% gsql u1 u1
gSQL> SELECT current_schema FROM dual;
CURRENT_SCHEMA
--------------
S1
1 row selected.
gSQL> SELECT * FROM t1;
C1
-----
S1.T1
1 row selected.
```

It retrieves table S2.T1 by interpreting table t1 using schema name of the session after SET SCHEMA state ment is used.

```
gSQL> SET SCHEMA s2;
Session set.
gSQL> SELECT current_schema FROM dual;
CURRENT_SCHEMA
--------------
S2
1 row selected.
gSQL> SELECT * FROM t1;
C1
-----
S2.T1
1 row selected.
```

Compatibility

Table 20-22 SQL standard compatibility

Feature ID	Description	Compatibility
F761	Session management	O

20.22 SET SESSION AUTHORIZATION user_identifier

Function

It sets the session user and current user.

Syntax

```
<set session user identifier statement> ::=
    SET SESSION AUTHORIZATION user_identifier
    ;
```

Invocation and Access Rules

ACCESS CONTROL ON DATABASE privilege is required for a logon user to perform <set session user identifier statement>.

The user information is managed in three types as follows.

- Logon user
  - It is a user who performed login, and it is maintained until the connection is closed.
- Session user
  - It is as same as the first logon user, but it can be changed using the SET SESSION AUTHORIZATION statement.
- Current user
  - It is generally as same as the session user, but it is temporarily changed internally in system to control access when using the PSM or view.
  - The session user and current user is similar to the difference between the unix system's real user and the effective user.

Syntax Rules and Parameters

user_identifier

It is the username to be altered.

Description

After performing SET SESSION AUTHORIZATION statement, all statements is performed based on the sess ion user. Therefore, the privilege for the session user is checked and the owner of when creating objects also is the session user.

Example

The following is an example that the user test with ACCESS CONTROL ON DATABASE privilege sets the u ser u1 to the session user.

```
gSQL> SET SESSION AUTHORIZATION u1;
Session set.
gSQL> SELECT LOGON_USER(), SESSION_USER(), CURRENT_USER FROM dual;
LOGON_USER() SESSION_USER() CURRENT_USER
------------ -------------- ------------
TEST         U1             U1
1 row selected.
```

Compatibility

Table 20-23 SQL standard compatibility

Feature ID	Description	Compatibility
F321	User authorization	O

20.23 SET SESSION CHARACTERISTICS AS transaction_mode

Function

It sets the transaction property of a session.

Syntax

```
<set session characteristics statement> ::=
    SET SESSION CHARACTERISTICS AS TRANSACTION <transaction_mode>
    ;
<transaction_mode> ::=
    { <transaction_access_mode> | ISOLATION LEVEL < isolation_level > }
<transaction_access_mode> ::=
    READ { ONLY | WRITE }
< isolation_level > ::=
    { READ COMMITTED | SERIALIZABLE }
```

Syntax Rules and Parameters

<transaction_access_mode>

It is ACCESS MODE of the following transactions.

- READ ONLY
- READ WRITE

<isolation_level>

It is ISOLATION LEVEL of the following transactions.

- READ COMMITTED
- SERIALIZABLE

Description

SET SESSION CHARACTERISTICS sets the transaction property of a session. In other words, properties of a ll transactions created within the session follows these properties.

However, **SET TRANSACTION transaction_mode** statement sets only the property of a single transaction which is performed next.

Examples

The following is an example that all transactions to be created within the session are set to READ ONLY.

```
gSQL> SET SESSION CHARACTERISTICS AS TRANSACTION READ ONLY;
Session set.
```

The following is an example that the isolation level of all transactions to be created within the session is s et to READ COMMITTED.

```
gSQL> SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION LEVEL READ COMMITTED;
Session set.
```

Compatibility

Table 20-24 SQL standard compatibility

Feature ID	Description	Compatibility
F761	Session management	O

For More Information

Refer to **SET TRANSACTION transaction_mode**.

20.24 SET TIME ZONE

Function

It sets the TIMEZONE of a session.

Syntax

```
<set local time zone statement> ::=
    SET TIME ZONE <set time zone value>
    ;
<set time zone value> ::=
    { '[+|-]hh:mm' | LOCAL }
```

Syntax Rules and Parameters

<set time zone value>

It is the TIMEZONE value to be set.

- hh:mm: It is a GMT OFFSET of the TIMEZONE to be set.
  - The range of the offset value is '-14:00' ~ '+14:00' .
- LOCAL: It is the TIME ZONE at the time of session creation.
  - TIME ZONE at the time of session creation is set to TIME ZONE of the client OS.

Description

Altering the time zone of the session affects the result value of function such as **CURRENT_TIME**, **CURRENT_TIMESTAMP**.

Example

The following is an example of altering the session time zone to '+09: 00'.

```
gSQL> SET TIME ZONE '+09:00';
Session set.
```

Compatibility

Table 20-25 SQL standard compatibility

Feature ID	Description	Compatibility
F411	Time zone specifications	O

20.25 SET TRANSACTION transaction_mode

Function

It sets the transaction property.

Syntax

```
<set transaction statement> ::=
    SET TRANSACTION <transaction_mode>
    ;
<transaction_mode> ::=
    { <transaction_access_mode> | ISOLATION LEVEL < isolation_level > }
<transaction_access_mode> ::=
    READ { ONLY | WRITE }
< isolation_level > ::=
    { READ COMMITTED | SERIALIZABLE }
```

Syntax Rules and Parameters

<transaction_access_mode>

It is ACCESS MODE of the following transactions.

- READ ONLY
- READ WRITE

<isolation_level>

It is ISOLATION LEVEL of the following transactions.

- READ COMMITTED
- SERIALIZABLE

Description

SET TRANSACTION sets property of the next transaction, and the property is reset to the default value aft
er the next transaction ends.

Example

The following is an example of setting the next transaction to READ ONLY.

```
gSQL> SET TRANSACTION READ ONLY;
Transaction set.
```

Compatibility

Table 20-26 SQL standard compatibility

Feature ID	Description	Compatibility
T251	SET TRANSACTION statement: LOCAL option	X

For More Information

Refer to **SET SESSION CHARACTERISTICS AS transaction_mode**.

20.26 TRUNCATE TABLE

Function

It truncates all rows from a table.

Syntax

```
<truncate table statement> ::=
    TRUNCATE TABLE table_name
        [ RESTART IDENTITY ¦ CONTINUE IDENTITY ]
        [ DROP STORAGE ¦ DROP ALL STORAGE ]
    ;
```

Invocation and Access Rules

One of the following privileges is required for a user to perform <truncate table statement>.

- The owner of that table
- CONTROL TABLE ON TABLE for the table
- (DROP TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- DROP ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the name of a target table whose rows are to be truncated.
It can define schema to which the table belongs such as schema_name.table_name and if schema_name is omitted, the default schema name of the user performing the statement is used.

[RESTART IDENTITY | CONTINUE IDENTITY]

- RESTART IDENTITY
  - If an identity column which has auto created value in that table exists, it automatically restarts val ue.
- CONTINUE IDENTITY
  - If an identity column which has auto created value in that table exists, it does not change the exis ting value.
- If it is not specified, the default value is CONTINUE IDENTITY.

[DROP STORAGE | DROP ALL STORAGE]

- DROP STORAGE
  - It drops allocated extents from the the table excluding the space of MINSIZE.
- DROP ALL STORAGE
  - It drops all extents allocated to the table.
- If it is not specified, the default value is DROP STORAGE.

Description

Data Definition Language (DDL) statement such as TRUNCATE TABLE can be rolled back if it is before wh en the transaction is committed.

Examples

The following is an example of performing TRUNCATE TABLE statement.

```
gSQL> TRUNCATE TABLE t1;
Table truncated.
```

The following is an example of restarting the value of the identity column when performing TRUNCATE T ABLE.

```
TRUNCATE TABLE t1 RESTART IDENTITY;
Table truncated.
```

Compatibility

The SQL standard does not define [DROP STORAGE | DROP ALL STORAGE] clause.

Table 20-27 SQL standard compatibility

Feature ID	Description	Compatibility
F200	TRUNCATE TABLE statement	O
F202	TRUNCATE TABLE: identity column restart option	O

20.27 UPDATE

Function

It updates rows in a table.

Syntax

```
<update statement: searched> ::=
    UPDATE table_name [ [ AS ] alias_name ]
        SET <set clause> [, ...]
        [ WHERE <search condition> ]
        [ <result offset clause> ]
        [ <fetch limit clause> ]
    ;
<set clause> ::=
      column_name = { <value expression> | DEFAULT }
    | ( column_name [, ...] ) = ( { <value expression> | DEFAULT } [, ...] )
    | ( column_name [, ...] ) = ( <query expression> )
<result offset clause> ::=
    OFFSET skip_count [ ROW | ROWS ]
<fetch limit clause> ::=
      <fetch first clause>
    | <limit clause>
<fetch first clause> ::=
    FETCH [ FIRST | NEXT ] [ row_count ] [ ROW ONLY | ROWS ONLY ]
<limit clause>
    LIMIT { fetch_row_count | offset_row_count, fetch_row_count | ALL }
```

Invocation and Access Rules

One of the following privileges is required for a user to perform <update statement: searched>.

- UPDATE(columns) ON TABLE for all columns which are targets to be updated
- (UPDATE or CONTROL TABLE) ON TABLE for the table

- (UPDATE TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
- UPDATE ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the name of a target table whose rows are to be updated.
It can define schema to which the table belongs such as schema_name.table_name and if schema_name is omitted, the default schema name of the user performing the statement is used.

[AS alias_name]

It is the alias of table_name.

〈set clause〉

It defines the columns to be updated and its values to be assigned. The number of columns in 〈set clause 〉 should be as same as the number of values.

It can be defined as follows.

- column_name = { 〈value expression〉 | DEFAULT }

```
UPDATE table_name
   SET column1 = value1, column2 = value2, column3 = value3
```

- (column_name [, ...]) = ({ 〈value expression〉 | DEFAULT } [, ...])

```
UPDATE table_name
   SET ( column1, column2, column3 ) = ( value1, value2, value3 )
```

- (column_name [, ...]) = (〈query expression〉)

```
UPDATE table_name
   SET column1 = ( SELECT max(value1) FROM other_table_name )
```

〈query expression〉 should be a query which creates a single row.

If DEFAULT is defined as a column value, the default values (refer to **〈default clause〉**) defined when exec uting **CREATE TABLE** is used. If it is not defined, NULL value is assigned.

WHERE 〈search condition〉

It updates the rows which satisfy WHERE condition.
If WHERE condition is not specified, all rows are updated.
For more information about WHERE condition, refer to **where clause** of **SELECT**.

〈result offset clause〉

It specifies the number of rows to skip from the query result.
For more information, refer to **〈result offset clause〉** of **SELECT**.

〈fetch limit clause〉

It specifies the number of rows to fetch in two ways, which are 〈fetch first clause〉 and 〈limit clause〉.

- 〈fetch first clause〉
  - It specifies the number of rows to be fetched.
  - For more information, refer to **〈fetch first clause〉** of **SELECT** statement.
- 〈limit clause〉
  - It specifies the number of rows to be fetched, or it simultaneously specifies both the number of r ows to be skipped and the number of rows to be fetched.
  - For more information, refer to **〈limit clause〉** of **SELECT** statement.

Description

Differences among UPDATE-related Statements

- **UPDATE**
  - It updates multiple rows which satisfy the condition.
  - e.g. UPDATE t1 SET c2 = c2 + 1 WHERE c1 > 0;
- **UPDATE name WHERE CURRENT OF cursor_name**
  - It updates the row which the current cursor indicates.
  - e.g. UPDATE t1 WHERE CURRENT OF cursor;
- **UPDATE name RETURNING**
  - It updates multiple rows which satisfy the conditions, and the updated rows can be retrieved in t he same way as **SELECT** statement (API such as SQLFetch ()).

    - e.g. UPDATE t1 SET c2 = c2 + 1 WHERE c1 > 0 RETURNING c2;
- **UPDATE name RETURNING .. INTO**
    - It updates row equal to or less than one, and if a single row is updated, it obtains the value to th e host variable of RETURNING INTO clause.
    - e.g. UPDATE t1 SET c2 = c2 + 1 WHERE c1 = 0 RETURNING c2 INTO :v1;

Examples

The following is an example of updating multiple rows which satisfy the condition.

```
gSQL> UPDATE lineitem
         SET l_shipdate = CURRENT_DATE
       WHERE l_returnflag = 'R';
5 rows updated.
```

The following is an example of updating the value of multiple columns.

```
gSQL> UPDATE lineitem
         SET l_shipdate   = CURRENT_DATE
           , l_returnflag = 'A'
       WHERE l_returnflag = 'R';
5 rows updated.
```

The following is an example of updating multiple columns by enclosing them with parentheses.

```
gSQL> UPDATE lineitem
         SET ( l_shipdate  , l_returnflag )
           = ( CURRENT_DATE, 'A' )
       WHERE l_returnflag = 'R';
5 rows updated.
```

The following is an example of updating the column value by using the subquery.

```
gSQL> UPDATE lineitem
         SET l_discount = ( SELECT MAX(l_discount) + 0.01 FROM lineitem )
       WHERE l_returnflag = 'R';
5 rows updated.
```

The following is an example of updating part of the rows which satisfy the condition by using OFFSET an d FETCH clauses.

```
gSQL> UPDATE lineitem
         SET l_discount = l_discount + 0.01
       WHERE l_returnflag = 'R'
      OFFSET 3
      FETCH 2;
2 rows updated.
```

Compatibility

The SQL standard does not define the following clauses in UPDATE statement.

- 〈result offset clause〉
- 〈fetch limit clause〉

Table 20-28 SQL standard compatibility

Feature ID	Description	Compatibility
F781	Self-referencing operations	X
T111	Updatable joins, unions, and columns	X

20.28 UPDATE name RETURNING

Function

It updates rows in a table, and retrieves the rows of before or after the update.

Syntax

```
<update statement: searched> ::=
    UPDATE table_name [ [ AS ] alias_name ]
        SET <set clause> [, ...]
        [ WHERE <search condition> ]
        [ <result offset clause> ]
        [ <fetch limit clause> ]
        <returning clause>
<set clause> ::=
      column_name = { <value expression> | DEFAULT }
    | ( column_name [, ...] ) = ( { <value expression> | DEFAULT } [, ...] )
    | ( column_name [, ...] ) = ( <query expression> )
<result offset clause> ::=
    OFFSET skip_count [ ROW | ROWS ]
<fetch limit clause> ::=
      <fetch first clause>
    | <limit clause>
<fetch first clause> ::=
    FETCH [ FIRST | NEXT ] [ row_count ] [ ROW ONLY | ROWS ONLY ]
<limit clause>
    LIMIT { fetch_row_count | offset_row_count, fetch_row_count | ALL }
<returning clause> ::=
    { RETURN | RETURNING } [ NEW | OLD ] { * | { <value expression> [ [AS] alias_name] } [,
...] }
```

Invocation and Access Rules

The user should satisfy the following conditions to perform <update returning query statement>.

- One of the following privileges is required to perform UPDATE statement.
  - UPDATE(columns) ON TABLE for all columns which are targets to be updated
  - (UPDATE or CONTROL TABLE) ON TABLE for the table
  - (UPDATE TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - UPDATE ANY TABLE ON DATABASE

- One of the following privileges for all columns used in RETURNING clause is required.
  - SELECT(columns) ON TABLE for all columns used in RETURNING clause
  - (SELECT or CONTROL TABLE) ON TABLE for the table
  - (SELECT TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - SELECT ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the name of a target table whose rows are to be updated.

[AS alias_name]

It is the alias of table_name.

<set clause>

It defines the columns to be updated and its values to be assigned. The number of columns in <set clause > should be as same as the number of values.
For more information, refer to **UPDATE**.

WHERE <search condition>

It updates the rows which satisfy WHERE condition.
If WHERE condition is not specified, all rows are updated.
For more information about WHERE condition, refer to **where clause** of **SELECT**.

〈result offset clause〉

It specifies the number of rows to skip from the query result.
For more information, refer to **〈result offset clause〉** of **SELECT**.

〈fetch limit clause〉

It specifies the number of rows to fetch in two ways, which are 〈fetch first clause〉 and 〈limit clause〉.

- 〈fetch first clause〉
  - It specifies the number of rows to be fetched.
  - For more information, refer to **〈fetch first clause〉** of **SELECT** statement.
- 〈limit clause〉
  - It specifies the number of rows to be fetched, or it simultaneously specifies both the number of r ows to be skipped and the number of rows to be fetched.
  - For more information, refer to **〈limit clause〉** of **SELECT** statement.

〈returning clause〉

It defines the updated rows as a result set, and specifies columns to be retrieved from the result set.

- RETURN and RETURNING are the keywords with the same meaning.
- NEW | OLD
  - NEW: It searches for updated rows based on the row after the update.
  - OLD: It searches for updated rows based on the row before the update.
  - If it is omitted, the default value is NEW.
- 〈value expression〉
  - It is as same as 〈select list〉 in SELECT statement, but aggregation can not be used.
- [[AS] alias_name]
  - It can name 〈value expression〉 by using AS clause.

Description

For more information, refer to **Differences among UPDATE-related Statements**.

Examples

The following is an example of obtaining values of the updated rows by using RETURNING clause.

```
gSQL> UPDATE lineitem
         SET l_discount = l_discount + 0.01
       WHERE l_returnflag = 'R'
   RETURNING l_orderkey, l_linenumber, l_discount;
L_ORDERKEY L_LINENUMBER L_DISCOUNT
---------- ------------ ----------
         8            1        .07
         9            2        .11
        12            5        .05
        15            1        .03
        16            2        .08
5 rows updated.
```

The following is an example of obtaining values before the update for the updated rows by using RETUR NING OLD clause.

```
gSQL> UPDATE lineitem
         SET l_discount = l_discount + 0.01
       WHERE l_returnflag = 'R'
   RETURNING OLD l_orderkey, l_linenumber, l_discount;
L_ORDERKEY L_LINENUMBER L_DISCOUNT
---------- ------------ ----------
         8            1        .06
         9            2         .1
        12            5        .04
        15            1        .02
        16            2        .07
5 rows updated.
```

Compatibility

The SQL standard does not define <update returning query statement>.

20.29 UPDATE name RETURNING .. INTO

Function

It updates a single row of a table, and the updated value is obtained into the host variable.

Syntax

```
<update statement: searched> ::=
    UPDATE table_name [ [ AS ] alias_name ]
        SET <set clause> [, ...]
        [ WHERE <search condition> ]
        [ <result offset clause> ]
        [ <fetch limit clause> ]
        <returning into clause>
    ;
<set clause> ::=
      column_name = { <value expression> | DEFAULT }
    | ( column_name [, ...] ) = ( { <value expression> | DEFAULT } [, ...] )
    | ( column_name [, ...] ) = ( <query expression> )
<result offset clause> ::=
    OFFSET skip_count [ ROW | ROWS ]
<fetch limit clause> ::=
      <fetch first clause>
    | <limit clause>
<fetch first clause> ::=
    FETCH [ FIRST | NEXT ] [ row_count ] [ ROW ONLY | ROWS ONLY ]
<limit clause>
    LIMIT { fetch_row_count | offset_row_count, fetch_row_count | ALL }
<returning into clause> ::=
    { RETURN | RETURNING } [ NEW | OLD ] { * | { <value expression> [ [AS] alias_name] } [,
...] } INTO variable_name [, ...]
```

Invocation and Access Rules

The user should satisfy the following conditions to perform <update returning query statement>.

- One of the following privileges is required to perform UPDATE statement.
  - UPDATE(columns) ON TABLE for all columns which are targets to be updated
  - (UPDATE or CONTROL TABLE) ON TABLE for the table
  - (UPDATE TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - UPDATE ANY TABLE ON DATABASE

- One of the following privileges for all columns used in RETURNING clause is required.
  - SELECT(columns) ON TABLE for all columns used in RETURNING clause
  - (SELECT or CONTROL TABLE) ON TABLE for the table
  - (SELECT TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - SELECT ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the name of a target table whose rows are to be updated.

[AS alias_name]

It is the alias of table_name.

<set clause>

It defines the columns to be updated and its values to be assigned. The number of columns in <set clause > should be as same as the number of values.
For more information, refer to **UPDATE**.

WHERE <search condition>

It updates the rows which satisfy WHERE condition.
If WHERE condition is not specified, all rows are updated.
For more information about WHERE condition, refer to **where clause** of **SELECT**.

〈result offset clause〉

It specifies the number of rows to skip in the query result.
For more information, refer to **〈result offset clause〉** of **SELECT**.

〈fetch limit clause〉

It specifies the number of rows to fetch in two ways, which are 〈fetch first clause〉 and 〈limit clause〉.

- 〈fetch first clause〉
  - It specifies the number of rows to be fetched.
  - For more information, refer to **〈fetch first clause〉** of **SELECT** statement.
- 〈limit clause〉
  - It specifies the number of rows to be fetched, or it simultaneously specifies both the number of r ows to be skipped and the number of rows to be fetched.
  - For more information, refer to **〈limit clause〉** of **SELECT** statement.

RETURNING .. AS ..

It defines the updated rows as a result set, and specifies columns to be retrieved from the result set.
For more information, refer to **〈returning clause〉** of **UPDATE name RETURNING**.

INTO variable_name [, ...]

The number of variables specified in INTO clause should be equal to the number of the expressions specifi ed in RETURNING clause.
The row to be updated should be one or less. If two or more rows are updated, an error occurs.

Description

For more information, refer to **Differences among UPDATE-related Statements**.

Example

The following is an example of obtaining column values of the updated rows into the host variables.

- Declare the host variable.

```
gSQL> \VAR v_discount NUMBER
gSQL> UPDATE lineitem
         SET l_discount = l_discount + 0.01
       WHERE l_orderkey = 12 AND l_linenumber = 5
   RETURNING l_discount INTO :v_discount;
V_DISCOUNT
----------
       .05
1 row updated.
```

Compatibility

In the SQL standard, <update returning into statement> statement does not exist.

20.30 UPDATE name WHERE CURRENT OF cursor_name

Function

It updates a single row which the current cursor indicates.

Syntax

```
<update statement: positioned> ::=
    UPDATE table_name [ [ AS ] alias_name ]
        SET <set clause> [, ...]
        WHERE CURRENT OF cursor_name
    ;
```

Invocation and Access Rules

The user should satisfy the following conditions to perform <update statement: positioned>.

- One of the following privileges is required to perform UPDATE statement.
  - UPDATE(columns) ON TABLE for all columns which are targets to be updated
  - (UPDATE or CONTROL TABLE) ON TABLE for the table
  - (UPDATE TABLE or CONTROL SCHEMA) ON SCHEMA for the schema to which the table belongs
  - UPDATE ANY TABLE ON DATABASE

Syntax Rules and Parameters

table_name

It is the name of a table whose rows are to be updated.

[AS alias_name]

It is the alias of table_name.

〈set clause〉

It defines the columns to be updated and its values to be assigned. The number of columns in 〈set clause 〉 should be as same as the number of values.
For more information, refer to **UPDATE**.

cursor_name

The cursor corresponding to cursor_name should satisfy the following conditions.

- The cursor should be OPEN. (Refer to **OPEN cursor_name**.)
- Fetched rows by using the cursor should exist. (Refer to **FETCH cursor_name**.)
- The query used for the cursor should identify table_name. (Refer to **DECLARE cursor_name**.)
- The cursor should be updatable for table_name. (Refer to **DECLARE cursor_name**.)

Description

For more information, refer to **Differences among UPDATE-related Statements**.

Examples

The following is an example that 〈update statement: positioned〉 is performed in interactive SQL (gsql) us ing the cursor.

- Declare the host variable.

```
gSQL> \VAR v_discount NUMBER
```

- Declare the cursor.

```
gSQL> DECLARE update_cursor CURSOR FOR
        SELECT l_discount
          FROM lineitem
         WHERE l_orderkey = 8 AND l_linenumber = 1
```

```
           FOR UPDATE;
Cursor declared.
```

- Open the cursor.

```
gSQL> OPEN update_cursor;
Cursor is open.
```

- Fetch the row.

```
gSQL> FETCH update_cursor INTO :v_discount;
V_DISCOUNT
----------
       .06
1 row fetched.
```

- Update the current row.

```
gSQL> UPDATE lineitem
         SET l_discount = l_discount + 0.01
       WHERE CURRENT OF update_cursor;
1 row updated.
```

- Close the cursor.

```
gSQL> CLOSE update_cursor;
Cursor closed.
gSQL> COMMIT;
Commit complete.
```

The following is an example of performing <update statement: positioned> by using the cursor in embed ded SQL program.

```
{
    ...
    EXEC SQL BEGIN DECLARE SECTION;
        ...
        double v_discount;
        ...
    EXEC SQL END DECLARE SECTION;
    ...
    EXEC SQL DECLARE update_cursor CURSOR FOR
```

```
            SELECT l_discount
              FROM lineitem
             WHERE l_orderkey = 8 AND l_linenumber = 1
               FOR UPDATE;
    ...
    EXEC SQL OPEN update_cursor;
    ...
    EXEC SQL FETCH NEXT update_cursor INTO :v_discount;
    ...
    EXEC SQL UPDATE lineitem
                SET l_discount = l_discount + 0.01
              WHERE CURRENT OF update_cursor;
    ...
    EXEC SQL CLOSE update_cursor;
    ...
    EXEC SQL COMMIT WORK;
    ...
}
```

Compatibility

Table 20-29 SQL standard compatibility

Feature ID	Description	Compatibility
F831	Full cursor update	O
B031	Basic dynamic SQL	O

For More Information

Refer to **CLOSE cursor_name**.

Part IV.

PSM Manual

21.

Overview of PSM

21.1 Features of PSM

Closely Interworking with SQL

GOLIDILOCKS PSM can closely interworks with GOLDILOCKS SQL.

- It supports all data types supported by GOLDILOCKS SQL.
- It supports an attribute type (%TYPE, %ROWTYPE), so a flexible tables and column types are availabl e.
- It supports all operators and built-in functions supported by GOLDILOCKS SQL.
- It supports all DML, DCL (COMMIT/ROLLBACK) supported by GOLDILOCKS SQL.
- It supports SQL SELECT statements by declaring a cursor and using OPEN, FETCH, CLOSE statements.
- It supports SQL DDL statements through the dynamic SQL feature.

Improving Performance

GOLDILOCKS PSM is operated only within a server, so it reduces the number of communications between the user application and the DBMS server. Therefore, the overall performance is improved.

Improving Productivity

The language of GOLDILOCKS PSM is quite similar to that of script, a less effort is required to write a cod e for the desired feature. The time of writing a client application can be reduced if the procedure and fun cure is created by modularizing the user's work logic.

Portability

Procedures and functions created by GOLDILOCKS PSM can also be used in ODBC, JDBC, an embedded S QL, and various development tools in the same way. Also, it can be easily transplanted in a different platf orm regardless of the platform type of server or a client.

Easy Maintenance

GOLDILOCKS PSM is implemented into a single module in a server from each similar logics in separate cli ents. Therefore, it is easy to manage, and enable to modify when it is in need even during the module is i n use.

21.2 Language Elements

Data Types

GOLDILOCKS PSM provides all built-in data types provided by GOLDILOCKS SQL, and it even supports use r defined records and collection types.
For more information, refer to **PSM DataTypes**.

Variables

Required variables declared by a user in GOLDILOCKS PSM can be used in all expressions.

For more information, refer to the followings.

- **Declarative Part**
- **Assignment**

Control Structures

GOLDILOCKS PSM supports most of decision branch statements, unconditional branch statements (GOT O), loop statement for repeated executions which were provided by a general script language.
For more information, refer to **PSM Control Statements**.

Subprograms

GOLDILOCKS PSM subprogram is a PSM block with a name and it can be repeatedly performed. If a subp rogramd has an argument, then it can be performed by being given different arguments when each time it is called.
Subprogram is either in a procedure form or a function form, and the function form has a return value. It also supports a nested subprogram which is declared in a specific block and used within it.
For more information, refer to **Using PSM Subprograms**.

21.3 Processing Transaction in PSM

A database transaction is a work unit which consists of one or more of SQL statements and it can not be disassembled. The followings are SQL statements which use transactions in GOLDILOCKS database.

- DML except for SELECT statements
- All DDL

A transaction starts in the following cases.

- When performing SQL which uses a transaction for the first time immediately after the connection.
- When performing SQL which uses a transaction for the first time since COMMIT or ROLLBACK

A transaction ends when a user performs COMMIT or ROLLBACK, or terminates the connection.

A subprogram module created by GOLDILOCKS PSM is a statement which does not uses its own transacti on, so a new transaction does not occur when the subprogram is called.

However, to guarantee atomicity of the SQL statement, the the available SQL types in a subprogram may vary depending on the case of calling the subprogram.

- When a user directly calls a subprogram module by using CALL statement, or performs an anonymou s block
  - A superordinate statement does not exists, so all kinds of SQL can be used within a subprogram and COMMIT/ROLLBACK is also allowed.
- When a subprogram module is used within a DML statement (INSERT/UPDATE/DELETE) except for S ELECT
  - A superordinate statement has a transaction, so all kinds of SQL can be used within a subprogra m. However, to guarantee the atomicity of the superordinate statement, neither COMMIT nor R OLLBACK is allowed.
- When a subprogram module is called within a SELECT statement
  - A superordinate statement does not use a transaction, so any kind of SQL using a transaction can not be used even within a called subprogram, nor is COMMIT/ROLLBACK allowed. Only SELECT s tatement is allowed.

22.

PSM DataTypes

22.1 Built-in Data Types

GOLDILOCKS PSM supports all basic data types provided by GOLDILOCKS SQL in the same way.
The followings are the basic data types.
For more information, refer to **Data Type**.

Numeric Types

- NUMBER
- NUMERIC
- FLOAT
- NATIVE_INTEGER
- NATIVE_DOUBLE

CHARACTER STRING Types

- CHARACTER (CHAR)
- CHARACTER VARYING (VARCHAR, VARCHAR2)
- CHARACTER LONG VARYING (LONG VARCHAR)

Exceptionally, for the compatibility with other database, GOLDILOCKS PSM provides VARCHAR types (VARCHAR2, CHAR VARYING, CHARACTER VARYING) whose precision is not specified, a nd which is not supported by GOLDILOCKS SQL.

This type can not be used when declaring an ordinary variable, but it can be used only when de claring an argument of a subprogram, a return type or an argument of a cursor. If this type is sp ecified, then the argument and the return type is determined as a type with the biggest value a mong VARCHAR types. (precision = 4000)

BINARY STRING Type

- BINARY
- BINARY VARYING (VARBINARY)
- BINARY LONG VARYING (LONG VARBINARY)

DATE/TIME Type

- DATE
- TIME [WITH/WITHOUT TIME ZONE]
- TIMESTAMP [WITH/WITHOUT TIME ZONE]

INTERVAL Type

- INTERVAL YEAR
- INTERVAL MONTH
- INTERVAL YEAR TO MONTH
- INTERVAL DAY
- INTERVAL HOUR
- INTERVAL MINUTE
- INTERVAL SECOND
- INTERVAL DAY TO HOUR
- INTERVAL DAY TO MINUTE
- INTERVAL DAY TO SECOND
- INTERVAL HOUR TO MINUTE
- INTERVAL HOUR TO SECOND
- INTERVAL MINUTE TO SECOND

BOOLEAN Type

BOOLEAN

ROWID Type

ROWID

Declaring Built-in Data Type Variables

A variable can be declared in each declaration section within an anonymous block, a procedure or a funct ion.

```
DECLARE
  V_MSG VARCHAR(20) := 'HELLO, WORLD!';
BEGIN
  DBMS_OUTPUT.PUT_LINE( 'My First Message Is : ' || V_MSG );
END;
/
```

For more information, refer to **Built-in Data Type References**.

22.2 Attribute Data Types

It is a data type which is used when specifying types of other PSM variables, cursors, tables or a specific c olumn in a table.

If an attribute variable or a target object (table) of a function is altered, then PSM (procedure, function) is automatically compiled again according to the altered type and it is applied.

%TYPE

It is used when specifying a type of other variables or a column type of a specific table. The following obj ects can be referenced.

- Scalar type (Including built-in data type) variables
- User defined record type variables
- A specific field of a record type variable
- Collection type variables
- A specific field of a collection type variable
- A specific column of a table

%TYPE is used as follows.

```
CREATE TABLE EMP ( ID INTEGER, NAME VARCHAR(32) );
INSERT INTO EMP VALUES ( 1001, 'Tom Jackson' );
COMMIT;
DECLARE
  V_NAME EMP.NAME%TYPE;
BEGIN
  SELECT NAME INTO V_NAME FROM EMP;
  DBMS_OUTPUT.PUT_LINE( 'EMP.NAME = ' || V_NAME );
END;
/
```

%ROWTYPE

It is used to specify the structure of a specific table, or the record type as same as the returned type of a s pecific cursor. It can refer to the following objects.

- Table
- Cursor
- Cursor variable

Neither the record type variable nor collection type variable can be a target of %ROWTYPE.

The following is an example of using %ROWTYPE.

```
DECLARE
  V_EMP EMP%ROWTYPE;
BEGIN
  SELECT * INTO V_EMP FROM EMP WHERE ID = 1001;
  DBMS_OUTPUT.PUT_LINE( 'Name of ID 1001 Is : ' || V_EMP.NAME );
END;
/
```

Constraint Attributes Inheritance

The constraint attributes of variables declared as an attribute type inherits the constraint attributes of the reference target as follows.

Table 22-1 Whether to inherit constraints of the attribute types

Attribute type	Reference target	NOT NULL	Default value
%TYPE	Scalar variable	O	X
	Record type variable	O	O
	A specific field of a record type variable	O	X
	Collection type variable - scalar element	O	X
	Collection type variable - record element	O	O
	A specific field of a collection type variable	O	X
	A specific column of a table	X	X
%ROWTYPE	Table	X	X
	Cursor	X	X

22.3 User-defined Record Type

The record type variable is a complex structure type which consists of several fields of each different type. A record type variable can be created by copying types of other tables or cursors by using %ROWTYPE, o r by declaring a data structure which is appropriate to a specific purpose by a user.

A user defined record type can be defined by using TYPE keyword in PSM declarative part as follows. Eac h field can optionally specify a NOT NULL constraint and a default value.

```
DECLARE
  TYPE MY_EMP_TYPE IS RECORD
  (
    ID INTEGER := 99999,
    NAME VARCHAR(32) NOT NULL DEFAULT 'anonymous'
  );
  V_EMP MY_EMP_TYPE;
BEGIN
  SELECT ID, NAME INTO V_EMP.ID, V_EMP.NAME FROM EMP;
  DBMS_OUTPUT.PUT_LINE('ID = ' || V_EMP.ID);
  DBMS_OUTPUT.PUT_LINE('NAME = ' || V_EMP.NAME);
END;
/
```

The following is an example of using it in a nested procedure or in a nested function.

```
DECLARE
  TYPE MY_EMP_TYPE IS RECORD
  (
    ID INTEGER := 99999,
    NAME VARCHAR(32) NOT NULL DEFAULT 'anonymous'
  );
  V_EMP MY_EMP_TYPE;
  PROCEDURE SET_EMP( A_EMP IN OUT MY_EMP_TYPE )
  IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE('ID = ' || A_EMP.ID);
    DBMS_OUTPUT.PUT_LINE('NAME = ' || A_EMP.NAME);
    SELECT ID, NAME INTO A_EMP.ID, A_EMP.NAME FROM EMP;
  END;
BEGIN
```

```
  SET_EMP( V_EMP );
  DBMS_OUTPUT.PUT_LINE('ID = ' || V_EMP.ID);
  DBMS_OUTPUT.PUT_LINE('NAME = ' || V_EMP.NAME);
END;
/
```

A user defined record type can be used as an argument or a returned type of a general regional variable, a nested procedure, or a nested function. However, it can not be used as an argument or a returned type of a schema-level procedure or a schema-level function.

22.4 User-defined Collection Type

User defined collection type is a kind of an array structure which stores one or more data. GOLDILOCKS PSM supports an associative array type which can be stored as key/value pair among collection types.

The following is a basic SYNTAX to declare an associative type.

```
TYPE <type_name> IS TABLE OF <element_data_type> INDEX BY <index_key_data_type>
```

- <type_name> is a user defined name for the type.
- <element_data_type> specifies a data type of a value which configures an array.
- <index_key_data_type> specifies a data type of an index key which explores for an element.

For example, if the information corresponding to (number) is in form of (name, age), then a table can be created and stored in a database as follows.

```
CREATE TABLE INFO
(
   NO INTEGER,
   NAME VARCHAR(20),
   AGE INTEGER
)
CREATE UNIQUE INDEX IDX_NO ON INFO (NO)
```

- If the information above is stored as an associative array, then its configuration is as follows.
  - Index_key corresponds to a number (NO).
  - Element_data consists of a name (NAME) and an age (AGE).

An associative array variable to store it is defined within a real PSM as follows.

```
TYPE rec IS RECORD (NAME VARCHAR(20), AGE INTEGER);
TYPE info IS TABLE OF rec INDEX BY INTEGER;
```

For more information, refer to **COLLECTION Variable Declaration**.

Associative Array

An associative array type variable has a data type key specified in an index clause, and it is a PSM variable which can store a value of element data type specified in TABLE OF clause in one or more values in a key/value form.

- The features of an associative array type of GOLDILOCKS PSM is as follows.
  - It has an exploring key in a data type form specified in INDEX BY clause, and it is automatically so rted and stored.
  - It consists of elements specified in TABLE OF clause.
  - It provides exploring functions called as collection methods.
  - It is stored in a replace form when storing an element in an existing index key.
  - The maximum storage size is restricted to the available size of MEMORY_TEMP_TBS_SIZE.

The following is an example of declaring an element type as an SQL data type, and inserting data.

```
gSQL> CREATE TABLE T1
(
  C1 VARCHAR(20),
  C2 VARCHAR(20)
);
Table created.
gSQL> DECLARE
  TYPE rec IS TABLE OF VARCHAR(20) INDEX BY VARCHAR(10);
  V1 rec;
BEGIN
  V1('aa') := 'Dog';
  V1('bb') := 'Cat';
  INSERT INTO T1 VALUES ( V1('aa'), V1('bb') );
END;
/
Anonymous PL block executed.
gSQL> SELECT * FROM T1;
C1  C2
--- ---
Dog Cat
1 row selected.
```

The following is an example of having a record type as an element.

```
gSQL> CREATE TABLE T1
(
  C1 VARCHAR(20),
  C2 VARCHAR(20)
);
Table created.
gSQL> DECLARE
  TYPE rec IS TABLE OF T1%ROWTYPE INDEX BY VARCHAR(10);
```

```
  V1 rec;
BEGIN
  V1('person1').C1 := 'seoul';
  V1('person1').C2 := '12';
  V1('person2').C1 := 'busan';
  V1('person2').C2 := '24';
  INSERT INTO T1 VALUES V1('person1'), V1('person2');
END;
/
Anonymous PL block executed.
gSQL> SELECT * FROM T1;
C1    C2
----- --
seoul 12
busan 24
2 rows selected.
```

- The following is a data type which can be specified in an index key of an associative array of GOLDIL OCKS PSM.
  - INTEGER
  - LONG
  - CHAR
  - VARCHAR

- The following is a data type which can be used as an element type of an associative array of GOLDILO CKS PSM.
  - SQL Data Type
  - %TYPE
  - %ROWTYPE
  - User defined Record Type

For more information, refer to **Built-in Data Types**, **%TYPE**, **%ROWTYPE**, **User Defined Record Type**.

Assign Values to Collection Variables

The following rules are applied when assigning values to collection valuables.

- The rules as same as <Assign Statement> is applied to an element assignment.
- To assign values to the entire collection variable, the types of collection variables should be same.
- When assigning an element of a collection variable, it is operated according to the element data type as follows.

- Assigning a user defined type element is allowed among same type.
- Else, it is allowed to assign when data types of a field configuring an element are compatible at a run-time.

The following is an example of an error due to the assignment of a different type.

```
DECLARE
  TYPE udr1 IS RECORD (F1 INTEGER, F2 VARCHAR(20));
  TYPE udr2 IS RECORD (F1 INTEGER, F2 VARCHAR(20));
  TYPE rec1 IS TABLE OF udr1 INDEX BY VARCHAR(10);
  TYPE rec2 IS TABLE OF udr2 INDEX BY VARCHAR(10);
  V1 rec1;
  V2 rec2;
BEGIN
  V2('person1').F1 := 24;
  V2('person1').F2 := 'seoul korea';
  V1 := V2;
END;
/
ERR-HY000(17032): PSM compilation error :
(1) at (14:9): ERR-HY000(17007): invalid expression
```

In the example above, configuration of each field of an element which consists of user defined type is same. However, the variable types are different, so the assignment fails.

The following is an example of succeeding in an assignment by using the same type.

```
DECLARE
  TYPE udr1 IS RECORD (F1 INTEGER, F2 VARCHAR(20));
  TYPE rec1 IS TABLE OF udr1 INDEX BY VARCHAR(10);
  V1 rec1;
  V2 rec1;
BEGIN
  V2('person1').F1 := 24;
  V2('person1').F2 := 'seoul korea';
  V1 := V2;
END;
/
Anonymous PL block executed.
```

Assignment of an associative type in an element unit is allowed only when the data types of elements are compatible.

Refer to the following example.

```
gSQL> CREATE TABLE T1
(
  c1 VARCHAR(20),
  c2 VARCHAR(20)
);
Table created.
gSQL> DECLARE
TYPE record_org1 IS RECORD (f1 VARCHAR(20), f2 VARCHAR(20));
TYPE rec1 IS TABLE OF t1%rowtype INDEX BY VARCHAR(10);
TYPE rec2 IS TABLE OF record_org1 INDEX BY VARCHAR(10);
v1 record_org1;
v2 rec1;
v3 rec2;
v4 t1%rowtype;
BEGIN

  -- From record_org1 type to %rowtype
  V2('first') := v1;
  -- From record_org1 type to record_org1 type
  V3('first') := v1;

  -- From t1%rowtype to record_org1 type
  V3('second') := V2('first');
END;
/
Anonymous PL block executed.
```

An associative type variable is stored in a volatile memory space, and the user accessible TEMP TABLESPA CE should be extended when the space is insufficient. The following is an example of an error due to the memory insufficiency.

```
DECLARE
TYPE rec IS TABLE OF t1%rowtype INDEX BY varchar(20);
v1 rec;
BEGIN
  BEGIN
    FOR i IN 1 .. 100000
    LOOP
      v1(i).c1 := i;
```

```
      v1(i).c2 := i;
    END LOOP;
    EXCEPTION WHEN OTHERS THEN
                dbms_output.put_line('error: count=' || v1.count());
                dbms_output.put_line('sqlcode=' || SQLCODE);
                dbms_output.put_line('sqlmsg =' || SQLERRM);
  END;
  dbms_output.put_line('v1.count=' || v1.count());
END;
/
error: count=95004
sqlcode=-14015
sqlmsg =[SUNJESOFT][PSM][GOLDILOCKS]there is no extendible datafile in tablespace
'MEM_TEMP_TBS'
v1.count=95004
Anonymous PL block executed.
```

Collection Method

A collection method is a function or a procedure which is provided to easily operate a collection type variable. Associative array provides the following methods.

Table 22-2 Collection method

Method	Type	Input argument	Return type	Description
FIRST	Function	X	Index key data type	It returns the first index key.
LAST	Function	X	Index key data type	It returns the last index key.
COUNT	Function	X	INTEGER	It returns the number of elements.
EXISTS	Function	O	BOOLEAN	It returns whether an index key exists or not.
PRIOR	Function	O	Index key data type	It returns an index key prior to the input index key.
NEXT	Function	O	Index key data type	It returns an index key next to the input index key.
DELETE	Procedure	O	N/A	It deletes an element corresponding to the input index key.

A collection method is used as follows.

```
gSQL> CREATE TABLE T1
(
  C1 VARCHAR(20),
  C2 VARCHAR(20)
);
Table created.
gSQL> DECLARE
  TYPE rec IS TABLE OF T1%ROWTYPE INDEX BY VARCHAR(10);
  V1 rec;
BEGIN
  V1('person1').C1 := 'seoul';
  V1('person1').C2 := '12';
  V1('person2').C1 := 'busan';
  V1('person2').C2 := '24';
  V1('person3').C1 := 'Daegu';
  V1('person3').C2 := '36';
  -- First method
  DBMS_OUTPUT.PUT_LINE('First Index Key = ' || V1.first() );
  -- Last method
  DBMS_OUTPUT.PUT_LINE('Last Index Key = ' || V1.last() );
  -- Count method
  DBMS_OUTPUT.PUT_LINE('Count of element = ' || V1.count() );
  -- Prior Method
  DBMS_OUTPUT.PUT_LINE('Prior (person1) = ' || V1.prior('person1') ); -- return NULL
  DBMS_OUTPUT.PUT_LINE('Prior (person3) = ' || V1.prior('person3') );
  -- Next Method
  DBMS_OUTPUT.PUT_LINE('Next (person1) = ' || V1.next('person1') );
  DBMS_OUTPUT.PUT_LINE('Next (person3) = ' || V1.next('person3') ); -- return NULL
  -- Exists Method
  DBMS_OUTPUT.PUT_LINE('Exists (person2) = ' || V1.exists('person2') );
  -- Delete Method
  V1.delete('person2');
  -- Exists Method
  DBMS_OUTPUT.PUT_LINE('After delete, Exists (person2) = ' || V1.exists('person2') );
END;
/
First Index Key = person1
Last Index Key = person3
Count of element = 3
Prior (person1) =
Prior (person3) = person2
```

```
Next (person1) = person2
Next (person3) =
Exists (person2) = TRUE
After delete, Exists (person2) = FALSE
Anonymous PL block executed.
```

If a delete procedure deleting an element can not find an index key corresponding to the input argument, then an error occurs as follows.

```
gSQL> DECLARE
  TYPE rec IS TABLE OF T1%ROWTYPE INDEX BY VARCHAR(10);
  V1 rec;
BEGIN
  V1('person1').C1 := 'seoul';
  V1('person1').C2 := '12';
  -- Call delete procedure
  V1.delete('person2');
END;
/
ERR-HY000(17045): no data found :
  V1.delete('person2');
  *
ERROR at line 9:
Anonymous PL block executed.
```

22.5 SYS_REFCURSOR

SYS_REFCURSOR is a predefined type for a cursor variable, and it is used to declare a cursor variable.

It is used to declare a cursor variable in the following form.

```
cursor_variable_name SYS_REFCURSOR;
```

Cursor variable can be used together with OPEN FOR, FETCH, CLOSE statement as follows.

```
DECLARE
  v1 VARCHAR(20);
  v2 VARCHAR(20);
  cv1 SYS_REFCURSOR;
  cv2 SYS_REFCURSOR;
BEGIN
  OPEN cv1 FOR SELECT * FROM T1;
  cv2 := cv1;
  FETCH cv2 INTO V1, V2;
  DBMS_OUTPUT.PUT_LINE('V1 = ' || V1 || ' , V2 = ' || V2);

END;
/
V1 = Seoul , V2 = 24
Anonymous PL block executed.
```

- The variable declared by using SYS_REFCURSOR is used as follows.
  - It is possible to assign each cursor variable among cursor variables. An existing open cursor of the cursor variable which is on the right of the assign can not be used any more.
  - An explicit cursor can not be assigned to a cursor variable.
  - A variable of a different data type can not be assigned to a cursor variable.
  - A nested function/procedure and schema-level function/procedure can be used as a parameter.

For more information, refer to **Cursor Variable**, **Cursor Variable Declaration**, **OPEN FOR Statement**.

23.

PSM Control Statements

23.1 Assignment

It inserts the calculation value of the right expression into the left variable by using assignment operators (':=').

Assignment Target

An assignment target is on the left side of the assignment operator, and the following items may exist on it.

- Variables (Including an argument of a procedure or a function except for IN type)
- Bind parameter such as '?' or ':V1' (It can be used only in an anonymous PL block.)

Assigning Expression

An expression available on PSM is on the right side of the assignment operator. Expressions available on a PSM expression is as follows.

- Constant
- All operators, built-in functions or pseudo columns provided by GOLDILOCKS SQL
- Bind parameter bound to IN or IN OUT type (It can be used only in an anonymous PL block.)
- PSM variable
- PSM nested function
- Schema-level function

The following is an example of assignment statement per each type.

```
CREATE OR REPLACE FUNCTION ADD_TEN( A1 INTEGER )
RETURN INTEGER
IS
BEGIN
  RETURN A1 + 10;
END;
/
COMMIT;
DECLARE
  FUNCTION ADD_ONE( A1 INTEGER )
```

```
  RETURN INTEGER
  IS
  BEGIN
    RETURN A1 + 1;
  END;
  V_NUM INTEGER;
  V_STR VARCHAR(10);
BEGIN
  V_NUM := 10;                  ❶ Numeric literal
  V_STR := 'ABC';               ❷ String literal
  :V_PARAM := V_STR || 'DEF';   ❸ PSM variable, SQL operator
  V_NUM := ADD_ONE( 100 ) + ADD_TEN( 1000 );
                                ❹ Nested function, Schema-level function
END;
/
```

The following expressions can not be used in an PSM expression.

- Column of a table, view or the corresponding object
- SQL objects such as an index, a sequence or a synonym
- Subquery
- Nested procedure
- Schema-level procedure

The following is an example of an error due to a wrong expression.

```
gSQL> CREATE SEQUENCE SEQ1;
Sequence created.
gSQL> COMMIT;
Commit complete.
gSQL> DECLARE
  V1 INTEGER;
BEGIN
  V1 := SEQ1.NEXTVAL;
END;
/
ERR-HY000(17032): PSM compilation error :
(1) at (4:9): ERR-42000(16074): sequence number not allowed here
```

To assign an object value of a type which is not available in an PSM expression, use SELECT INTO stateme nt as follows.

```
gSQL> DECLARE
  V1 INTEGER;
BEGIN
  SELECT SEQ1.NEXTVAL INTO V1 FROM DUAL;
END;
/
Anonymous PL block executed.
```

Assignment Compatibility

Even though the statement can use an expression of an assignment statement, it may succeed or fail dep ending on a target type and the final result type of an expression. The available expression types accordin g to a target type is as follows.

Table 23-1 Assignment compatibility

Target type	Final result type of an expression
Scalar type • Scalar type variable • A specific field of a record type variable • Scalar element of a collection variable with a key value	It is allowed only when it is a scalar type with a value conve rsible to a target type.
Attribute record type (%ROWTYPE)	It is allowed only when the number of fields are same and each field value is conversible to the field of the correspon ding order of a target.
User defined record type	Only the exactly same type is allowed.
Collection type • Collection type variable whose key value is not sp ecified	Only the exactly same type is allowed.

Even though the internal structures of user defined record type variables are same, an error occurs if the t ype names are different.

```
gSQL> DECLARE
  TYPE MY_REC1 IS RECORD ( F1 INTEGER := 1, F2 VARCHAR(10) := 'AAA' );
  TYPE MY_REC2 IS RECORD ( F1 INTEGER := 1, F2 VARCHAR(10) := 'AAA' );
  V1 MY_REC1;
  V2 MY_REC2;
BEGIN
  V2 := V1;
END;
```

```
/
ERR-HY000(17032): PSM compilation error :
(1) at (7:9): ERR-HY000(17007): invalid expression
```

Moreover, if a target is an argument variable defined with IN attribute or is a bind parameter bound with IN attribute, an error occurs.

```
gSQL> DECLARE
  FUNCTION ADD_ONE( A1 IN INTEGER )
  RETURN INTEGER
  IS
  BEGIN
    A1 := A1 + 1;
    RETURN A1;
  END;
  V1 INTEGER;
BEGIN
  V1 := ADD_ONE( 10 );
END;
/
ERR-HY000(17032): PSM compilation error :
(1) at (6:5): ERR-HY000(17024): (A1) cannot be used as assignment target
```

Assignment compatibility rules are applied same even when assigning the value of an actual parameter to a formal parameter variable while calling a function or a procedure.

23.2 PL Block

A PL block is a basic unit which configures PSM. A PL block can be nested again in another PL block. Item s declared within a PL block can be referenced only within the range of its own block (Including subordin ate blocks)

PL Block Configuration

A single PL block is divided into the following three parts.
For more information, refer to **Block (BEGIN .. END)**.

```
[DECLARE
❶ Declarative part]
BEGIN
❷ Statements
[EXCEPTION
❸ Handlers]
END;
```

Declarative Part

A declarative part is an area in which a local item to be used within a PL block is declared. If an item such as a local variable to be declared does not exist, then a declarative part is not defined, but it starts from a n executable part.

Local items to be declared in a declarative part is as follows.

- Variable (Refer to **PSM DataTypes**.)
- User defined type
- Cursor
- Nested function or a procedure (Subprogram)
- User defined exception

All names of items declared in a single PL block should be unique regardless of their types. For example, a cursor whose name is as same as a name of a variable can not be declared.

```
gSQL> DECLARE
  C1 INTEGER;
  CURSOR C1 IS SELECT * FROM DUAL;
```

```
BEGIN
  OPEN C1;
  FETCH C1 INTO C1;
  CLOSE C1;
END;
/
ERR-HY000(17032): PSM compilation error :
(1) at (3:10): ERR-HY000(17027): duplicated identifier
(2) at (5:8): ERR-HY000(17031): mismatch identifier type
(3) at (6:9): ERR-HY000(17031): mismatch identifier type
(4) at (7:9): ERR-HY000(17031): mismatch identifier type
```

However, an item in a subordinate PL block whose name is as same as an item in a superordinate PL bloc k can be declared. In this case, that name means the item found first when exploring from the current loc ation toward the superordinate block.

```
gSQL> <<PP>>
DECLARE     ❶ Parent scope begin
  V1 VARCHAR(10) := 'AAA';
BEGIN
  <<CC>>
  DECLARE   ❷ Child scope begin
    V1 INTEGER := 99;
  BEGIN
    DBMS_OUTPUT.PUT_LINE( 'PP.V1 = ' || PP.V1 );
    DBMS_OUTPUT.PUT_LINE( 'CC.V1 = ' || CC.V1 );
    DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
  END;      ❸ Child scope end
END;        ❹ Parent scope end
/
PP.V1 = AAA
CC.V1 = 99
V1 = 99
Anonymous PL block executed.
```

Executable Part

In an executable part, a declared items are manipulated and included statements are performed. Stateme nts available in an executable parts are as follows.

- Control Statements (Refer to **PSM Control Statements**.)

- Cursor Statements (Refer to **PSM Cursor Statements**.)
- SQL Statements (static/ dynamic) (Refer to **Using SQLs In PSM**.)

Exception Handling Part

It defines handlers which handles multiple exceptions occurring during the execution.

If an exception occurs during executing several statements in an executable part, then it explores from th e location of that PL block to the superordinate PL block to find out whether a handler is registered to ha ndle the exception. If there is the handler, then it performs the contents of that handle, then initializes th at exceptional situation.

```
DECLARE
  V1 INTEGER;
BEGIN
  SELECT C1 INTO V1 FROM T1 WHERE C1 = 100;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    DBMS_OUTPUT.PUT_LINE('SQL%ISOPEN   = ' || SQL%ISOPEN);
    DBMS_OUTPUT.PUT_LINE('SQL%FOUND    = ' || SQL%FOUND);
    DBMS_OUTPUT.PUT_LINE('SQL%NOTFOUND = ' || SQL%NOTFOUND);
    DBMS_OUTPUT.PUT_LINE('SQL%ROWCOUNT = ' || SQL%ROWCOUNT);
END;
/
```

If an appropriate handler is not found even after exploring to the top-level PL block, then the occurred ex ception is returned to a caller, and it is terminated.

```
gSQL> DECLARE
  v1 INTEGER;
BEGIN
  v1 := 1/ 0;
END;
/
ERR-HY000(17007): invalid expression :
  v1 := 1/ 0;
        *
ERROR at line 4:
ERR-22012(12122): divisor is equal to zero
```

For more information about declaration of user defined exception, built-in handler types or installing a ha ndler, refer to **Error Handling**.

23.3 NULL Statement

It is used to specify a statement which does not have any role in PSM such as no-op.
For more information, refer to **NULL Statement**.

```
DECLARE
  V1 INTEGER := 1;
BEGIN
  IF V1 < 10 THEN
    V1 := V1 + 1;
  ELSE
    NULL;  -- do nothing
  END IF;
END;
/
```

23.4 Testing Conditions

Conditional branch statements perform statements according to true/false of the given condition. GOLDI LOCKS PSM provides two conditional branch statements, which are IF and CASE.

IF

IF statement performs statements which are true among conditions of the given expression. Conditions o f an expression can be specified next to IF and ELSIF. If the condition of an expression is true when checki ng conditions in an order specified when performing, then it performs the statements below THEN clause. If all IF conditions and ELSIF conditions are false, and ELSE clause is specified, then it performs the statem ents below ELSE clause.

IF statement always ends with END IF keyword.
For more information, refer to **IF Statement**.

```
DECLARE
V1 INTEGER := 1;
BEGIN
  -- IF COND
  IF V1 > 0 THEN
    DBMS_OUTPUT.PUT_LINE( 'IF COND' );
  ELSIF V1 = 0 THEN
    DBMS_OUTPUT.PUT_LINE( 'ELSIF COND' );
  ELSE
    DBMS_OUTPUT.PUT_LINE( 'ELSE COND' );
  END IF;
  -- ELSIF COND
  IF V1 = 0 THEN
    DBMS_OUTPUT.PUT_LINE( 'IF COND' );
  ELSIF V1 > 0 THEN
    DBMS_OUTPUT.PUT_LINE( 'ELSIF COND' );
  ELSE
    DBMS_OUTPUT.PUT_LINE( 'ELSE COND' );
  END IF;
  -- ELSE COND
  IF V1 = 0 THEN
    DBMS_OUTPUT.PUT_LINE( 'IF COND' );
```

```
  ELSIF V1 < 0 THEN
    DBMS_OUTPUT.PUT_LINE( 'ELSIF COND' );
  ELSE
    DBMS_OUTPUT.PUT_LINE( 'ELSE COND' );
  END IF;
END;
/
```

ELSIF clause and ELSE clause are optional, so it may or may not be specified.

When ELSE statement is not specified and any conditions of IF or ELSIF is not true, then it does not perfor m any statement, but proceeds to the next statement without causing any exception.

CASE

CASE statement selects a single array among arrays of multiple statements such as IF statement, then per forms it. CASE statement ends with END CASE keyword, and ELSE clause is optional so it may or may not be specified.

If WHEN clause with an appropriate condition can not be found and ELSE clause is specified, then it arra ys statements below ELSE clause. If an appropriate condition does not exist nor is ELSE clause specified, t hen CASE_NOT_FOUND exception occurs.
For more information, refer to **CASE Statement**.

WHEN clauses in CASE is evaluated in an specified order, and if an appropriate WHEN clause is found an d statements of that WHEN clause are arrayed, then all WHEN clauses after that are ignored.

There are two kinds of CASE statements as follows.

- Simple CASE statement
- Searched CASE statement

Simple CASE Statement

A simple CASE statement arrays statements owned by WHEN clause which has the same value as a select or expression among expressions specified next to WHEN clause by using a selector expression specified next to CASE keyword. Then it ends.

```
DECLARE
V1 VARCHAR(1) := '2';
BEGIN
```

```
  CASE V1 WHEN '0' THEN DBMS_OUTPUT.PUT_LINE ('Result = 0');
          WHEN '1' THEN DBMS_OUTPUT.PUT_LINE ('Result = 1');
          WHEN '2' THEN DBMS_OUTPUT.PUT_LINE ('Result = 2');
          ELSE DBMS_OUTPUT.PUT_LINE ('Result = Other');
  END CASE;
END;
/
```

Searched CASE Statement

A searched CASE statement does not have a selector expression, instead a conditional expression which i s evaluated as boolean type per each WHEN clause is specified. When it is executed, it arrays statements owned by the first WHEN clause which is TRUE by evaluating expressions of each WHEN clause. Then it e nds.

```
DECLARE
V1 VARCHAR(1) := '2';
BEGIN
  CASE WHEN V1 = 0 THEN DBMS_OUTPUT.PUT_LINE ('Result = 0');
       WHEN V1 = 1 THEN DBMS_OUTPUT.PUT_LINE ('Result = 1');
       WHEN V1 = 2 THEN DBMS_OUTPUT.PUT_LINE ('Result = 2');
       ELSE DBMS_OUTPUT.PUT_LINE ('Result = OTHER');
  END CASE;
END;
/
```

23.5 Iterative Control

Iterative control statements performs a series of statements several times. GOLDILOCKS PSM provides the following iterative control statements.

- Basic loop
- FOR loop
- WHILE loop

Basic Loop

A basic loop statement encloses a series of statements which are to be performed several times with LOOP and END LOOP keywords. This statement indefinitely performs the inner statements, and it can escape out of the loop by using EXIT or GOTO statements which are sequential control statements.
For more information, refer to **Basic LOOP Statement**.

The following is an example of escaping from a basic loop which has the closest scope from that location by using EXIT WHEN statement.
For more information, refer to **EXIT Statement**.

```
gSQL> DECLARE
  V1 INTEGER := 1;
BEGIN
  LOOP
    DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
    V1 := V1 + 1;
    EXIT WHEN V1 > 10;   -- Escape Condition
  END LOOP;
END;
/
V1 = 1
V1 = 2
V1 = 3
V1 = 4
V1 = 5
V1 = 6
V1 = 7
V1 = 8
V1 = 9
```

```
V1 = 10
Anonymous PL block executed.
```

To escape out of a specific loop when multiple loops are nested, specify a label in front of that loop and s pecify that label as a target label in EXIT statement.

```
gSQL> DECLARE
  V1 INTEGER := 1;
BEGIN
  <<OUTER_LOOP>>
  LOOP
    <<INNER_LOOP>>
    LOOP
      DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
      V1 := V1 + 1;
      EXIT OUTER_LOOP WHEN V1 > 5;  -- Escape Condition
    END LOOP INNER_LOOP;
    DBMS_OUTPUT.PUT_LINE( 'END OF INNER LOOP' );
    V1 := V1 + 5;
  END LOOP OUTER_LOOP;
  DBMS_OUTPUT.PUT_LINE( 'END OF OUTER LOOP' );
END;
/
V1 = 1
V1 = 2
V1 = 3
V1 = 4
V1 = 5
END OF OUTER LOOP
Anonymous PL block executed.
```

FOR Loop

FOR loop statement repeatedly performs a series of statements as many times as the number of integers of the given scope.
For more information, refer to **FOR LOOP Statement**.

A user creates an index variable with the name specified next to FOR keyword.
Then, it sets the lower bound specified on the left side of a scope operator (..) which is next to IN keywor d as an initial value of the index variable. Then, it performs a series of statements by increasing the index

variable by 1 per each loop until it becomes as same as the higher bound specified on the right side of a s cope operator (..).

```
BEGIN
  FOR I IN 0 .. 2 LOOP
    DBMS_OUTPUT.PUT_LINE( 'I = ' || I );
  END LOOP;
END;
/
I = 0
I = 1
I = 2
Anonymous PL block executed.
```

When REVERSE keyword is specified in front of the scope, it sets the higher bound specified on the right side of a scope operator (..) as an initial value of the index variable. Then, it performs a series of statemen ts by decreasing the index variable by 1 per each loop until it becomes as same as the lower bound specifi ed on the left side of a scope operator (..).

```
BEGIN
  FOR I IN REVERSE 0 .. 2 LOOP
    DBMS_OUTPUT.PUT_LINE( 'I = ' || I );
  END LOOP;
END;
/
I = 2
I = 1
I = 0
Anonymous PL block executed.
```

WHILE Loop

WHILE loop statement repeatedly performs a series of statements while the given condition is TRUE.
For more information, refer to **WHILE LOOP Statement**.

```
DECLARE
V1 integer := 0;
BEGIN
  WHILE V1 < 10 LOOP
    DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
```

```
    V1 := V1 + 1;
  END LOOP;
END;
/
```

WHILE loop checks given conditions before performing statements of a body so if the condition is FALSE f rom the beginning, then it may not perform statements of a body at all.

23.6 Sequential Control

Sequential control statements move on which a program is performed from the current performing location to another location.

GOLDILOCKS PSM provides the following three sequential control statements.

- GOTO
- CONTINUE
- EXIT

GOTO

GOTO statement moves the performing location to the statement in which the given label is. It can be specified in front of any statement which is executable within PSM of a label. A target label to which GOTO moves can exist before and after the GOTO statement. However, only a visible statement can perform GOTO.
For more information, refer to **GOTO Statement**.

Conditions for a visible statement is as follows.

- Sibling statements located before and after the current performing location
- A superordinate statement of the current location, and sibling statements located before and after that superordinate statement

The following is an example of moving a performing location to a sibling statement of the current location.

```
gSQL> BEGIN
  <<PARENT_PREV>>
  DBMS_OUTPUT.PUT_LINE('PARENT PREV');
  <<PARENT>>
  IF 1 > 0 THEN
    <<SIBLING_PREV>>
    DBMS_OUTPUT.PUT_LINE('SIBLING PREV');
    <<CURRENT_POSITION>>
    GOTO SIBLING_NEXT;
    DBMS_OUTPUT.PUT_LINE('SIBLINGS');
    DBMS_OUTPUT.PUT_LINE('SIBLINGS');
    <<SIBLING_NEXT>>
```

```
    DBMS_OUTPUT.PUT_LINE('SIBLING NEXT');
  ELSE
    <<NON_SIBLING>>
    DBMS_OUTPUT.PUT_LINE('NON SIBLING');
  END IF;
  <<PARENT_NEXT>>
  DBMS_OUTPUT.PUT_LINE('PARENT NEXT');
END;
/
PARENT PREV
SIBLING PREV
SIBLING NEXT
PARENT NEXT
Anonymous PL block executed.
```

The following is an example of moving a performing location to a superordinate statement of a current lo cation.

```
gSQL> BEGIN
  <<PARENT_PREV>>
  DBMS_OUTPUT.PUT_LINE('PARENT PREV');
  <<PARENT>>
  IF 1 > 0 THEN
    <<SIBLING_PREV>>
    DBMS_OUTPUT.PUT_LINE('SIBLING PREV');
    <<CURRENT_POSITION>>
    GOTO PARENT_NEXT;
    DBMS_OUTPUT.PUT_LINE('SIBLINGS');
    DBMS_OUTPUT.PUT_LINE('SIBLINGS');
    <<SIBLING_NEXT>>
    DBMS_OUTPUT.PUT_LINE('SIBLING NEXT');
  ELSE
    <<NON_SIBLING>>
    DBMS_OUTPUT.PUT_LINE('NON SIBLING');
  END IF;
  <<PARENT_NEXT>>
  DBMS_OUTPUT.PUT_LINE('PARENT NEXT');
END;
/
PARENT PREV
SIBLING PREV
```

```
PARENT NEXT
Anonymous PL block executed.
```

The following is an example of moving a performing location to a statement which is not a sibling, so an error occurs.

```
gSQL> BEGIN
  <<PARENT_PREV>>
  DBMS_OUTPUT.PUT_LINE('PARENT PREV');
  <<PARENT>>
  IF 1 > 0 THEN
    <<SIBLING_PREV>>
    DBMS_OUTPUT.PUT_LINE('SIBLING PREV');
    <<CURRENT_POSITION>>
    GOTO NON_SIBLING;
    DBMS_OUTPUT.PUT_LINE('SIBLINGS');
    DBMS_OUTPUT.PUT_LINE('SIBLINGS');
    <<SIBLING_NEXT>>
    DBMS_OUTPUT.PUT_LINE('SIBLING NEXT');
  ELSE
    <<NON_SIBLING>>
    DBMS_OUTPUT.PUT_LINE('NON SIBLING');
  END IF;
  <<PARENT_NEXT>>
  DBMS_OUTPUT.PUT_LINE('PARENT NEXT');
END;
/
ERR-HY000(17032): PSM compilation error :
(1) at (12:10): ERR-HY000(17010): can not find label name
```

Likewise, when moving a performing location to a statement which is not a superordinate sibling of the c urrent statement, then an error occurs.

GOTO is a good tool to make a PSM program logic flexible. However, like in other languages, if it is used too much, then it may downgrade the readability, so maintenance becomes difficult. Therefore, it is reco mmended to use loop statements such as FOR or WHILE for an ordinary logics, and use GOTO only when it is necessary.

CONTINUE

CONTINUE statement terminates the current iteration and starts the next iteration within a loop statemen t such as FOR or WHILE.
For more information, refer to **CONTINUE Statement**.

```
DECLARE
  V1 INTEGER := 1;
BEGIN
  <<AAA>>
  WHILE V1 <= 10 LOOP
    DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
    V1 := V1 + 1;
    IF V1 <= 2 THEN
      DBMS_OUTPUT.PUT_LINE( 'CONTINUE' );
      CONTINUE;
    ELSE
      EXIT;
    END IF;
    DBMS_OUTPUT.PUT_LINE( 'END-OF-WHILE' );
  END LOOP AAA;
END;
/
```

If a target label is specified, it starts the next iteration of a loop which has the corresponding label. If it is not specified, it starts the next iteration of the closest (inner) loop.

```
DECLARE
  V1 INTEGER := 1;
BEGIN
  <<AAA>>
  FOR I IN 0..5 LOOP
    <<BBB>>
    WHILE V1 <= 10 LOOP
      DBMS_OUTPUT.PUT_LINE( 'I = ' || I );
      DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
      V1 := V1 + 1;
      IF V1 <= 2 THEN
        CONTINUE BBB;
      ELSE
```

```
        EXIT AAA;
      END IF;
    END LOOP BBB;
  END LOOP AAA;
END;
/
```

WHEN clause can be optionally specified in CONTINUE statement. In this case, it terminates the current it eration and starts the next iteration, only when an expression next to WHEN clause is TRUE.

```
DECLARE
  V1 INTEGER :=0 ;
BEGIN
  LOOP
    V1 := V1 + 1;
    CONTINUE WHEN V1 < 5;
    EXIT;
  END LOOP;
END;
/
```

EXIT

EXIT statement escapes the current loop and performs the next statement.
For more information, refer to **EXIT Statement**.

```
DECLARE
  V1 INTEGER := 1;
BEGIN
  WHILE V1 <= 10 LOOP
    DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
    IF V1 > 5 THEN
      EXIT;
    END IF;
    V1 := V1 + 1;
  END LOOP;
END;
/
```

If a target loop is specified, then it escapes a loop which has the corresponding label.

```
DECLARE
  V1 INTEGER := 1;
BEGIN
  <<AAA>>
  FOR I IN 0..5 LOOP
    <<BBB>>
    WHILE V1 <= 10 LOOP
      DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
      IF V1 >= 5 THEN
        EXIT AAA;
      END IF;
      V1 := V1 + 1;
    END LOOP BBB;
  END LOOP AAA;
END;
/
```

WHEN clause can be optionally specified in EXIT statement, like as in CONTINUE statement. In this case, it performs EXIT, only when the given condition is TRUE.

```
DECLARE
  V1 INTEGER := 1;
BEGIN
  <<AAA>>
  FOR I IN 0..5 LOOP
    <<BBB>>
    WHILE V1 <= 10 LOOP
      DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
      EXIT AAA WHEN V1 >= 5;
      V1 := V1 + 1;
    END LOOP BBB;
  END LOOP AAA;
END;
/
```

23.7 Error Handling

Errors in PSM are as follows depending on the point of time when it occurs.

- Errors at compile time
- Errors at run time

Errors at Compile Time

A compile error outputs an error which occurs during the validation of PSM statement specified in a proc edure/function.

A syntax error outputs only the location of the first found syntax error as follows, so a user should compil e it again to find out if there are any additional errors.

```
CREATE OR REPLACE PROCEDURE PROC1
IS
BEGIN
  V1 = 1;
  V2 = 1;
END;
/
ERR-42000(40000): syntax error:
  V1 = 1;
.....^
Error at line 4
```

If a syntax error does not exist, then GOLDILOCKS performs a validation while compiling a PSM statemen t and the errors occurred in this case are output as follows.

```
CREATE OR REPLACE PROCEDURE PROC1
IS
BEGIN
  V1 := 1;
  V2 := 2;
END;
/
ERR-01000(16409): Warning: Routine definition has compilation errors
ERR-HY000(17032): PSM compilation error :
```

```
(1) at (4:3): ERR-HY000(17006): unknown variable or column name (V1)
(2) at (5:3): ERR-HY000(17006): unknown variable or column name (V2)
Procedure created.
```

Compile errors output the information of errors within the size which is allowed in the error output buffe r of GOLDILOCKS. Therefore, some errors may not be output. If so, it is output in the following form to n otify that there are errors which are not output.

```
CREATE OR REPLACE PROCEDURE PROC1
IS
BEGIN
V1 := 1;
V2 := 2;
V3 := 2;
V4 := 2;
V5 := 2;
V6 := 2;
V7 := 2;
V8 := 2;
V10 := 2;
END;
/
ERR-01000(16409): Warning: Routine definition has compilation errors
ERR-HY000(17032): PSM compilation error :
(1) at (4:3): ERR-HY000(17006): unknown variable or column name (V1)
(2) at (5:3): ERR-HY000(17006): unknown variable or column name (V2)
(3) at (6:3): ERR-HY000(17006): unknown variable or column name (V3)
(4) at (7:3): ERR-HY000(17006): unknown variable or column name (V4)
(5) at (8:3): ERR-HY000(17006): unknown variable or column name (V5)
(6) at (9:3): ERR-HY000(17006): unknown variable or column name (V6)
(7) at (10:3): ERR-HY000(17006): unknown variable or column name (V7)
2 more errors...
```

An error message is output in the following form.

```
(Index) at (Line_Number : Column_Position): ERR-SQLState( Internal_ErrorCode):
Detail_Error_Message
```

- Index
  - It is a sequence of an error occurrence.
- Line_Number
  - It is a location of a line of source in which an error occurred.

- Column_Position
  - It is a location of a column of a source in which an error occurred.
- ERR-SQLState
  - It is a standard SQLState.
- Internal_ErrorCode
  - It is an internal error code.
- Detail_Error_Message
  - It is details of an error message.

Errors at Run-time

An error may occur in a PSM statement for various reasons even when PSM is normally executed. In this case, GOLDILOCKS provides an exception handling to help a user to control the error.

The following is an example of an error occurrence at the time of execution. If an error occurs when executing GOLDILOCKS PMS, then it outputs a location of an error, a PSM error and an internal error together. In other words, one or more errors may occur, so all errors should be retrieved from the database through a function such as SQLGetDiagRec if a user wants to find out the exact error message during the development process such as ODBC.

```
DECLARE
  V1 INTEGER;
BEGIN
  V1 := 1 / 0;
END;
/
ERR-HY000(17007): invalid expression :
  V1 := 1 / 0;
        *
ERROR at line 4:
ERR-22012(12122): divisor is equal to zero
```

If an error occurs at the time of operation, PSM immediately stops the operation and notifies it to a user. However, if a user wants to directly control these errors and keep the program running, the user should perform the exception handling.

The following is an example of definition to run the user program without interruption by performing the exception handling on the location of an error.

```
DECLARE
  V1 INTEGER;
BEGIN
  BEGIN
    V1 := 1 / 0;
  EXCEPTION WHEN OTHERS
            THEN DBMS_OUTPUT.PUT_LINE('SQLCODE = ' || SQLCODE);
                 DBMS_OUTPUT.PUT_LINE('SQLERRM = ' || SQLERRM);
  END;

  DBMS_OUTPUT.PUT_LINE('next code');
END;
/
SQLCODE = -12122
SQLERRM = [SUNJESOFT][PSM][GOLDILOCKS]divisor is equal to zero
next code
Anonymous PL block executed.
```

For more information, refer to **EXCEPTION Handling**, **SQLCODE Function**, **SQLERRM Function**.

Cursor Attributes

Cursor attributes are attributes (variables) which are provided to display the cursor status specified by a us er or to display the inside of the database, while processing PSM.

- Cursors are specified as follows.
  - Implicit cursor
    - It is a cursor which is OPEN/ FETCH/ CLOSE inside of the database when it is required.
  - Explicit cursor
    - It is a cursor which OPEN/ FETCH /CLOSE after when a user explicitly perform declaration/ de finition.

Implicit Cursor Attributes

Implicit cursor attributes in PSM is used to find out the processing status of a SQL statement which was p roximately performed. Each value of attributes can be used in expressions such as a conditional expressio n, so a user can control it such as branching logics of a program through that value.
For more information about attributes, refer to the following table.

Table 23-2 Implicit cursor attributes

Attribute name	Return type	Description
ISOPEN	BOOLEAN	It is always FALSE because it is internally closed.
FOUND	BOOLEAN	If the data is returned by the previous statement, then it is TRUE. Otherwise, it is FALSE.
NOTFOUND	BOOLEAN	It is the opposite value of FOUND.
ROWCOUNT	INTEGER	It is the number of rows affected by the previous statement.

The syntax is used in the following form.

```
SQL%Attribute_name
Attribute_name := ISOPEN ¦ FOUND ¦ NOTFOUND ¦ ROWCOUNT
```

Each attribute can be viewed as the following example.

```
gSQL> CREATE TABLE T1
(
  C1 VARCHAR(20),
  C2 VARCHAR(20)
);
Table created.
gSQL> INSERT INTO T1 VALUES ('Seoul', '24');
1 row created.
gSQL> INSERT INTO T1 VALUES ('Pusan', '44');
1 row created.
gSQL> BEGIN
    UPDATE T1 SET C2 = C2 + 1;

    DBMS_OUTPUT.PUT_LINE('ISOPEN   = ' ¦¦ SQL%ISOPEN );
    DBMS_OUTPUT.PUT_LINE('FOUND    = ' ¦¦ SQL%FOUND );
    DBMS_OUTPUT.PUT_LINE('NOTFOUND = ' ¦¦ SQL%NOTFOUND );
    DBMS_OUTPUT.PUT_LINE('ROWCOUNT = ' ¦¦ SQL%ROWCOUNT );
END;
/
ISOPEN   = FALSE
FOUND    = TRUE
NOTFOUND = FALSE
ROWCOUNT = 2
Anonymous PL block executed.
```

Explicit Cursor Attributes

Explicit cursor attributes is used to enquire the status of an explicit cursor.
For more information about attributes, refer to the following table.

Table 23-3 Cursor attributes

Attributes	Return type	Description
%ISOPEN	BOOLEAN	It is TRUE only when a cursor is normally open. Otherwise, it is FALSE.
%FOUND	BOOLEAN	It is NULL before FETCH, and it is TRUE when FETCH is normally performed. I t is FALSE when data does not exist, and it is NULL after CLOSE.
%NOTFOUND	BOOLEAN	It is the opposite value of %FOUND.
%ROWCOUNT	INTEGER	It is NULL before OPEN, and it is 0 when it is normally OPEN. It increases by 1 whenever FETCH succeeds.

It is used as follows within PSM.

```
Cursor_Name % Attribute_name
Attribute_name :=  ISOPEN
                 ¦ FOUND
                 ¦ NOTFOUND
                 ¦ ROWCOUNT
```

The following is an example of using an explicit cursor attribute.

```
gSQL> CREATE TABLE T1
(
  C1 VARCHAR(20),
  C2 VARCHAR(20)
);
Table created.
gSQL> INSERT INTO T1 VALUES ('Seoul', '24');
1 row created.
gSQL> INSERT INTO T1 VALUES ('Pusan', '44');
1 row created.
gSQL> DECLARE
  CURSOR C1 IS SELECT * FROM T1;
  v1 c1%ROWTYPE;
BEGIN
```

- Check if the cursor is open through ISOPEN.

```
IF C1%ISOPEN = FALSE
  THEN
      OPEN C1;
  END IF;
  LOOP
      FETCH C1 INTO v1;
```

- Check if there is any more data through NOTFOUND.

```
EXIT WHEN C1%NOTFOUND;
```

- Find out the number of fetched rows through ROWCOUNT.

```
DBMS_OUTPUT.PUT_LINE('COUNT = ' || C1%ROWCOUNT );
  END LOOP;
  CLOSE C1;

END;
/
COUNT = 1
COUNT = 2
Anonymous PL block executed.
```

EXCEPTION Handling

Exceptions within PSM is a definition of various errors occurred while a user is executing PSM. If an error occurs at the time of operation, PSM immediately stops the operation and returns an error to a user. EXC EPTION handling is a function provided to a user to control EXCEPTION situation and keep the program r unning without interruption.

Exception Handler

An error situation can be controlled through an exception handler defined in PSM BLOCK as follows.

```
DECLARE
  V1 INTEGER;
BEGIN
  BEGIN
    V1 := 1 / 0;
```

- Exception handler

```
EXCEPTION WHEN OTHERS
            THEN DBMS_OUTPUT.PUT_LINE('SQLCODE = ' || SQLCODE);
                 DBMS_OUTPUT.PUT_LINE('SQLERRM = ' || SQLERRM);
  END;

  DBMS_OUTPUT.PUT_LINE('next code');
END;
/
SQLCODE = -12122
SQLERRM = [SUNJESOFT][PSM][GOLDILOCKS]divisor is equal to zero
next code
```

In the example above, the program is stopped by *divide by zero* at the time of execution. However, if an e rror occurred by an exception handler, then only the corresponding BLOCK stops, but the processing cont inues from the next location.

An exception handler is specified as follows.

```
BEGIN
EXCEPTION WHEN exception_name, [exception_name, ...] THEN pl_statements;
         [WHEN exception_name, [exception_name, ...] THEN pl_statements;
END
```

- Features of an exception handler are as follows.
  - A single exception handler is specified in BLOCK.
  - One or more Exception_Name can be specified and its operation can be defined within an except ion handler.
  - Exception_name within an exception handler should not be duplicate.
  - It is operated only for an error within PL BLOCK SCOPE in which an exception handler is specified.
  - OTHERS indicating all exceptional situations should be unique, and should be specified at last.
  - If an exception handler is normally completed, the previous errors are cleared. Therefore, a user s hould store them in a separate PSM variable if in need.

An exception in GOLDILOCKS has the following properties.

Table 23-4 EXCEPTION types

Type	Definer	Error code	Name	Raise implicitly	Raise explicitly
Predefined	Database	O	O	O	Optionally
User defined	User	User-defined	User defined	X	O

For more information about a predefined exception or a user defined exception provided by GO LDILOCKS, refer to **Exception Declaration**.

Propagating Exception

If an appropriate exception handler does not exist in the current BLOCK when an exception occurs, then i t is propagated to an exception handler of the superordinate BLOCK until the exception is processed.

If an error occurred on LINE_1 in the following code, the result varies upon whether an exception handler operates.

```
BEGIN
  ...
  BEGIN
    LINE_1
    EXCEPTION HANDLER_1
  END;
  LINE_2
  EXCEPTION HANDLER_2
END;
/
```

- When it is processed in EXCEPTION_HANDLER_1
  - If the handler operation is normally completed, then it is performed from LINE_2.
- When an appropriate handler does not exist in EXCEPTION_HANDLER_1
  - It can not perform LINE_2, and the currently occurred EXCEPTION is propagated to EXCEPTION_ HANDLER_2.
    - When an appropriate handler exists in EXCEPTION_HANDLER_2, if a handler operation is nor mally completed, then the program ends.
    - If a handler does not exist in EXCEPTION_HANDLER_2, the program stops and the currently occurred error is returned to a user.

If an error occurred in an exception handler which was processing an exception, then the error code is propagated to a superordinate BLOCK.

```
DECLARE
  V1 INTEGER;
  E1 EXCEPTION;
  E2 EXCEPTION;
BEGIN
```

```
  BEGIN
    RAISE E1;
  EXCEPTION WHEN E1 THEN V1 := 1 / 0;    ❶ A new error occurs.
  END;
  EXCEPTION WHEN E1
            THEN DBMS_OUTPUT.PUT_LINE('User Exception');
            WHEN ZERO_DIVIDE
            THEN DBMS_OUTPUT.PUT_LINE('Zero divide');
END;
/
Zero divide
Anonymous PL block executed.
```

User Defined Exception

A user defined exception is an exception used by setting an exception name and an error-code by a user except for a predefined. User defined exception can not implicitly occur, but a user should execute it by e xplicitly using a RAISE statment.

The following is an example of causing an exception by using a RAISE statement.

```
DECLARE
  V1   INTEGER;
  high EXCEPTION;
  low  EXCEPTION;
BEGIN
  BEGIN
      V1 := 10;
      IF V1 > 10
      THEN
          RAISE high;
      ELSE
          RAISE low;
      END IF;
  EXCEPTION WHEN high
            THEN DBMS_OUTPUT.PUT_LINE('High');
            WHEN low
            THEN DBMS_OUTPUT.PUT_LINE('Low');
  END;
END;
/
Low
```

- A user defined exception is processed depending on the following two cases.
  - When an error-code is set
  - When an error-code is not set

It is operated as follows when a user sets an error code.

```
DECLARE
  V1 INTEGER;
  E1 EXCEPTION;
  PRAGMA EXCEPTION_INIT( E1, -12122);
BEGIN
  BEGIN
    V1 := 1 / 0;
  EXCEPTION WHEN E1
            THEN DBMS_OUTPUT.PUT_LINE('SQLCODE = ' || SQLCODE);
                 DBMS_OUTPUT.PUT_LINE('SQLERRM = ' || SQLERRM);
  END;
END;
/
SQLCODE = -12122
SQLERRM = [SUNJESOFT][PSM][GOLDILOCKS]divisor is equal to zero
```

The ZERO_DIVIDE predefined exception exists, but a user sets a handler by setting an error code as E1 ex ception. For the program compatibility between vendors with each different error code as above, an exce ption handler is available of which a user directly resets an error code.

A user defined exception in which an error-code is not set is set as the following error code and message in an exception handler.

```
DECLARE
  V1 INTEGER;
  E1 EXCEPTION;
BEGIN
  BEGIN
    RAISE E1;
  EXCEPTION WHEN E1
            THEN DBMS_OUTPUT.PUT_LINE('SQLCODE = ' || SQLCODE);
                 DBMS_OUTPUT.PUT_LINE('SQLERRM = ' || SQLERRM);
  END;
END;
/
SQLCODE = 1
```

```
SQLERRM = [SUNJESOFT][PSM][GOLDILOCKS]User-Defined Exception
Anonymous PL block executed.
```

A propagation process of a user defined exception is as same as that of a predefined exception. However, if an error-code is not set in a user defined exception, it is propagated as unhandled user exception error to the superordinate BLOCK.

```
DECLARE
  V1 INTEGER;
  E1 EXCEPTION;
  E2 EXCEPTION;
BEGIN
  BEGIN
    RAISE E1;
  EXCEPTION WHEN E2
            THEN DBMS_OUTPUT.PUT_LINE('SQLCODE = ' || SQLCODE);
                 DBMS_OUTPUT.PUT_LINE('SQLERRM = ' || SQLERRM);
  END;
END;
/
ERR-HY000(17017): Unhandled user exception :
    RAISE E1;
          *
ERROR at line 7:
```

An unhandled user exception can be caught by specifying a user defined exception in an exception handler or by using OTHERS which is a predefined exception.

PRAGMA EXCEPTION_INIT

It is used for a user sets a specific DBMS error code when using a user defined exception. DBMS error codes varies depending on vendors, so it can be used for a compatibility of an exception handler of PSM code.

The syntax is as follows.

```
<PRAGMA EXCEPTION_INIT> ::=
    PARAGMA EXCEPTION_INIT ( exception_name, internal_errorcode )
```

- Constraints are as follows.
  - exception_name should be declared in advance.

- exception_name can not use a predefined exception.
- Only an internal error-code of GOLDILOCKS is available for internal_errorcode.
- It can not set 0 which means SUCCESS.

The following is an example of handling an exception of when assigning NULL to V1 (the PSM variable to which NOT NULL constraints are applied) by using a user defined exception.

```
DECLARE
  V1 VARCHAR(20) NOT NULL := 10;
  USER_EXCEPT EXCEPTION;
```

- Declare the PSM NOT NULL constraint error as a user defined exception.

```
PRAGMA EXCEPTION_INIT( USER_EXCEPT, -17009 );
BEGIN
```

- Generate a situation of an exception.

```
V1 := NULL;
  EXCEPTION WHEN USER_EXCEPT THEN DBMS_OUTPUT.PUT_LINE('Check Value');
END;
/
Check Value
Anonymous PL block executed.
```

The following error occurs when setting a value which is not an internal error-code of GOLDILOCKS.

```
DECLARE
  E1 EXCEPTION;
  PRAGMA EXCEPTION_INIT( E1, 99999);
BEGIN
    NULL;
END;
/
ERR-HY000(17032): PSM compilation error :
(1) at (3:30): ERR-HY000(17021): Invalid error number for PRAGMA EXCEPTION_INIT
```

DBMS_STANDARD.RAISE_APPLICATION_ERROR

It is used to easily generate an exception through an arbitrary user error code and a message, instead of e xplicitly defining a user exception. It belongs to a DBMS_STANDARD module, and the routines of this mo dule can be called even when omitting a name of that module.

There are three types of arguments as follows.

Table 23-5 RAISE_APPLICATION_ERROR

Argument	Description
error_code IN NATIVE_INT EGER	It is a value of an error code to be arbitrarily generated. Only the value within the rang e of -20000 ~ -20999 can be used.
error_message IN VARCHA R(4000)	It is a message to be stored in SQLERRM when an error occurs.
stack_flag IN BOOLEAN := FALSE	It is a flag of whether to stack on the existing errors (TRUE), or to replace with all exist ing errors (FALSE). The default value is FALSE.

The following is a simple example of using it.

```
CREATE OR REPLACE PROCEDURE PROC1
IS
BEGIN
  RAISE_APPLICATION_ERROR (-20000, 'USER DEFINED ERROR TEST');
END;
/
Procedure created.
BEGIN
  PROC1;
EXCEPTION WHEN OTHERS THEN
  DBMS_OUTPUT.PUT_LINE( 'SQLCODE : ' || SQLCODE || ' SQLERRM : ' || SQLERRM );
END;
/
SQLCODE : -20000 SQLERRM : [SUNJESOFT][PSM][GOLDILOCKS]USER DEFINED ERROR TEST
Anonymous PL block executed.
```

24.

PSM Cursor Statements

24.1 Implicit Cursor

An implicit cursor is defined and managed by PSM statement in the database. PSM statement uses an im plicit cursor whenever executing SELECT statement or DML statement. A user can not control an implicit cursor, but it can obtain the information about the query through an implicit cursor attribute.

Implicit Cursor Attributes

An implicit cursor attributes refer to the execution result of the most recently executed SELECT statement or DML statement. If neither SELECT statement nor is DML statement recently executed, then the attribut es values are NULL.

Table 24-1 Implicit cursor attributes

Attribute name	Return type	Description
SQL%ISOPEN	BOOLEAN	It always returns FALSE, because the most recently executed SELECT statem ent or DML statement is always terminated.
SQL%FOUND	BOOLEAN	It it TRUE when the most recently executed SELECT statement or DML state ment returns a result. Otherwise, it is FALSE.
SQL%NOTFOUN D	BOOLEAN	It has the opposite value of SQL%FOUND.
SQL%ROWCOU NT	INTEGER	It is the number of ROWs returned from the most recently executed SELECT statement or DML statement.

Examples

- When the most recently executed query does not exist

```
gSQL>
BEGIN
  DBMS_OUTPUT.PUT_LINE('SQL%ISOPEN   = ' || SQL%ISOPEN);
  DBMS_OUTPUT.PUT_LINE('SQL%FOUND    = ' || SQL%FOUND);
  DBMS_OUTPUT.PUT_LINE('SQL%NOTFOUND = ' || SQL%NOTFOUND);
  DBMS_OUTPUT.PUT_LINE('SQL%ROWCOUNT = ' || SQL%ROWCOUNT);
END;
/
SQL%ISOPEN   = FALSE
```

```
SQL%FOUND    =
SQL%NOTFOUND =
SQL%ROWCOUNT =
Anonymous PL block executed.
```

- When the most recently executed query exists

```
gSQL>
CREATE TABLE t_month( month_char VARCHAR(15) , month_num INTEGER );
Table created.
gSQL>
INSERT INTO t_month VALUES( 'JANUARY' , 1 ), ( 'MARCH' , 3 ),
                          ( 'MAY' , 5 ), ( 'JULY' , 7 ),
                          ( 'SEPTEMBER' , 9 ), ( 'NOVEMBER' , 11 );
6 rows created.
gSQL>
DECLARE
  row_count INTEGER;
BEGIN
  SELECT COUNT(*) INTO row_count FROM t_month;
  DBMS_OUTPUT.PUT_LINE('SQL%ISOPEN   = ' || SQL%ISOPEN);
  DBMS_OUTPUT.PUT_LINE('SQL%FOUND    = ' || SQL%FOUND);
  DBMS_OUTPUT.PUT_LINE('SQL%NOTFOUND = ' || SQL%NOTFOUND);
  DBMS_OUTPUT.PUT_LINE('SQL%ROWCOUNT = ' || SQL%ROWCOUNT);
END;
/
SQL%ISOPEN   = FALSE
SQL%FOUND    = TRUE
SQL%NOTFOUND = FALSE
SQL%ROWCOUNT = 1
Anonymous PL block executed.
```

24.2 Explicit Cursor

An explicit cursor is declared, defined and managed by a user. The user specifies the cursor name and sho uld connect to SELECT statement or DML statement.

- Use an explicit cursor as follows.
  - The user open the explicit cursor through open statement, and fetches the query result by execut ing the fetch statement. Last, it closes the cursor with close statement.
  - The explicit cursor is available for *cursor for loop* statement.

The explicit cursor is not available for an expression. In other words, it can not assign the value to the curs or, nor is it used as a parameter of the procedure or the routine.

- Example of Explicit Cursor

```
gSQL>
CREATE TABLE t_month( month_char VARCHAR(15), month_num INTEGER );
Table created.
gSQL>
INSERT INTO t_month VALUES( 'FEBRUARY' , 2 ), ( 'APRIL' , 4 ),
                          ( 'JUNE' , 6 ), ( 'AUGUST' , 8 ),
                          ( 'OCTOBER' , 10 ), ( 'DECEMBER' , 12 );
6 rows created.
gSQL>
DECLARE
  CURSOR cur_month IS SELECT * FROM t_month;
  v_month_char VARCHAR(15);
  v_month_num  INTEGER;
BEGIN
  OPEN cur_month;                                          ❶ OPEN explicit cursor.
LOOP                                                       ❷ Fetch multiple ROWs by using loop.
    FETCH cur_month INTO v_month_char, v_month_num;        ❸ FETCH explicit cursor.

    EXIT WHEN cur_month%NOTFOUND;                          ❹ Check if all ROWs in the result
set are fetched.

    DBMS_OUTPUT.PUT_LINE( 'v_month_char : ' || v_month_char || ' , v_month_num : ' ||
v_month_num );
  END LOOP;

  CLOSE cur_month;                                         ❺ CLOSE explicit cursor.
```

```
END;
/
v_month_char : FEBRUARY , v_month_num : 2
v_month_char : APRIL , v_month_num : 4
v_month_char : JUNE , v_month_num : 6
v_month_char : AUGUST , v_month_num : 8
v_month_char : OCTOBER , v_month_num : 10
v_month_char : DECEMBER , v_month_num : 12
Anonymous PL block executed.
```

Declaring and Defining Explicit Cursors

An explicit cursor should be declared and defined in PSM declare section to use it. The cursor name, the parameter information and the return type of when declaring the explicit cursor should be as same as wh en defining the explicit cursor. For more information about the syntax of the explicit cursor and others, re fer to **Explicit Cursor Declaration and Definition**.

Example of Declaring and Defining Explicit Cursor

```
gSQL>
DECLARE
  CURSOR cur_month1( p1 INTEGER ) RETURN t_month%rowtype;
  CURSOR cur_month1( p1 INTEGER ) RETURN t_month%rowtype IS SELECT * FROM t_month WHERE
month_num = p1;
  CURSOR cur_month2 IS SELECT * FROM t_month;
BEGIN
  NULL;
END;
/
Anonymous PL block executed.
```

Opening and Closing Explicit Cursors

- Perform the cursor query by executing **OPEN Statement** after declaring and defining the explicit curs or.
- If the query of the explicit cursor is SELECT ... FOR UPDATE statement, then it locks the ROW in the re sult set

- If the query of the explicit cursor refers to an explicit cursor parameter or a PSM variable, then it affec ts the query result.
- Close the explicit cursor by using **CLOSE Statement** after using the cursor. It can not fetch the record from the result set after the explicit cursor is closed.
- Once after the explicit cursor is OPENed it can not be reopened, CLOSE it first to OPEN it again.

Example of Explicit Cursor OPEN and CLOSE

```
gSQL>
DECLARE
  CURSOR cur_month( p1 INTEGER ) IS SELECT * FROM t_month WHERE month_num = p1;
BEGIN
  OPEN cur_month( 2 );      -- Explicit Cursor OPEN
  CLOSE cur_month;          -- Explicit Cursor CLOSE
END;
/
Anonymous PL block executed.
```

Fetching Data With Explicit Cursors

- It fetches the row from the result set by executing **FETCH Statement** after OPENing the explicit cursor.
- Fetch statement retrieves the current ROW of the result set and stores the ROW value in PSM variable or in record type variable, then moves the cursor to the next ROW.
- PSM **FETCH Statement** does not throw an exception even when there is not a fetched ROW.
- When performing FETCH in **Basic LOOP Statement**, then use %NOTFOUND property to detect the ter minating condition.

Example of FETCHing Explicit Cursor

```
gSQL>
DECLARE
  CURSOR cur_month IS SELECT * FROM t_month;
  v_month_char VARCHAR(15);
  v_month_num  INTEGER;
BEGIN
  OPEN cur_month;

  LOOP
    FETCH cur_month INTO v_month_char, v_month_num;
```

```
    EXIT WHEN cur_month%NOTFOUND;

    DBMS_OUTPUT.PUT_LINE( 'v_month_char : ' || v_month_char || ' , v_month_num : ' ||
v_month_num );
  END LOOP;

  CLOSE cur_month;
END;
/
v_month_char : FEBRUARY , v_month_num : 2
v_month_char : APRIL , v_month_num : 4
v_month_char : JUNE , v_month_num : 6
v_month_char : AUGUST , v_month_num : 8
v_month_char : OCTOBER , v_month_num : 10
v_month_char : DECEMBER , v_month_num : 12
Anonymous PL block executed.
```

When Explicit Cursor Queries Need Column Aliases

If the query of an explicit cursor includes an expression, then the column always needs an alias name. The fetched result is stored in %ROWTYPE variable created by referring to the explicit cursor, and it can be us ed as a reference.

Example of Using Alias in Query of Explicit Cursor

```
gSQL>
DECLARE
  CURSOR cur_month IS SELECT month_char, ( ' == ' || month_num ) as equal_month_num FROM
t_month;
  var cur_month%rowtype;
BEGIN
  OPEN cur_month;

  LOOP
    FETCH cur_month INTO var;

    EXIT WHEN cur_month%NOTFOUND;
```

```
    DBMS_OUTPUT.PUT_LINE( var.month_char || var.equal_month_num );
  END LOOP;
  CLOSE cur_month;
END;
/
FEBRUARY == 2
APRIL == 4
JUNE == 6
AUGUST == 8
OCTOBER == 10
DECEMBER == 12
Anonymous PL block executed.
```

Explicit Cursors that Accept Parameters

It can declare and define the explicit cursor having the parameter.

- Parameter of the explicit cursor
  - It allows IN bind type only.
  - It can have DEFAULT value. It is not necessary to specify an argument.
  - It can be referenced in the query of the cursor.
  - If it is out of the explicit cursor's range, then it can not refer to the parameter of the explicit curso r.

Example of Using Explicit Cursor Parameter

```
gSQL>
DECLARE
  CURSOR cur_month( p1 IN INTEGER ) IS SELECT month_char, month_num FROM t_month WHERE
month_num = p1;
  v_month_char VARCHAR(15);
  v_month_num  INTEGER;
BEGIN
  OPEN cur_month(4);

  FETCH cur_month INTO v_month_char, v_month_num;
  DBMS_OUTPUT.PUT_LINE( 'v_month_char : ' || v_month_char || ' , v_month_num : ' ||
v_month_num );
gSQL>
```

```
  CLOSE cur_month;
END;
/
v_month_char : APRIL , v_month_num : 4
Anonymous PL block executed.
```

- Example of Explicit Cursor Parameter's DEFAULT value

```
gSQL>
DECLARE
  CURSOR cur_month( p1 INTEGER DEFAULT 2 ) IS SELECT month_char, month_num FROM t_month WHERE
month_num = p1;
  v_month_char VARCHAR(15);
  v_month_num  INTEGER;
BEGIN
  OPEN cur_month;                    ❶ It is allowed to omit the actual parameter.

  FETCH cur_month INTO v_month_char, v_month_num;
  DBMS_OUTPUT.PUT_LINE( 'v_month_char : ' || v_month_char || ' , v_month_num : ' ||
v_month_num );
  CLOSE cur_month;

  OPEN cur_month(4);          ❷ It can be OPENed by specifying the actual parameter.

  FETCH cur_month INTO v_month_char, v_month_num;
  DBMS_OUTPUT.PUT_LINE( 'v_month_char : ' || v_month_char || ' , v_month_num : ' ||
v_month_num );
  CLOSE cur_month;
END;
/
v_month_char : FEBRUARY , v_month_num : 2
v_month_char : APRIL , v_month_num : 4
Anonymous PL block executed.
```

Explicit Cursor Attributes

The explicit cursor attribute describes the status of an explicit cursor. It can be used as an explicit cursor a ttribute by adding the attribute after the cursor name.

Table 24-2 Explicit cursor attributes

Attribute name	Return type	Description
explicit_cursor_name%ISOPEN	BOOLEAN	If the explicit cursor is normally OPEN, then it is TRUE. Otherwise, it is FALSE.
explicit_cursor_name%FOUND	BOOLEAN	If the explicit cursor is not FETCHed yet, then it is NULL. If the data exists, then it is TRUE. If the data does not exist, then it is FALSE. If it is CLOSEd, then it is NULL.
explicit_cursor_name%NOTFOUND	BOOLEAN	It has an opposite value of explicit_cursor_name%FOUND.
explicit_cursor_name%ROWCOUNT	INTEGER	If the explicit cursor is not OPEN yet, then it is NULL. If it is normally OPEN, then it is 0. It is increased by 1 whenever executing FETCH, and it is NULL after CLOSE.

Example of Using Explicit Cursor Parameter

- Before OPEN the explicit cursor

```
gSQL>
DECLARE
  CURSOR cur_month IS SELECT month_char, month_num FROM t_month;
BEGIN
  DBMS_OUTPUT.PUT_LINE('cur_month%ISOPEN   = ' || cur_month%ISOPEN);
  DBMS_OUTPUT.PUT_LINE('cur_month%FOUND    = ' || cur_month%FOUND);
  DBMS_OUTPUT.PUT_LINE('cur_month%NOTFOUND = ' || cur_month%NOTFOUND);
  DBMS_OUTPUT.PUT_LINE('cur_month%ROWCOUNT = ' || cur_month%ROWCOUNT);
END;
/
cur_month%ISOPEN   = FALSE
cur_month%FOUND    =
cur_month%NOTFOUND =
cur_month%ROWCOUNT =
Anonymous PL block executed.
```

- After OPEN the explicit cursor

```
gSQL>
DECLARE
  CURSOR cur_month IS SELECT month_char, month_num FROM t_month;
BEGIN
```

```
  OPEN cur_month;

  DBMS_OUTPUT.PUT_LINE('cur_month%ISOPEN   = ' || cur_month%ISOPEN);
  DBMS_OUTPUT.PUT_LINE('cur_month%FOUND    = ' || cur_month%FOUND);
  DBMS_OUTPUT.PUT_LINE('cur_month%NOTFOUND = ' || cur_month%NOTFOUND);
  DBMS_OUTPUT.PUT_LINE('cur_month%ROWCOUNT = ' || cur_month%ROWCOUNT);
  CLOSE cur_month;
END;
/
cur_month%ISOPEN   = TRUE
cur_month%FOUND    =
cur_month%NOTFOUND =
cur_month%ROWCOUNT = 0
Anonymous PL block executed.
```

- After FETCH the explicit cursor

```
gSQL>
DECLARE
  CURSOR cur_month IS SELECT month_char, month_num FROM t_month LIMIT 3;
  v_month_char VARCHAR(15);
  v_month_num  INTEGER;
BEGIN
  OPEN cur_month;
  DBMS_OUTPUT.PUT_LINE( '[ First Fetch ]' );

  FETCH cur_month INTO v_month_char, v_month_num;
  DBMS_OUTPUT.PUT_LINE('cur_month%ISOPEN   = ' || cur_month%ISOPEN);
  DBMS_OUTPUT.PUT_LINE('cur_month%FOUND    = ' || cur_month%FOUND);
  DBMS_OUTPUT.PUT_LINE('cur_month%NOTFOUND = ' || cur_month%NOTFOUND);
  DBMS_OUTPUT.PUT_LINE('cur_month%ROWCOUNT = ' || cur_month%ROWCOUNT);
  DBMS_OUTPUT.PUT_LINE( '[ Second Fetch ]' );
  FETCH cur_month INTO v_month_char, v_month_num;
  DBMS_OUTPUT.PUT_LINE('cur_month%ISOPEN   = ' || cur_month%ISOPEN);
  DBMS_OUTPUT.PUT_LINE('cur_month%FOUND    = ' || cur_month%FOUND);
  DBMS_OUTPUT.PUT_LINE('cur_month%NOTFOUND = ' || cur_month%NOTFOUND);
  DBMS_OUTPUT.PUT_LINE('cur_month%ROWCOUNT = ' || cur_month%ROWCOUNT);
  DBMS_OUTPUT.PUT_LINE( '[ Last Fetch ]' );

  FETCH cur_month INTO v_month_char, v_month_num;
```

```
  DBMS_OUTPUT.PUT_LINE('cur_month%ISOPEN   = ' || cur_month%ISOPEN);
  DBMS_OUTPUT.PUT_LINE('cur_month%FOUND    = ' || cur_month%FOUND);
  DBMS_OUTPUT.PUT_LINE('cur_month%NOTFOUND = ' || cur_month%NOTFOUND);
  DBMS_OUTPUT.PUT_LINE('cur_month%ROWCOUNT = ' || cur_month%ROWCOUNT);
  DBMS_OUTPUT.PUT_LINE( '[ Over Last Fetch ]' );
  FETCH cur_month INTO v_month_char, v_month_num;
  DBMS_OUTPUT.PUT_LINE('cur_month%ISOPEN   = ' || cur_month%ISOPEN);
  DBMS_OUTPUT.PUT_LINE('cur_month%FOUND    = ' || cur_month%FOUND);
  DBMS_OUTPUT.PUT_LINE('cur_month%NOTFOUND = ' || cur_month%NOTFOUND);
  DBMS_OUTPUT.PUT_LINE('cur_month%ROWCOUNT = ' || cur_month%ROWCOUNT);
  CLOSE cur_month;
END;
/
[ First Fetch ]
cur_month%ISOPEN   = TRUE
cur_month%FOUND    = TRUE
cur_month%NOTFOUND = FALSE
cur_month%ROWCOUNT = 1
[ Second Fetch ]
cur_month%ISOPEN   = TRUE
cur_month%FOUND    = TRUE
cur_month%NOTFOUND = FALSE
cur_month%ROWCOUNT = 2
[ Last Fetch ]
cur_month%ISOPEN   = TRUE
cur_month%FOUND    = TRUE
cur_month%NOTFOUND = FALSE
cur_month%ROWCOUNT = 3
[ Over Last Fetch ]
cur_month%ISOPEN   = TRUE
cur_month%FOUND    = FALSE
cur_month%NOTFOUND = TRUE
cur_month%ROWCOUNT = 3
Anonymous PL block executed.
```

- After CLOSE the explicit cursor

```
gSQL>
DECLARE
  CURSOR cur_month IS SELECT month_char, month_num FROM t_month;
  v_month_char VARCHAR(15);
```

```
  v_month_num  INTEGER;
BEGIN
  OPEN cur_month;

  FETCH cur_month INTO v_month_char, v_month_num;
  CLOSE cur_month;
  DBMS_OUTPUT.PUT_LINE('cur_month%ISOPEN   = ' || cur_month%ISOPEN);
  DBMS_OUTPUT.PUT_LINE('cur_month%FOUND    = ' || cur_month%FOUND);
  DBMS_OUTPUT.PUT_LINE('cur_month%NOTFOUND = ' || cur_month%NOTFOUND);
  DBMS_OUTPUT.PUT_LINE('cur_month%ROWCOUNT = ' || cur_month%ROWCOUNT);
END;
/
cur_month%ISOPEN   = FALSE
cur_month%FOUND    =
cur_month%NOTFOUND =
cur_month%ROWCOUNT =
Anonymous PL block executed.
```

24.3 Cursor Variable

Create Cursor Variables

Declare the variable by declaring predefined SYS_REFCURSOR type variable or by defining REF CURSOR TYPE so that the cursor variable is created. For more information, refer to **Cursor Variable Declaration**.

- REF CURSOR types are as follows.
  - Strong REF CURSOR type
    - If return_type is specified when defining REF CURSOR type, then it is the strong REF CURSOR type.
    - The strong REF CURSOR type variable can connect only the query whose query set type is as same as the defined return_type to the query of CURSOR VARIABLE.
    - Assigning the strong REF CURSOR type variable is available when the defined return_types are same.
  - Weak REF CURSOR type
    - If return_type is not specified when defining REF CURSOR type, then it is the weak REF CURSOR type.
    - SYS_REFCURSOR type is the weak REF CURSOR type as well.
    - It is more flexible than the strong REF CURSOR variable.
    - The weak ref cursor type variable can be assigned as the SYS_REFCURSOR type variable, and the SYS_REFCURSOR type variable can be assigned as the weak REF CURSOR type variable.

Example of Create Cursor Variable

- Definition of REF CURSOR type

```
gSQL>
DECLARE
  TYPE strong_refcur IS REF CURSOR RETURN t_month%ROWTYPE;   ❶ strong ref cursor type
  TYPE weak_refcur IS REF CURSOR;                            ❷ weak ref cursor type
BEGIN
  NULL;
END;
/
Anonymous PL block executed.
```

- Declaration of CURSOR VARIABLE

```
gSQL>
DECLARE
  TYPE strong_refcur IS REF CURSOR RETURN t_month%ROWTYPE;   ❶ strong ref cursor type
  TYPE weak_refcur IS REF CURSOR;                           ❷ weak ref cursor type
  v_sys_refcursor     SYS_REFCURSOR;                        ❸ weak ref cursor type variable
  v_strong_refcursor  strong_refcur;                        ❹ strong ref cursor type
variable
  v_weak_refcursor    weak_refcur;                          ❺ weak ref cursor type variable
BEGIN
  NULL;
END;
/
Anonymous PL block executed.
```

Opening and Closing Cursor Variables

- It connects the cursor variable and SELECT statement or DML statement to perform by performing **OPEN FOR Statement** after declaring the cursor variable.
- The cursor query may have the bind variable designating the specified value in using clause of OPEN FOR statement.
- If FOR UPDATE statement exists in cursor query, then it LOCKs the ROW in the result set.
- When REOPENing the cursor variable, then it does not require CLOSE. The connection to the previous query is cut after REOPENing.
- Perform **CLOSE Statement** after finish using the cursor variable.
- It does not allow accessing the query result of CLOSEd cursor variable or referring to the cursor variable attributes.

Example of Cursor Variable's OPEN and CLOSE

```
DECLARE
  v_refcur_month SYS_REFCURSOR;
BEGIN
  OPEN v_refcur_month FOR SELECT * FROM t_month;
  CLOSE v_refcur_month;
END;
/
Anonymous PL block executed.
```

- Cursor variable's OPEN with using clause.

```
gSQL>
DECLARE
  v_refcur_month SYS_REFCURSOR;
  var            INTEGER;
BEGIN
  var := 1;

  OPEN v_refcur_month FOR 'SELECT * FROM t_month WHERE month_num = :v' USING IN var;
END;
/
Anonymous PL block executed.
```

Fetching Data with Cursor Variables

It can bring the row from the result set by executing **FETCH Statement** after OPENing the cursor variable. The result returned by the cursor variable should be compatible with into clause.

Example of FETCHing Cursor Variable

```
gSQL>
DECLARE
  TYPE ref_month IS REF CURSOR RETURN t_month%ROWTYPE;
  v_refcur_month ref_month;

  v_month        t_month%ROWTYPE;
BEGIN
  OPEN v_refcur_month FOR SELECT * FROM t_month;

  LOOP
    FETCH v_refcur_month INTO v_month;

    EXIT WHEN v_refcur_month%NOTFOUND;

    DBMS_OUTPUT.PUT_LINE( 'v_month.month_char : ' || v_month.month_char || ' ,
v_month.month_num : ' || v_month.month_num );
  END LOOP;

  CLOSE v_refcur_month;
END;
```

```
/
v_month.month_char : JANUARY , v_month.month_num : 1
v_month.month_char : MARCH , v_month.month_num : 3
v_month.month_char : MAY , v_month.month_num : 5
v_month.month_char : JULY , v_month.month_num : 7
v_month.month_char : SEPTEMBER , v_month.month_num : 9
v_month.month_char : NOVEMBER , v_month.month_num : 11
Anonymous PL block executed.
```

Assigning Value to Cursor Variables

The cursor variable can assign another cursor variable value or the host variable value. If the source cursor variable is OPEN and assigns the target cursor variable value, then both cursor variables indicate the same SQL. If the source cursor variable is not OPEN and assigns the target cursor variable value, then both curs or variables are not OPEN.

Example of Assigning Cursor Variable

```
gSQL>
DECLARE
  source_refcur_month SYS_REFCURSOR;
  target_refcur_month SYS_REFCURSOR;
  v_month_char   VARCHAR(15);
  v_month_num    INTEGER;
BEGIN
  DBMS_OUTPUT.PUT_LINE( 'OPEN source_refcur_month' );

  OPEN source_refcur_month FOR SELECT * FROM t_month;
  DBMS_OUTPUT.PUT_LINE( 'source_refcur_month%ISOPEN : ' || source_refcur_month%ISOPEN );
  DBMS_OUTPUT.PUT_LINE( 'target_refcur_month%ISOPEN : ' || target_refcur_month%ISOPEN );
  DBMS_OUTPUT.PUT_LINE( 'Assign source_refcur_month to target_refcur_month' );

  target_refcur_month := source_refcur_month;
  DBMS_OUTPUT.PUT_LINE( 'source_refcur_month%ISOPEN : ' || source_refcur_month%ISOPEN );
  DBMS_OUTPUT.PUT_LINE( 'target_refcur_month%ISOPEN : ' || target_refcur_month%ISOPEN );
  DBMS_OUTPUT.PUT_LINE( 'CLOSE source_refcur_month' );
  CLOSE source_refcur_month;
  DBMS_OUTPUT.PUT_LINE( 'source_refcur_month%ISOPEN : ' || source_refcur_month%ISOPEN );
  DBMS_OUTPUT.PUT_LINE( 'target_refcur_month%ISOPEN : ' || target_refcur_month%ISOPEN );
```

```
END;
/
OPEN source_refcur_month
source_refcur_month%ISOPEN : TRUE
target_refcur_month%ISOPEN : FALSE
Assign source_refcur_month to target_refcur_month
source_refcur_month%ISOPEN : TRUE
target_refcur_month%ISOPEN : TRUE
CLOSE source_refcur_month
source_refcur_month%ISOPEN : FALSE
target_refcur_month%ISOPEN : FALSE
Anonymous PL block executed.
```

Cursor Variable Attributes

Cursor variable attributes are as same as explicit cursor attributes, and it describes the status of the cursor variable. It can be used as a cursor variable attribute by adding the attribute after the cursor variable nam e. Attributes except for cursor_variable_name%ISOPEN attribute cause 'cursor is not defined' error befor e OPEN the cursor variable or after CLOSE the cursor variable.

Table 24-3 Cursor variable attributes

Attribute name	Return type	Description
cursor_variable_name %ISOPEN	BOOLEAN	If the cursor variable is OPEN, then it is TRUE. Otherwise, it is FALSE.
cursor_variable_name %FOUND	BOOLEAN	If the data exists after FETCH, then it is TRUE. If the data does not exis t, then it is FALSE.
cursor_variable_name %NOTFOUND	BOOLEAN	It has an opposite value of cursor_variable_name%FOUND after FETC H.
cursor_variable_name %ROWCOUNT	INTEGER	If the cursor variable is OPEN, then it is 0, and it is increased by 1 whe never executing FETCH.

Example of Using Cursor Variable Attributes

- Before OPEN the cursor variable

```
gSQL>
DECLARE
  v_refcur_month SYS_REFCURSOR;
```

```
BEGIN
  DBMS_OUTPUT.PUT_LINE('v_refcur_month%ISOPEN   = ' || v_refcur_month%ISOPEN);
  DBMS_OUTPUT.PUT_LINE('v_refcur_month%FOUND    = ' || v_refcur_month%FOUND);
  DBMS_OUTPUT.PUT_LINE('v_refcur_month%NOTFOUND = ' || v_refcur_month%NOTFOUND);
  DBMS_OUTPUT.PUT_LINE('v_refcur_month%ROWCOUNT = ' || v_refcur_month%ROWCOUNT);
END;
/
v_refcur_month%ISOPEN   = FALSE
ERR-2F000(17036): cursor is not defined :
  DBMS_OUTPUT.PUT_LINE('v_refcur_month%FOUND    = ' || v_refcur_month%FOUND);
                        *
ERROR at line 5:
```

- After OPEN the cursor variable

```
gSQL>
DECLARE
  v_refcur_month SYS_REFCURSOR;
BEGIN
  OPEN v_refcur_month FOR SELECT * FROM t_month;

  DBMS_OUTPUT.PUT_LINE('v_refcur_month%ISOPEN   = ' || v_refcur_month%ISOPEN);
  DBMS_OUTPUT.PUT_LINE('v_refcur_month%FOUND    = ' || v_refcur_month%FOUND);
  DBMS_OUTPUT.PUT_LINE('v_refcur_month%NOTFOUND = ' || v_refcur_month%NOTFOUND);
  DBMS_OUTPUT.PUT_LINE('v_refcur_month%ROWCOUNT = ' || v_refcur_month%ROWCOUNT);
  CLOSE v_refcur_month;
END;
/
v_refcur_month%ISOPEN   = TRUE
v_refcur_month%FOUND    =
v_refcur_month%NOTFOUND =
v_refcur_month%ROWCOUNT = 0
Anonymous PL block executed.
```

- After FETCH the cursor variable

```
gSQL>
DECLARE
  v_refcur_month SYS_REFCURSOR;
  v_month_char VARCHAR(15);
  v_month_num  INTEGER;
BEGIN
```

```
  OPEN v_refcur_month FOR SELECT * FROM t_month LIMIT 3;
  DBMS_OUTPUT.PUT_LINE( '[ First Fetch ]' );

  FETCH v_refcur_month INTO v_month_char, v_month_num;

  DBMS_OUTPUT.PUT_LINE('v_refcur_month%ISOPEN   = ' || v_refcur_month%ISOPEN);
  DBMS_OUTPUT.PUT_LINE('v_refcur_month%FOUND    = ' || v_refcur_month%FOUND);
  DBMS_OUTPUT.PUT_LINE('v_refcur_month%NOTFOUND = ' || v_refcur_month%NOTFOUND);
  DBMS_OUTPUT.PUT_LINE('v_refcur_month%ROWCOUNT = ' || v_refcur_month%ROWCOUNT);
  DBMS_OUTPUT.PUT_LINE( '[ Second Fetch ]' );

  FETCH v_refcur_month INTO v_month_char, v_month_num;

  DBMS_OUTPUT.PUT_LINE('v_refcur_month%ISOPEN   = ' || v_refcur_month%ISOPEN);
  DBMS_OUTPUT.PUT_LINE('v_refcur_month%FOUND    = ' || v_refcur_month%FOUND);
  DBMS_OUTPUT.PUT_LINE('v_refcur_month%NOTFOUND = ' || v_refcur_month%NOTFOUND);
  DBMS_OUTPUT.PUT_LINE('v_refcur_month%ROWCOUNT = ' || v_refcur_month%ROWCOUNT);
  DBMS_OUTPUT.PUT_LINE( '[ Last Fetch ]' );

  FETCH v_refcur_month INTO v_month_char, v_month_num;

  DBMS_OUTPUT.PUT_LINE('v_refcur_month%ISOPEN   = ' || v_refcur_month%ISOPEN);
  DBMS_OUTPUT.PUT_LINE('v_refcur_month%FOUND    = ' || v_refcur_month%FOUND);
  DBMS_OUTPUT.PUT_LINE('v_refcur_month%NOTFOUND = ' || v_refcur_month%NOTFOUND);
  DBMS_OUTPUT.PUT_LINE('v_refcur_month%ROWCOUNT = ' || v_refcur_month%ROWCOUNT);
  DBMS_OUTPUT.PUT_LINE( '[ Over Last Fetch ]' );

  FETCH v_refcur_month INTO v_month_char, v_month_num;

  DBMS_OUTPUT.PUT_LINE('v_refcur_month%ISOPEN   = ' || v_refcur_month%ISOPEN);
  DBMS_OUTPUT.PUT_LINE('v_refcur_month%FOUND    = ' || v_refcur_month%FOUND);
  DBMS_OUTPUT.PUT_LINE('v_refcur_month%NOTFOUND = ' || v_refcur_month%NOTFOUND);
  DBMS_OUTPUT.PUT_LINE('v_refcur_month%ROWCOUNT = ' || v_refcur_month%ROWCOUNT);
  CLOSE v_refcur_month;
END;
/
[ First Fetch ]
v_refcur_month%ISOPEN   = TRUE
v_refcur_month%FOUND    = TRUE
v_refcur_month%NOTFOUND = FALSE
v_refcur_month%ROWCOUNT = 1
```

```
[ Second Fetch ]
v_refcur_month%ISOPEN   = TRUE
v_refcur_month%FOUND    = TRUE
v_refcur_month%NOTFOUND = FALSE
v_refcur_month%ROWCOUNT = 2
[ Last Fetch ]
v_refcur_month%ISOPEN   = TRUE
v_refcur_month%FOUND    = TRUE
v_refcur_month%NOTFOUND = FALSE
v_refcur_month%ROWCOUNT = 3
[ Over Last Fetch ]
v_refcur_month%ISOPEN   = TRUE
v_refcur_month%FOUND    = FALSE
v_refcur_month%NOTFOUND = TRUE
v_refcur_month%ROWCOUNT = 3
Anonymous PL block executed.
```

- After CLOSE the cursor variable

```
gSQL>
DECLARE
  v_refcur_month SYS_REFCURSOR;
  v_month_char VARCHAR(15);
  v_month_num  INTEGER;
BEGIN
  OPEN v_refcur_month FOR SELECT * FROM t_month LIMIT 3;
  FETCH v_refcur_month INTO v_month_char, v_month_num;

  CLOSE v_refcur_month;

  DBMS_OUTPUT.PUT_LINE('v_refcur_month%ISOPEN   = ' || v_refcur_month%ISOPEN);
  DBMS_OUTPUT.PUT_LINE('v_refcur_month%FOUND    = ' || v_refcur_month%FOUND);
  DBMS_OUTPUT.PUT_LINE('v_refcur_month%NOTFOUND = ' || v_refcur_month%NOTFOUND);
  DBMS_OUTPUT.PUT_LINE('v_refcur_month%ROWCOUNT = ' || v_refcur_month%ROWCOUNT);
END;
/
gSQL>
v_refcur_month%ISOPEN   = FALSE
ERR-2F000(17036): cursor is not defined :
  DBMS_OUTPUT.PUT_LINE('v_refcur_month%FOUND    = ' || v_refcur_month%FOUND);
                        *
```

```
ERROR at line 13:
```

Cursor Variable as Routine Parameter

Using the cursor variable as routine parameter is useful to transfer the query result. The parameter should be IN or IN OUT type to FETCH or CLOSE the cursor variable in the procedure or the function. The parame ter should be OUT or IN OUT type to OPEN the cursor variable. Declaring the ref cursor type in the packag e is useful to transfer the query result of the cursor variable in each different routine.

Example of Using Cursor Variable as Parameter

- Example of routine parameter

```
gSQL>
CREATE OR REPLACE PROCEDURE p_open( p_refcur_month OUT SYS_REFCURSOR ) AS
BEGIN
  OPEN p_refcur_month FOR SELECT * FROM t_month;
END;
/
Procedure created.
gSQL>
CREATE OR REPLACE PROCEDURE p_fetch( p_refcur_month IN SYS_REFCURSOR ) AS
  v_month_char VARCHAR(15);
  v_month_num  INTEGER;
BEGIN
  FETCH p_refcur_month INTO v_month_char, v_month_num;
  DBMS_OUTPUT.PUT_LINE( 'v_month_char : ' || v_month_char || ' , v_month_num : ' ||
v_month_num );
END;
/
Procedure created.
gSQL>
DECLARE
  v_refcur_month SYS_REFCURSOR;
BEGIN
  p_open( v_refcur_month );
  DBMS_OUTPUT.PUT_LINE( 'v_refcur_month%ISOPEN : ' || v_refcur_month%ISOPEN );

  p_fetch( v_refcur_month );
```

```
  DBMS_OUTPUT.PUT_LINE( 'v_refcur_month%FOUND : ' || v_refcur_month%FOUND );

  CLOSE v_refcur_month;
  DBMS_OUTPUT.PUT_LINE( 'v_refcur_month%ISOPEN : ' || v_refcur_month%ISOPEN );
END;
/
v_refcur_month%ISOPEN : TRUE
v_month_char : JANUARY , v_month_num : 1
v_refcur_month%FOUND : TRUE
v_refcur_month%ISOPEN : FALSE
Anonymous PL block executed.
```

- Example of using package's ref cursor type

```
gSQL>
CREATE OR REPLACE PACKAGE pkg_refcur AS
  TYPE ref_month IS REF CURSOR RETURN t_month%ROWTYPE;
  v_month_char VARCHAR(15);
  v_month_num  INTEGER;

  PROCEDURE p_open( p_refcur_month OUT ref_month, p_month IN INTEGER );
  PROCEDURE p_fetch( p_refcur_month IN ref_month );
  PROCEDURE p_close( p_refcur_month IN ref_month );
END;
/
Package created.
gSQL>
CREATE OR REPLACE PACKAGE BODY pkg_refcur AS
  PROCEDURE p_open( p_refcur_month OUT ref_month, p_month IN INTEGER ) AS
  BEGIN
    IF p_month = 1 THEN
      OPEN p_refcur_month FOR SELECT * FROM t_month WHERE month_num = 1;
    ELSIF p_month = 3 THEN
      OPEN p_refcur_month FOR SELECT * FROM t_month WHERE month_num = 3;
    ELSIF p_month = 5 THEN
      OPEN p_refcur_month FOR SELECT * FROM t_month WHERE month_num = 5;
    ELSIF p_month = 7 THEN
      OPEN p_refcur_month FOR SELECT * FROM t_month WHERE month_num = 7;
    ELSIF p_month = 9 THEN
      OPEN p_refcur_month FOR SELECT * FROM t_month WHERE month_num = 9;
    ELSIF p_month = 11 THEN
```

```
      OPEN p_refcur_month FOR SELECT * FROM t_month WHERE month_num = 11;
    ELSE
      DBMS_OUTPUT.PUT_LINE( 'NO ODD MONTH' );
    END IF;

    DBMS_OUTPUT.PUT_LINE( 'p_refcur_month%ISOPEN : ' || p_refcur_month%ISOPEN );
  END;

  PROCEDURE p_fetch( p_refcur_month IN ref_month ) AS
  BEGIN
    FETCH p_refcur_month INTO v_month_char, v_month_num;
    DBMS_OUTPUT.PUT_LINE( 'v_month_char : ' || v_month_char || ' , v_month_num : ' ||
v_month_num );
  END;

  PROCEDURE p_close( p_refcur_month IN ref_month ) AS
  BEGIN
    CLOSE p_refcur_month;
    DBMS_OUTPUT.PUT_LINE( 'p_refcur_month%ISOPEN : ' || p_refcur_month%ISOPEN );
  END;
END;
/
Package created.
gSQL>
DECLARE
  v_pkg_refcur_month pkg_refcur.ref_month;
BEGIN
  pkg_refcur.p_open( v_pkg_refcur_month, 5 );
  pkg_refcur.p_fetch( v_pkg_refcur_month );
  pkg_refcur.p_close( v_pkg_refcur_month );
END;
/
p_refcur_month%ISOPEN : TRUE
v_month_char : MAY , v_month_num : 5
p_refcur_month%ISOPEN : FALSE
Anonymous PL block executed.
```

Cursor Variable as Host Variable

The cursor variable can be declared as the host variable. The host cursor variable is useful to transfer the query result between the server and the client, and it can share the same query result sets between the se rver and the client.

Declare REF CURSOR type host variable and transfer it to the server to use the cursor variable as the host variable. Calling the cursor variable between the server and the client is not limited. Declare the cursor var iable in the client and fetch it after OPENing in the server, then it can be closed after it is FETCHed in the client.

Example of Host Cursor Variable

```
gSQL>
\var host_refcur_month REFCURSOR
gSQL>
BEGIN
  OPEN :host_refcur_month FOR SELECT * FROM t_month;
END;
/
Anonymous PL block executed.
gSQL>
DECLARE
  v_month_char VARCHAR(15);
  v_month_num  INTEGER;
BEGIN
  FETCH :host_refcur_month INTO v_month_char, v_month_num;
  DBMS_OUTPUT.PUT_LINE( 'v_month_char : ' || v_month_char || ' , v_month_num : ' ||
v_month_num );
END;
/
v_month_char : JANUARY , v_month_num : 1
Anonymous PL block executed.
gSQL>
BEGIN
  CLOSE :host_refcur_month;
END;
/
Anonymous PL block executed.
```

25.

Using PSM Subprograms

25.1 Anonymous PL Block

An anonymous PL block is an SQL statement for a one - time execution of a PSM statement instead of st oring a PSM statement in the database. An anonymous PL block is a one of a regular SQL provided by GO LDILOCKS, so it is used in the same way as other SQL s in ODBC, JDBC and precompiler which are provide d by GOLDILOCKS. A syntax is as same as a syntax of a general basic block
For more information, refer to **Block (BEGIN .. END)**.

```
<<Label>> ❶ (optional)
DECLARE   ❷ (optional)
❸ declare items (variables, cursors, types, ...) (optional)
BEGIN     ❹ (required)
❺ PSM statements to execute (required)
EXCEPTION ❻ Exception Handling Part (optional)
END;
```

Refer to the following instructions when using an anonymous PL block.

Interface	Instructions for use
in common	A bind parameter can be used unlike a schema-level procedure/ function. It also supports a prepare-execute method.
ODBC	It is used completely same as a general SQL.
JDBC	A CallableStatement class should be used if it has a bind parameter with IN-OUT or OUT attribute. (Refer to **CallableStatement**.)
Precompiler (gpec)	N/A
Interactive command tool (gsql)	It should notify the end of the statement by entering '/'<Enter> after inputting a n anonymous PL block.

25.2 Nested Procedure

A nested procedure is a procedure type subprogram declared within a specific PL block. A nested proced ure has a scope which can be referenced only on a declared PL block and its subordinates.
For more information, refer to **Procedure Declaration and Definition**.

```
DECLARE
  PROCEDURE PROC1( A1 INTEGER )  ❶ Define Nested Procedure
    IS
    BEGIN
      DBMS_OUTPUT.PUT_LINE( 'A1 = ' || A1 );
    END;
BEGIN
  PROC1( 100 );  ❷ Call Nested Procedure
END;
/
```

The followings are items available within a nested subprogram.

- An argument variable of a nested subprogram
- A PL block in which a nested subprogram is defined, and variables and various items (type, cursor,...) defined in its superordinate scope.
- A bind parameter when it is an anonymous PL block. (e.g. '?', ':V1')

A nested procedure supports a forward declaration, so it can separately specify a declare statement and a define statement. By using this, a logic of cross call between two nested procedures can be implemented.

```
gSQL> DECLARE
  PROCEDURE PROC1( A1 INTEGER );  ❶ declare proc1
  PROCEDURE PROC2( A1 INTEGER )   ❷ define  proc2
  IS
  BEGIN
    IF A1 > 0 THEN
      DBMS_OUTPUT.PUT_LINE( '(proc2)A1 = ' || A1 );
      PROC1( A1 -1 );  ❸ call proc1
    END IF;
  END;
  PROCEDURE PROC1( A1 INTEGER )   ❹ define  proc1
  IS
  BEGIN
    IF A1 > 0 THEN
```

```
      DBMS_OUTPUT.PUT_LINE( '(proc1)A1 = ' || A1 );
      PROC2( A1 -1 );  ❺ call proc2
    END IF;
  END;
BEGIN
  PROC1(5); ❻ call proc1
END;
/

(proc1)A1 = 5
(proc2)A1 = 4
(proc1)A1 = 3
(proc2)A1 = 2
(proc1)A1 = 1
Anonymous PL block executed.
```

GOLDILOCKS PSM limits the maximum subordinate statement depths to 50 to prevent the infinite cross r eferencing. If it is exceeded, then the following error occurs. (It is also applied same to a nested function, schema-level procedure/ function.)

```
gSQL> DECLARE
  PROCEDURE PROC1( A1 INTEGER );  ❶ declare proc1
  PROCEDURE PROC2( A1 INTEGER )   ❷ define  proc2
  IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE( 'A1 = ' || A1 );
    PROC1( A1 -1 );
  END;
  PROCEDURE PROC1( A1 INTEGER )   ❸ define  proc1
  IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE( 'A1 = ' || A1 );
    PROC2( A1 -1 );
  END;
BEGIN
  PROC1(100);
END;
/
ERR-42000(16411): maximum number of recursive SQL levels (50) exceeded.
```

25.3 Nested Function

A nested function is as same as a nested procedure, but it is a subprogram in function form.
For more information, refer to **Function Declaration and Definition**.

```
gSQL> DECLARE
  V1 INTEGER := 0;
  FUNCTION FUNC1( A1 INTEGER )
    RETURN INTEGER
    IS
    BEGIN
      RETURN A1 * 10;
    END;
BEGIN
  V1 := FUNC1( 10 );
  DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
END;
/
V1 = 100
Anonymous PL block executed.
```

A nested function can be used in all PSM expressions within a visible scope, but is can not be used in an SQL statement.

```
gSQL> DECLARE
  V1 INTEGER := 0;
  FUNCTION FUNC1( A1 INTEGER )
    RETURN INTEGER
    IS
    BEGIN
      RETURN A1 * 10;
    END;
BEGIN
  SELECT FUNC1(10) INTO V1 FROM DUAL;
  DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
END;
/
ERR-HY000(17032): PSM compilation error :
(1) at (10:3): ERR-HY000(17079): a nested function not allowed in executing SQL
```

25.4 Schema-level Procedure

A schema-level subprogram is an SQL object which has a name and is stored in a database. A schema-level procedure is a schema-level subprogram in procedure form.

Creating Schema-level Procedure

A schema-level procedure is created as follows. A precision and a scale value should be specified if in need.
For more information, refer to **CREATE PROCEDURE**.

```
CREATE OR REPLACE PROCEDURE PROC1( A1 INTEGER, A2 INTEGER )
IS
  V1 INTEGER;
BEGIN
  SELECT COUNT(*)
    INTO V1
    FROM T1
    WHERE T1.I1 >= A1 AND T1.I1 <= A2;
  DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
END;
/
```

The information about a created schema-level procedure can be found with a INFORMATION_SCHEMA.ROUTINES table.
For more information, refer to **ROUTINES**.

```
gSQL> SELECT SPECIFIC_NAME, ROUTINE_DEFINITION
  FROM  INFORMATION_SCHEMA.ROUTINES
  WHERE SPECIFIC_NAME = 'PROC1';
SPECIFIC_NAME ROUTINE_DEFINITION
------------- ------------------------------------------------------
PROC1         PROCEDURE "PUBLIC"."PROC1" ( A1 INTEGER, A2 INTEGER )
              IS
                V1 INTEGER;
              BEGIN
                SELECT COUNT(*)
                  INTO V1
```

```
                FROM T1
                WHERE T1.I1 >= A1 AND T1.I1 <= A2;
              DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
            END;

1 row selected.
```

The information about an argument can be found with a INFORMATION_SCHEMA.PARAMETERS table. For more information, refer to **PARAMETERS**.

```
gSQL> SELECT P.PARAMETER_NAME, P.ORDINAL_POSITION
  FROM  INFORMATION_SCHEMA.ROUTINES     R,
        INFORMATION_SCHEMA.PARAMETERS   P
  WHERE R.SPECIFIC_NAME = 'PROC1'
    AND R.SPECIFIC_SCHEMA = P.SPECIFIC_SCHEMA
    AND R.SPECIFIC_NAME = P.SPECIFIC_NAME
  ORDER BY P.ORDINAL_POSITION;
PARAMETER_NAME ORDINAL_POSITION
-------------- ----------------
A1                            1
A2                            2
2 rows selected.
```

Using Schema-level Procedure

A schema-level procedure is used by an inner statement of other PSM or by a CALL statement. Other PSM statements call a schema-level procedure in the same form of a nested procedure.

```
gSQL> BEGIN
  PROC1( 2, 4 );  ❶ Call schema-level procedure
END;
/
V1 = 3
Anonymous PL block executed.
```

A CALL statement which is an SQL executing the given PSM is executed as follows.
For more information, refer to **CALL Statement**.

```
gSQL> CALL PROC1( 2, 4 );
V1 = 3
Procedure Call complete.
```

It supports the following syntax for a procedure to support a procedure call escape sequence used in ODB C or JDBC.

```
{ CALL procedure_name( param1, param2, ... ) }
```

A syntax of a procedure call escape sequence is used like as a general SQL.

```
gSQL> { CALL PROC1(2, 4) };
V1 = 3
Procedure Call complete.
```

gsql which is an interactive command tool of GOLDILOCKS does not support performing a procedure thr ough \EXEC which is the tool's own command.

The following options about a privilege of when executing a schema-level procedure can be specified. If a user except for a definer performs an SQL within PSM according to these options, it may refer to a table with the same name in different schemas according to the definition of that user's schema-path. (A nam e of the object used when declaring an item (e.g. T1%ROWTYPE) is always interpreted as a definer.)

- AUTHID DEFINER (default): The user is altered to the user created that procedure, then it is performe d.
- AUTHID CURRENT_USER: The user is not altered, and it is performed by the current user.

```
CREATE OR REPLACE PROCEDURE "PROC1"( A1 INTEGER, A2 INTEGER )
AUTHID CURRENT_USER
IS
  V1 INTEGER;
BEGIN
  SELECT COUNT(*)
    INTO V1
    FROM T1
    WHERE T1.I1 >= A1 AND T1.I1 <= A2;
    DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
END;
/
```

Dropping Schema-level Procedure

A schema-level procedure is dropped by the following DROP PROCEDURE statement.
For more information, refer to **DROP PROCEDURE**.

```
DROP PROCEDURE PROC1;
```

It also supports IF EXISTS statement like as other DROP statements of GOLDILOCKS.

```
DROP PROCEDURE IF EXISTS PROC1;
```

Recompiling Schema-level Procedure

If an object referenced inside of a schema-level subprogram is altered, then it may affects the correspondi ng subprogram and should create the execution plan again.

Referenced Object in a Declarative Part or an Argument

If the used object is altered when defining various items in a declarative part or an argument type, then t he procedure plan is automatically recompiled.

- An object used in %TYPE, %ROWTYPE
- An object used in an explicit cursor definition statement

If the corresponding procedure is performed for the first time after an object is altered, then the plan is a utomatically recreated according to the following order.

1. Bring the plan of the corresponding procedure from a plan cache.
2. Search for objects which were altered during validating an object list of that plan.
3. Discard the current plan, and create a new plan from a procedure definition statement stored in a di ctionary.
4. Register a newly created plan on a plan cache.
5. Execute the plan.

Referenced Object in an SQL of the Body

An SQL plan used in the body is not stored in a procedure plan, and only the SQL text is stored. When ex ecuting a procedure, it creates a plan by compiling the corresponding SQL statement in real time, then ex ecutes it. Therefore, an SQL object used within a body does not affect the procedure plan itself.

However, if an object is altered, the number or type of binds which are the existing interfaces of a proced ure may be altered, or columns may be dropped. Therefore, an error may occur when executing it if the p rocedure is not appropriately altered.

25.5 Schema-level Function

A schema-level function is a schema-level subprogram in function form, which is used within an expressio n.

Creating Schema-level Function

A schema-level function is created as follows. A precision and a scale value of an argument type should b e specified if in need.
For more information, refer to **CREATE FUNCTION**.

```
gSQL> CREATE OR REPLACE FUNCTION FUNC1( A1 INTEGER, A2 INTEGER )
RETURN INTEGER
IS
  V1 INTEGER;
BEGIN
  SELECT COUNT(*)
    INTO V1
    FROM T1
    WHERE T1.I1 >= A1 AND T1.I1 <= A2;
  RETURN V1;
END;
/
Function created.
```

Like as a procedure, a created schema-level function is found throughINFORMATION_SCHEMA.ROUTINE S and INFORMATION_SCHEMA.PARAMETERS table.

Using Schema-level Function

A schema-level function can be used in all expressions in which a general SQL or an SQL object within PS M can be used.

```
gSQL> SELECT FUNC1( 2, 4 ) FROM DUAL;
FUNC1( 2, 4 )
-------------
            3
```

```
1 row selected.
```

It is used as follows within PSM.

```
gSQL> DECLARE
  V1 INTEGER;
BEGIN
  V1 := FUNC1( 2, 4 );
  DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
END;
/
V1 = 3
Anonymous PL block executed.
```

It can be performed by using CALL statement as follows.
For more information, refer to **CALL Statement**.

```
gSQL> \var V1 INTEGER
gSQL> CALL FUNC1( 2, 4 ) INTO :V1;
Procedure Call complete.
gSQL> \print V1
V1
--
 3
```

Like as a schema-level procedure, a procedure call escape sequence statement is also supported.

```
{ ? = CALL function_name( param1, param2, ... ) }
```

```
gSQL> { :V1 = CALL FUNC1( 2, 4 ) };
Procedure Call complete.
gSQL> \print V1
V1
--
 3
```

Dropping Schema-level Function

A schema-level function is dropped by the following DROP FUNCTION statement.
For more information, refer to **DROP FUNCTION**.

```
gSQL> DROP FUNCTION FUNC1;
Function dropped.
```

25.6 Built-in Procedures

GOLDILOCKS PSM provides the following built-in procedures to debug or process the exception when m aterializing the procedure and the function.

Table 25-1 Built-in procedures

Procedure	Description
DBMS_OUTPUT.ENABLE(buffer_si ze IN NATIVE_INTEGER := 20000)	It activates a message logging feature to the given buffer size. If the buffer size is not given it is set to 20000 bytes by default. If the feature i s already activatedm then it discards all messages and creates a new buffer.
DBMS_OUTPUT.DISABLE	It deactivates a message logging feature. All messages previously logged are discarded.
DBMS_OUTPUT.SET_LOG(file_pat h IN VARCHAR(4000))	It simultaneously outputs on the file in a given path when logging a message. If the given path is a relative path (its first letter is not /), then it searches for a target file below a directory corresponding to <SYSTEM_LOGGER_DIR> prope rty. Specify the permission as follows. • CALL DBMS_OUTPUT.SET_LOG('a.txt' , 640)
DBMS_OUTPUT.PUT_LINE(item IN VARCHAR(4000))	It stores a message created with a given expression including the new line ch aracter on a buffer.
DBMS_OUTPUT.PUT(item IN VARC HAR(4000))	It stores a message created with a given expression on a buffer.
DBMS_OUTPUT.NEW_LINE	It stores the new line character on a buffer.
DBMS_OUTPUT.GET_LINE(line OU T VARCHAR(4000), status OUT NA TIVE_INTEGER)	It returns the oldest single message line which is not read among messages st ored on a buffer. If there is a message, a status returns 0, if there is not any m essage it returns 1.
DBMS_STANDARD.RAISE_APPLICA TION_ERROR(error_code IN NATIV E_INTEGER, error_message IN VAR CHAR(4000), stack_flag IN BOOLE AN := FALSE)	It generates an arbitrary user exception. and error_code should be the value b etween -20000 and -20999. If it is TRUE, the last stack_flag argument stacks a given error on the existing errors. If it is FALSE, the corresponding error repl aces all errors. (It can be omitted, and the default is FALSE when it is omitted.)
DBMS_LOCK.SLEEP(seconds IN NU MBER)	It pauses the session for the specified time.
DBMS_SQL.RETURN_RESULT(rc IN SYS_REFCURSOR)	It returns the result of the query executed through *ref cursor* to the client app lication.

```
gSQL> DECLARE
V1 VARCHAR(1024);
V2 INTEGER;
BEGIN
```

```
  DBMS_OUTPUT.ENABLE(2000);
  DBMS_OUTPUT.PUT_LINE('TEST MSG');
  DBMS_OUTPUT.GET_LINE( V1, V2);
  DBMS_OUTPUT.PUT_LINE('V1 = ' || v1);
  DBMS_OUTPUT.PUT_LINE('V2 = ' || v2);
END;
/
V1 = TEST MSG
V2 = 0
Anonymous PL block executed.
```

A message logging feature is managed per each section, and gsql (an interactive command tool of GOLD ILOCKS) can turn on or off the message logging feature with a serveroutput option. If any content remai ns in a message buffer after performing PSM, all are automatically output.

```
gSQL> \set serveroutput on
gSQL> \var msg VARCHAR(4000)
gSQL> \var status NATIVE_INTEGER
gSQL> CALL DBMS_OUTPUT.PUT_LINE( 'aaa' );
aaa
Procedure Call complete.
gSQL> CALL DBMS_OUTPUT.PUT_LINE( 'bbb' );
bbb
Procedure Call complete.
gSQL> CALL DBMS_OUTPUT.GET_LINE( :msg, :status );
Procedure Call complete.
gSQL> \print msg
MSG
----
null
gSQL> \print status
STATUS
------
     1
```

26.

Using SQLs in PSM

26.1 Static SQLs

Overview

A static SQL is an extended SQL to use PSM variable supported by GOLDILOCKS. PSM variable is available in every expression which allows the bind parameter in SQL.

- SELECT INTO Statement
  - For more information, refer to **SELECT INTO Statement** in PSM Language Element References.
- Data Manipulation Language(DML)
  - INSERT Statement Extension
    - For more information, refer to **INSERT Statement Extension** in PSM Language Element References.
  - INSERT INTO ... UPDATE Statement Extension
    - For more information, refer to **INSERT INTO ... UPDATE Statement Extension** in PSM Language Element References.
  - UPDATE Statement Extension
    - For more information, refer to **UPDATE Statement Extension** in PSM Language Element References.
  - DELETE Statement Extension
    - For more information, refer to **DELETE Statement Extension** in PSM Language Element References.
- Transaction Control Language
  - COMMIT
    - For more information, refer to **COMMIT** in SQL References.
  - ROLLBACK
    - For more information, refer to **ROLLBACK** in SQL References.
  - SAVEPOINT
    - For more information, refer to **SAVEPOINT savepoint_specifier** in SQL References.
  - LOCK TABLE
    - For more information, refer to **LOCK TABLE** in SQL References.

Examples

- When using the built-in data type PSM variable

```
gSQL>
DECLARE
  v_id      NUMBER;
```

```
  v_name    VARCHAR(50);
BEGIN
  -- SELELECT INTO Statement
  SELECT id , name
    INTO v_id , v_name
    FROM emp
   WHERE id = 201;
   DBMS_OUTPUT.PUT_LINE( '[SELECT INTO] ' || v_id || ' , ' || v_name );
  -- INSERT Statement Extension
  v_id   := 200;
  v_name := 'Jennifer Whalen';
  INSERT INTO emp VALUES( v_id , v_name );

  -- DELETE Statement Extension
  DELETE FROM emp
   WHERE id = v_id
  RETURNING name INTO v_name;
  DBMS_OUTPUT.PUT_LINE( '[DELETE] ' || v_id || ' , ' || v_name );

  -- UPDATE Statement Extension
  UPDATE emp
     SET id = v_id , name = v_name
   WHERE id = 200;
  -- Commit
  COMMIT;
END;
/
[SELECT INTO] 201 , Michael Hartstein
[DELETE] 200 , Jennifer Whalen
Anonymous PL block executed.
```

- When using the ROW type PSM variable

```
gSQL>
DECLARE
  TYPE rec_emp IS RECORD( f_id emp.id%TYPE , f_name emp.name%TYPE );
  v_rec_emp rec_emp;
BEGIN
  -- SELELECT INTO Statement
  SELECT id , name
    INTO v_rec_emp
```

```
    FROM emp
   WHERE id = 201;
   DBMS_OUTPUT.PUT_LINE( '[SELECT INTO] ' || v_rec_emp.f_id ||
                         ' , ' || v_rec_emp.f_name );
  -- INSERT Statement Extension
  v_rec_emp.f_id   := 200;
  v_rec_emp.f_name := 'Jennifer Whalen';
  INSERT INTO emp VALUES( v_rec_emp.f_id , v_rec_emp.f_name );

  -- DELETE Statement Extension
  DELETE FROM emp
   WHERE id = v_rec_emp.f_id
  RETURNING * INTO v_rec_emp;
   DBMS_OUTPUT.PUT_LINE( '[DELETE] ' || v_rec_emp.f_id ||
                         ' , ' || v_rec_emp.f_name );

  -- UPDATE Statement Extension
  UPDATE emp
     SET ROW = v_rec_emp
   WHERE id = 200;
  -- Commit
  COMMIT;
END;
/
[SELECT INTO] 201 , Michael Hartstein
[DELETE] 200 , Jennifer Whalen
Anonymous PL block executed.
```

Processing Query Result Sets

PSM processes result sets by using an implicit cursor or an explicit cursor.

PSM defines the implicit cursor as follows.

- SELECT INTO
- Implicit Cursor FOR LOOP

PSM defines the explicit cursor as follows.

- Explicit Cursor FOR LOOP

- It is an explicit cursor defined by a user, and it is available during executing PSM statement.

Processing Query Result Sets with SELECT INTO Statements

It retrieves the value by executing SELECT INTO statement using an implicit cursor, then stores it in PSM v ariable.
The result set of select into statement is always a single row.

Examples

```
gSQL>
DECLARE
  v_id   emp.id%TYPE;
  v_name emp.name%TYPE;
BEGIN
  SELECT id , name
    INTO v_id , v_name
    FROM emp
   WHERE id = 201;

   DBMS_OUTPUT.PUT_LINE( 'SQL%FOUND = ' || SQL%FOUND );
END;
/
SQL%FOUND = TRUE
Anonymous PL block executed.
```

Processing Query Result Sets with Cursor FOR LOOP Statements

Cursor For LOOP statement executes an implicit cursor and an explicit cursor, then repeatedly returns the row in the result set.

Implicit cursor FOR LOOP statement is cursor FOR LOOP statement which uses SELECT statement. Implicit cursor FOR LOOP statement returns a row in the result set by using an implicit cursor for the select statem ent.

An explicit cursor declared by a user is available in cursor FOR LOOP statement
An explicit cursor declared by a user is also available in another statement in PSM block.

Cursor FOR LOOP statement implicitly creates and uses %ROWTYPE variable for the type returned as a lo op index by a cursor.
A loop index is a variable which is available only during executing cursor FOR LOOP statement.

PSM statement which is operated during the loop can refer to the record and the field by using a loop ind ex.

Cursor FOR LOOP statement is executed by opening the user defined cursor after creating the loop index variable.
It stores the row result in the loop index variable whenever repeating the loop.
If the row is not returned anymore, then the cursor is closed. The cursor is closed even when it throws an exception during the execution.

Examples

- When using SELECT statement in cursor FOR LOOP statement

```
gSQL>
BEGIN
  FOR tmp IN ( SELECT id , name , manager_id FROM emp ) LOOP
    DBMS_OUTPUT.PUT_LINE( 'id = ' || tmp.id ||
                          ' , name = ' || tmp.name ||
                          ' , manager_id = ' || tmp.manager_id );
  END LOOP;
END;
/
id = 200 , name = Jennifer Whalen   , manager_id = 101
id = 201 , name = Michael Hartstein , manager_id = 101
id = 202 , name = Pat Fay           , manager_id = 301
id = 203 , name = Susan Mavris      , manager_id = 201
id = 204 , name = Hermann Baer      , manager_id = 201
id = 205 , name = Shelley Higgins   , manager_id = 301
id = 206 , name = William Gietz     , manager_id = 201
Anonymous PL block executed.
```

- When using an explicit cursor in cursor FOR LOOP statement
  - If the explicit cursor does not have a parameter

```
gSQL>
DECLARE
  CURSOR cur1 IS SELECT id , name , manager_id FROM emp;
BEGIN
  FOR tmp IN cur1 LOOP
    DBMS_OUTPUT.PUT_LINE( 'id = ' || tmp.id ||
                          ' , name = ' || tmp.name ||
                          ' , manager_id = ' || tmp.manager_id );
```

```
  END LOOP;
END;
/
id = 200 , name = Jennifer Whalen   , manager_id = 101
id = 201 , name = Michael Hartstein , manager_id = 101
id = 202 , name = Pat Fay           , manager_id = 301
id = 203 , name = Susan Mavris      , manager_id = 201
id = 204 , name = Hermann Baer      , manager_id = 201
id = 205 , name = Shelley Higgins   , manager_id = 301
id = 206 , name = William Gietz     , manager_id = 201
Anonymous PL block executed.
```

- If the explicit cursor has a parameter

```
gSQL>
DECLARE
  CURSOR cur1( p1 NUMBER ) IS SELECT id , name , manager_id
                               FROM emp
                              WHERE manager_id = p1;
BEGIN
  FOR tmp IN cur1( 201 ) LOOP
    DBMS_OUTPUT.PUT_LINE( 'id = ' || tmp.idgSQL>  ||
                          ' , name = ' || tmp.name ||
                          ' , manager_id = ' || tmp.manager_id );
  END LOOP;
END;
/
id = 203 , name = Susan Mavris      , manager_id = 201
id = 204 , name = Hermann Baer      , manager_id = 201
id = 206 , name = William Gietz     , manager_id = 201
Anonymous PL block executed.
```

Processing Query Result Sets with Explicit Cursors, OPEN, FETCH, and CLOSE

Declare and use an explicit cursor to control the result set as desired.
The user can manage the result set by using OPEN, FETCH, CLOSE statement after declaring the explicit c
ursor.

The query using PL statement may look complicated, but it can flexibly manage the result set as follows.

- It can process result sets in parallel by using multiple explicit cursors.
- It can process multiple rows in the result set in parallel or skip a specific row in a single loop state ment.
- Also, it can use multiple loop statements to process the result sets by dividing them.

For more information, refer to **Explicit Cursor**.

26.2 Dynamic SQL

Unlike a static SQL, the syntax of a dynamic SQL is determined at the time of execution.

In PSM, a dynamic SQL created by a user at run-time can be performed through EXECUTE IMMEDIATE or OPEN FOR statement.

- A dynamic SQL is used in the following cases.
  - When an SQL statement can not be determined when compiling. (e.g. When an SQL statement s hould be altered according to conditions.)
  - When an SQL which is not supported in a static SQL is performed. (e.g. DDL)
- The syntax of a dynamic SQL is unknown until the time of execution, so a run-time error may occur d ue to a syntax error, existence of a target object and validation of a user privilege.

- A dynamic SQL can be used in the following PSM statements.
  - **EXECUTE IMMEDIATE Statement**
  - **OPEN FOR Statement**

EXECUTE IMMEDIATE

Various dynamic SQLs can be performed in EXECUTE IMMEDIATE. However, a statement in a form of an SQL extension provided by PSM can not be used.

The statement is provided in the following form.

```
EXECUTE IMMEDIATE 'dynamic sql' [ USING [IN ¦ OUT ¦ INOUT] variable_list] [INTO variable_list]
[RETURNING INTO variable_list]
```

A value stored in a PSM variable can be applied to a database through USING, INTO, RETURNING INTO st atements which are provided to an EXECUTE IMMEDIATE statement, or a value can be stored from a dat abase to a PSM variable. The differences of using each statement are as follows.

- USING
  - It uses an IN-mode when applying a value of PSM variable to an SQL.
  - It uses an OUT-mode when storing the result of SQL processing in a PSM variable.
    - Therefore, if uses an OUT-mode, it should be described with PSM variables. (An expression is not available.)
  - An IN-mode is applied when a binding type of the variable specified in USING clause is not specifi ed.
  - It is operated as same as using SELECT INTO clause when a target of SELECT is returned as USING OUT.

    - Only a single row can be returned from the database. If two or more results occurs then an error occurs.
    - A PSM variable in a USING IN clause can use only a form of scalar type.
    - A PSM variable in a USING OUT clause can use a record type, but it can not use it by mixing with another type.
- INTO
    - It is used when storing the result of SELECT which is internally processed in a database using an implicit cursor.
    - It can be used only when executing SELECT clause using a dynamic SQL method. (SELECT INTO clause can not be used.)
    - A PSM variable can be a record-type, but it can not be used by mixing with another type.
- RETURING INTO
    - It is used when storing the before/ after of the data processed by INSERT/ UPDATE/ DELETE.
    - It can store only a single row.
    - A PSM variable can be a record-type, but it can not be used by mixing with another type.

The following is an example of outputting the SELECT_INTO statement result by using a dynamic SQL.

```
gSQL> CREATE TABLE T1
(
  C1 VARCHAR(20),
  C2 VARCHAR(20)
);
Table created.
gSQL> INSERT INTO T1 VALUES ('Seoul', '24');
1 row created.
gSQL> INSERT INTO T1 VALUES ('Pusan', '44');
1 row created.
gSQL> DECLARE
  V1 VARCHAR(20);
  V2 VARCHAR(20);
BEGIN
  EXECUTE IMMEDIATE 'SELECT * INTO ?, ? FROM T1 WHERE C1 = ''Seoul'''
  USING OUT V1, OUT V2;
  DBMS_OUTPUT.PUT_LINE('V1 = ' || V1);
  DBMS_OUTPUT.PUT_LINE('V2 = ' || V2);
END;
/
V1 = Seoul
V2 = 24
Anonymous PL block executed.
```

The following is an example of performing an altering operation and storing the result of when before th e alteration in a variable specified in RETURNING INTO.

```
gSQL> CREATE TABLE T1
(
  C1 VARCHAR(20),
  C2 VARCHAR(20)
);
Table created.
gSQL> INSERT INTO T1 VALUES ('Seoul', '24');
1 row created.
gSQL> INSERT INTO T1 VALUES ('Pusan', '44');
1 row created.
gSQL> DECLARE
  V1 VARCHAR(20);
  V2 VARCHAR(20);
  V3 VARCHAR(20);
  V4 VARCHAR(20);
BEGIN
  V1 := 'Daegu';
  V2 := '50';
  EXECUTE IMMEDIATE
      'UPDATE T1 SET C1 = ? ,  C2 = ? WHERE C1 = ''Seoul'' RETURNING OLD * INTO ?, ?'
  USING V1, V2 RETURNING INTO V3, V4;
  DBMS_OUTPUT.PUT_LINE('V3 = ' || V3);
  DBMS_OUTPUT.PUT_LINE('V4 = ' || V4);
END;
/
V3 = Seoul
V4 = 24
Anonymous PL block executed.
```

For more information. refer to **EXECUTE IMMEDIATE Statement**.

OPEN FOR, FETCH and CLOSE

When processing a query by using EXECUTE IMMEDIATE, the database can not return one or more result s. If an SQL to be processed is a dynamic SQL, and two or more result sets should be fetched, like as a cur sor, then use OPEN FOR.

The a dynamic SQL in the following form can be used in OPEN FOR statement.

```
OPEN Cursor_variable FOR dynamic_sql [USING variable_list]
```

The following is an example of performing OPEN FOR statement through a dynamic SQL.

```
gSQL> CREATE TABLE T1
(
  C1 VARCHAR(20),
  C2 VARCHAR(20)
);
Table created.
gSQL> INSERT INTO T1 VALUES ('Seoul', '24');
1 row created.
gSQL> INSERT INTO T1 VALUES ('Pusan', '44');
1 row created.
gSQL> DECLARE
  v1 VARCHAR(20);
  v2 VARCHAR(20);
  v3 VARCHAR(20);
  cv1 SYS_REFCURSOR;
  sqlstr VARCHAR(1024);
BEGIN

    sqlstr := 'SELECT * FROM T1 WHERE C1 >= ?';
    v3 := 'AAAA';
    OPEN cv1 FOR sqlstr USING v3;
    FETCH cv1 INTO v1, v2;
    DBMS_OUTPUT.PUT_LINE('V1 = ' || V1 || ' , V2 = ' || V2);
    CLOSE cv1;
END;
/
V1 = Seoul , V2 = 24
Anonymous PL block executed.
```

- USING clause in OPEN FOR statement has the following constraints.
  - Binding mode
    - Only IN mode is available.
  - Available items
    - PSM variable
    - Value expression whose result is returned as a scalar type

For more information, refer to **OPEN FOR Statement**, **FETCH Statement**, **CLOSE Statement**.

27.

PSM Packages

This chapter describes how to modularize common data and PSM codes used in various applications into the package.

27.1 Definition

A package is a schema object which bundles items such as logically-related PSM types, variables, subprograms, cursors and exceptions. A package is stored in the database through the compiling process, so that another program (another package, procedure, external program) may refer, share and execute the items in the package.

Every package has a specification in which referable multiple public items and subprograms are listed.
If a cursor exists among public items or a public subprogram exists, then the package may have a package body. The body includes the following contents.

- SQL statement to be performed by a public cursor, or PSM code which is to be executed by a public subprogram
- Items such as multiple variables and types subprograms which are private and used only internally
- Code for initializing public items (initialization part) which is performed only once when the package is used for the first time (instantiation).

AUTHID statement of the package specification determines whether to perform the package with an invoker privilege or with a definer privilege (Default: definer). It also determines whether to translate the unqualified object in the invoker or in the definer when referring to it.

Only the package owner and the user with EXECUTE PACKAGE privilege can retrieve the definition statement of package specification. However, only the package owner can retrieve the definition statement of package body.

27.2 Features

The following features of package provides an application developer and an operator with high reliability and reusability.

- Modularity
  - It bundles related items and subprograms together, so it is easy to manage the database and understand its structure.

- Easy programming
  - A user can program the related application only by defining a specification even without defining the body.
- Encapsulating
  - It can hide the body contents which is an internal implementation of the package from the packa ge user.
  - It can alter the body (implementation) without recompiling SQL and application which use the pa ckage.
- Performance optimization
  - The entire package is loaded on the memory at once, so that it can quickly call the subprogram in s the memory.
- Easy privilege control
  - It can bundle all separate privileges for using all public items and subprograms into a single privile ge for the entire package.

A package does not support a subprogram overloading, so functions of the same names can not be defin ed in the same block scope.

27.3 Specification

The following public items can be declared in the package specification.

- User-defined type
  - Record or collection (Associative array)
  - A user-defined type declared in a package can be used only in a PSM-family statement.
- Variable
  - It is recommended to define a get/ set subprogram for each variable not to directly refer to the va riable.
- Subprogram
  - Procedure
  - Function
- Explicit cursor
  - It is allowed to define only a name and a returned type in a package specification and define SQL statement performed by a cursor in a body.
  - A cursor variable can not be defined with a public item in a package.
- User-defined exception

Creating Package Specification

Create a package specification by using CREATE PACKAGE statement.

```
CREATE OR REPLACE PACKAGE MY_PKG
IS
  TYPE MY_REC IS RECORD (F1 INTEGER, F2 INTEGER );
END;
/
Package created.
```

Public items in a defined package can be referred from outside of a package by specifying the package na me together as follows.

```
DECLARE
V1 MY_PKG.MY_REC;
BEGIN
    V1.F1 := 10;
    V1.F2 := 20;
    DBMS_OUTPUT.PUT_LINE( 'V1.F1 = ' || V1.F1 || ', V1.F2 = ' || V1.F2 );
END;
/
V1.F1 = 10, V1.F2 = 20
Anonymous PL block executed.
```

Creating Package Body

The body should be declared when a public cursor or subprogram exists in a package, or the initialization code is required. Otherwise, body definition is optional.

The package body has the following constraints.

- The package body should be created in the schema which is as same as that of the package specificat ion.
- The package body should have the name as same as that of the package specification.
- If a cursor in which the SQL statement is not defined exists among open cursors declared in a packag e specification, then the cursor with the same name, record type and argument should be declared b y specifying SQL in the body.
- A subprogram whose name, argument and return type are same as those of an open subprogram de clared in a package specification should be defined in the body.

- The variable, type, exception which are as same as those defined in a package specification can not b e declared in the package body.

Create a package body as follows.

```
CREATE TABLE emp( empno NUMBER, sal NUMBER, comm NUMBER );
Table created.
INSERT INTO emp VALUES( 3548, 6000, 1000 );
1 row created.
INSERT INTO emp VALUES( 9369, 5000, NULL );
1 row created.
INSERT INTO emp VALUES( 7294, 4000, 500 );
1 row created.
COMMIT;
Commit complete.
CREATE OR REPLACE PACKAGE emp_mgmt
IS
  PROCEDURE adjust_sal(v_flag VARCHAR, v_empno NUMBER, v_pct NUMBER);
  FUNCTION get_annual_sal(v_empno NUMBER) RETURN NUMBER;
END;
/
Package created.
CREATE OR REPLACE PACKAGE BODY emp_mgmt
IS
  PROCEDURE adjust_sal(v_flag VARCHAR, v_empno NUMBER, v_pct NUMBER) IS
  BEGIN
    IF v_flag = 'INCREASE' THEN
      UPDATE emp SET sal = sal + (sal * (v_pct / 100)) WHERE empno = v_empno;
    ELSE
      UPDATE emp SET sal = sal - (sal * (v_pct / 100)) WHERE empno = v_empno;
    END IF;
  END;
  FUNCTION get_annual_sal (v_empno NUMBER) RETURN NUMBER
  IS
    v_sal NUMBER;
  BEGIN
    SELECT (sal + NVL(comm,0)) * 12 INTO v_sal FROM emp WHERE empno = v_empno;
    RETURN v_sal;
  END;
END;
/
Package created.
```

Call subprograms of the created package body as follows.

```
call emp_mgmt.adjust_sal('INCREASE',7369, 10);
Procedure Call complete.
SELECT emp_mgmt.get_annual_sal(7294) FROM DUAL;
EMP_MGMT.GET_ANNUAL_SAL(7294)
------------------------------
                        54000
1 row selected.
```

Package State

Packages are classified into a stateful package and a stateless package.

A stateful package includes one or more variables or cursors, and a stateless package is configured with s ubprograms only such as a procedure or a function.

A stateful package creates a package instance in a session when referring to the variable or the cursor of the package for the first time. In this case, it performs the following operations.

- Initialize a package variable
- Execute an initialize section

Generally, a package state is maintained as same as the lifetime of a session. However, a stateful package is invalidated and an instance is initialized in the following cases.

- When the package is recreated by executing REPLACE PACKAGE statement
- When the package is recompiled by executing ALTER PACKAGE statement
- When altering operation (DDL) which affects the package instance form occurs on any object which was referenced by the package.

A stateless package does not have a state, so it is automatically recompiled even when the related object 's form is altered. However, a stateful package drops the existing package instance, then it recreates a pa ckage instance based on the updated package information.

Therefore, be cautious to handle the situational exception when using a package in an application.

28.

PSM Language Element References

28.1 Assignment Statement

Function

Within a PSM block, it stores a value in a variable or in an out-bind parameter.

Syntax

```
<assignment statement> ::=
    <assignment target> := <value expression>
    ;
<assignment target> ::=
      collection_variable ( index )
    ¦ cursor_variable
    ¦ :host_cursor_variable
    ¦ out_parameter
    ¦ :host_variable [ :indicator_variable ]
    ¦ record_variable . field_name
    ¦ scalar_variable
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the body section of a PL block.

Syntax Rules and Parameters

- collection_variable
  - It is the variable name of COLLECTION type declared in DECLARE section.
- index
  - It is the key value which selects an element among elements of collection_variable.
- cursor_variable
  - It is the cursor name declared in DECLARE section.
- host_variable

    - It is the name of out/ in-out type client-side variable which is transferred from where calling PSM.
- indicator_variable
    - It is the indicator variable name to check the NULL value of host_variable and to find out the actual length of the value.
- record_variable
    - It is the variable name of RECORD type which has already been declared in DECLARE section.
- field_name
    - It is the name of a field among fields included in a RECORD type variable.
- scalar_variable
    - It is the variable name of a general scalar type which has already been declared in DECLARE section.

Description

Targets of assignment statements are classified into an external bind parameter and an internal PSM variable.

- External parameter
    - Host variable
    - Host cursor variable
- Internal variable
    - Out type function argument
    - Internally declared variable
        - Scalar/ record/ multi-set type
        - A specific field of a record type variable
        - A specific element of multi-set type variable

All variables except for a procedure/ function parameter can have a name assigning a scope.

Examples

The following is an example of using an assignment statement.

- Assignment for PSM internal variable

```
gSQL> DECLARE
  V1 INTEGER := 0;
BEGIN
  FOR I IN 1..10 LOOP
```

```
    V1 := V1 + I;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
END;
/
V1 = 55
Anonymous PL block executed.
```

- Assignment for a host variable

```
gSQL> \var P1 INTEGER
gSQL>
BEGIN
  :P1 := 100;
END;
/
gSQL> \print P1
 P1
---
100
Anonymous PL block executed.
```

Compatibility

The differences between an assignment statement of GOLDILOCKS and that of SQL standard are as follo ws.

- An assignment statement of the SQL standard defines a singleton variable assignment and a multiple variable assignment, but GOLDILOCKS supports only a singleton variable assignment.
- An assignment statement of the SQL standard starts with SET keyword, but GOLDILOCKS does not us e SET keyword.
- An assignment statement of the SQL standard uses an equal operator (=) between a target and a val ue, but GOLDILOCKS uses :=.

Table 28-1 SQL standard compatibility

Feature ID	Description	Compatibility
P002	Computational completeness	X
P006	Multiple assignment	X

For More Information

Refer to the followings.

- **Scalar Variable Declaration**
- **Record Variable Declaration**
- **COLLECTION Variable Declaration**
- **Cursor Variable Declaration**

28.2 Basic LOOP Statement

Function

It repeatedly performs statements within LOOP until the LOOP is terminated by performing GOTO or EXIT.

Syntax

```
<basic loop statement> ::=
    LOOP { <SQL procedure statement> ; }... END LOOP [ loop_name ]
    ;
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the body section of a PL block.

Syntax Rules and Parameters

- loop_name
  - It is the label name of <basic loop statement>.
  - Its role is only a comment, so it does not matter even when it is different from the real label nam e of <basic loop statement>.

Description

A basic loop statement is repeatedly performs statements within LOOP.
A basic loop statement is a loop-family statement, so it can be a target statement which GOTO, EXIT, and CONTINUE indicates as a label.

Examples

```
DECLARE
  V1 INTEGER := 1;
BEGIN
  LOOP
    DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
    V1 := V1 + 1;
    EXIT WHEN V1 > 2;
  END LOOP;
END;
/
V1 = 1
V1 = 2
Anonymous PL block executed.
```

Compatibility

〈basic loop statement〉 statement is as same as 〈loop statement〉 of the SQL standard.

Table 28-2 SQL standard compatibility

Feature ID	Description	Compatibility
P002	Computational completeness	O

For More Information

Refer to the followings.

- **CONTINUE Statement**
- **EXIT Statement**
- **GOTO Statement**

28.3 Block (BEGIN .. END)

Function

It creates a new scope and defines a variable, a cursor, a type and an exception.

Syntax

```
<PSM block> ::=
    [ DECLARE <declare item>... ] BEGIN <SQL procedure statement list> END
    ;
<declare item> ::=
    <variable declaration>
    | <explicit cursor declaration>
    | <explicit cursor definition>
    | <cursor variable declaration>
    | <type definition>
    | <exception declaration>
    | <exception init pragma>
    | <procedure declaration>
    | <procedure definition>
    | <function declaration>
    | <function definition>
<executable statement list> ::=
    [ <label list> ] { <SQL procedure statement> ; }...
<label list> ::=
    { << identifier >>  }...
<SQL procedure statement>
      <PSM Static SQL>
    | <PSM Dynamic SQL>
    | <PSM Control Statement>
```

Invocation and Access Rules

It can be used only within PROCEDURE, FUNCTION or an anonymous block.

Syntax Rules and Parameters

- Variable declaration
  - It declares scalar/ record/ array type variables which are to be used in a block.
- Explicit cursor declaration
  - It declares cursors which are to to be used in a block.
- Explicit cursor definition
  - It defines cursors which are to be used in a block.
- Cursor variable declaration
  - It declares a cursor variable to be used in a block.
- Type definition
  - It defines user-defined types which are to be used when declaring variables in a block.
- Exception declaration
  - It defines exceptions to be used in a block.
- Exception init pragma
  - It sets the error code which is to be processed by a user-defined exception.
- Procedure declaration
  - It declares procedures to be used in a block.
- Procedure definition
  - It defines procedures to be used in a block.
- Function declaration
  - It declares functions to be used in a block.
- Function definition
  - It defines functions to be used in a block.
- Static SQL
  - It indicates Data Manipulation Language (DML) and Data Control Language (DCL) which can be performed in PSM.
- Dynamic SQL
  - It indicates statements related to dynamic query processing supported by GOLDILOCKS PSM.
- PSM Control Statement
  - It indicates various flow control statements supported by GOLDILOCKS PSM.

Description

<psm block> is a basic component of PSM.
A block can have a declaration part and a exception handling part.
A block can be duplicated, and the duplicated block has a new subordinate variable scope. A superordinate block can not refer to a variable in a subordinate block.

Examples

```
gSQL>
<<MAIN>>
DECLARE
  V1 INTEGER := 1;
BEGIN
  DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
  <<SUB1>>
  DECLARE
    V1 VARCHAR(10) := 'ABC';
  BEGIN
    DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
    DBMS_OUTPUT.PUT_LINE( 'SUB1.V1 = ' || SUB1.V1 );
    DBMS_OUTPUT.PUT_LINE( 'MAIN.V1 = ' || MAIN.V1 );
 END;
END;
/
V1 = 1
V1 = ABC
SUB1.V1 = ABC
MAIN.V1 = 1
Anonymous PL block executed.
```

Compatibility

<compound statement> of the SQL standard defines ATOMIC /NOT ATOMIC statement which specifies a new savepoint, but GOLDILOCKS does not support it.

Table 28-3 SQL standard compatibility

Feature ID	Description	Remarks
P002	Computational completeness	It does not support ATOMIC statement.

For More Information

Refer to **Overview of PSM**.

28.4 CASE Statement

Function

It performs a statement list satisfying conditions which returns TRUE among given conditions.

Syntax

```
<case statement> ::=
    <simple case statement>
    | <searched case statement>
    ;
<simple case statement> ::=
    CASE <case operand> <simple case statement when clause>...
    [ <case statement else clase> ]
    END CASE
<searched case statement> ::=
    CASE <searched case statement when clause>...
    [ <case statement else clase> ]
    END CASE
<simple case statement when clause> ::=
    WHEN <when operand>
        THEN <executable statement list>
<searched case statement when clause> ::=
    WHEN <search condition>
        THEN <executable statement list>
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the body section of a PL block.

Syntax Rules and Parameters

- Case operand
  - They are all expressions which can be evaluated with a scalar value.
  - However, it can not include a subquery statement.
- When operand
  - It is an expression to compare whether it is as same as <case operand>.
  - However, it can not include a subquery statement.
- Search condition
  - It is a conditional expression to be performed when the evaluation result of <searched case statement> is TRUE.
  - However, it can not include a subquery statement.
- Executable statement list
  - It is a list of all statements which can be performed within PSM.

Description

It performs statements in WHEN clause returning TRUE by evaluating conditions like as IF statement.
It evaluates conditional expressions in order, and stops evaluating after finding the TRUE conditional clause.
If the case satisfying the condition does not exist and ELSE clause is not specified, then an error occurs.

Examples

Using a Simple CASE

```
gSQL> DECLARE
V1 integer := 0;
BEGIN
  SELECT 2 INTO V1 FROM DUAL;
  CASE V1 WHEN 0 THEN DBMS_OUTPUT.PUT_LINE ('Result = 0');
          WHEN 1 THEN DBMS_OUTPUT.PUT_LINE ('Result = 1');
          WHEN 2 THEN DBMS_OUTPUT.PUT_LINE ('Result = 2');
          ELSE DBMS_OUTPUT.PUT_LINE ('Result = OTHER');
  END CASE;
END;
/
```

```
Result = 2
Anonymous PL block executed.
```

Using a Searched CASE

```
gSQL> DECLARE
V1 integer := 0;
BEGIN
  SELECT 2 INTO V1 FROM DUAL;
  CASE WHEN V1 = 0 THEN DBMS_OUTPUT.PUT_LINE ('Result = 0');
       WHEN V1 = 1 THEN DBMS_OUTPUT.PUT_LINE ('Result = 1');
       WHEN V1 = 2 THEN DBMS_OUTPUT.PUT_LINE ('Result = 2');
       ELSE DBMS_OUTPUT.PUT_LINE ('Result = OTHER');
  END CASE;
END;
/
Result = 2
Anonymous PL block executed.
```

Compatibility

CASE statement of the SQL standard defines the comparison of row type (list type) values, but GOLDILOCKS does not support it.
CASE statement of the SQL standard can define multiple conditions in a list in <when operand> by delimiting them with ',', but GODILOCKS does not support it.

Table 28-4 SQL stantard compatibility

Feature ID	Description	Remarks
P002	Computational completeness	It does not support P004, P008.
P004	Extended CASE statement	-
P008	Comma-separated predicates in simple CASE statement	-

28.5 CLOSE Statement

Function

It closes an open cursor.

Syntax

```
<close statement> ::=
    CLOSE cursor_name
    ;
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the body section of a PL block.

Syntax Rules and Parameters

- cursor_name
  - It is the name of a cursor to be closed.

Description

It closes an open cursor.
A closed cursor can be opened again by using an open statement.

Examples

```
gSQL> CREATE TABLE T1 ( I1 INTEGER );
Table created.
gSQL> COMMIT;
Commit complete.
gSQL> DECLARE
  CURSOR C1 IS SELECT I1 FROM T1;
  V1 T1%ROWTYPE;
BEGIN
  OPEN C1;
  FETCH C1 INTO V1;
  CLOSE C1;
END;
/
Anonymous PL block executed.
```

Compatibility

The SQL standard does not define it.

For More Information

Refer to the followings.

- **FETCH Statement**
- **OPEN Statement**

28.6 Collection Method Invocation

Function

It provides a method which can explores a collection type variable.

Syntax

```
<collection method> ::=
         variable_name . <method>
    ;
<method> ::=
        first ()
      ¦ last  ()
      ¦ prior ( expression )
      ¦ next  ( expression )
      ¦ count ()
      ¦ exists ( expression )
      ¦ delete ( expression )
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the body section of a PL block.

Syntax Rules and Parameters

- It can be used only in a variable declared as a collection type.
- FIRST, LAST, COUNT can not have a parameter.
- A parameter should be specified when an object should be specified such as PRIOR, NEXT, EXISTS, D ELETE.
- DELETE is operated as same as PSM statement, and it can not return a different variable as a result. (I t can not be used in an expression.)

Description

Refer to the following table.

Table 28-5 Function

Name	Function	Return value	Whether to require an argument
FIRST	It returns the smallest key.	A key type specified in INDEX OF	X
LAST	It returns the biggest key.	A key type specified in INDEX OF	X
PRIOR	It returns a key smaller than the input key.	A key type specified in INDEX OF	O
NEXT	It returns a key bigger than the input key.	A key type specified in INDEX OF	O
COUNT	It returns the stored count.	INTEGER	X
DELETE	It deletes a value corresponding to a key.	N/A	O
EXISTS	It returns whether a key exists or not.	BOOLEAN	O

Examples

```
DECLARE
TYPE rec IS TABLE OF VARCHAR(20) INDEX BY VARCHAR(20);
v1 rec;
v2 VARCHAR(20);
BEGIN
  DBMS_OUTPUT.PUT_LINE( '-----------------------------------');
  DBMS_OUTPUT.PUT_LINE( 'first = ' || v1.first);
  DBMS_OUTPUT.PUT_LINE( 'last = '  || v1.last);
  DBMS_OUTPUT.PUT_LINE( 'prior = ' || v1.prior('aaa'));
  DBMS_OUTPUT.PUT_LINE( 'next = '  || v1.next('aaa'));
  DBMS_OUTPUT.PUT_LINE( 'count = ' || v1.count());
  FOR I IN 1 .. 9
  LOOP
      v1('a' || to_char(i)) := 'a' || to_char(i);
  END LOOP;
  DBMS_OUTPUT.PUT_LINE( '-----------------------------------');
  DBMS_OUTPUT.PUT_LINE( 'first = ' || v1.first);
  DBMS_OUTPUT.PUT_LINE( 'last = '  || v1.last);
  DBMS_OUTPUT.PUT_LINE( 'count = ' || v1.count());
  DBMS_OUTPUT.PUT_LINE( '-----------------------------------');
```

```
  DBMS_OUTPUT.PUT_LINE( 'Print all from first to last');
  v2 := v1.first;
  WHILE v2 IS NOT NULL
  LOOP
      DBMS_OUTPUT.PUT_LINE('Key= ' || v2 || ',Value=' || v1(v2));
      v2 := v1.next(v2);
  END LOOP;
  DBMS_OUTPUT.PUT_LINE( '-----------------------------------');
  DBMS_OUTPUT.PUT_LINE( 'Print all from last to first');
  v2 := v1.last;
  WHILE v2 IS NOT NULL
  LOOP
      DBMS_OUTPUT.PUT_LINE('Key= ' || v2 || ',Value=' || v1(v2));
      v2 := v1.prior(v2);
  END LOOP;
  v1.delete(v1.first());
  DBMS_OUTPUT.PUT_LINE('count = ' || v1.count() );
  DBMS_OUTPUT.PUT_LINE('first = ' || v1.first() );
END;
/
-----------------------------------
first =
last =
prior =
next =
count = 0
-----------------------------------
first = a1
last = a9
count = 9
-----------------------------------
Print all from first to last
Key= a1,Value=a1
Key= a2,Value=a2
Key= a3,Value=a3
Key= a4,Value=a4
Key= a5,Value=a5
Key= a6,Value=a6
Key= a7,Value=a7
Key= a8,Value=a8
Key= a9,Value=a9
```

```
-----------------------------------
Print all from last to first
Key= a9,Value=a9
Key= a8,Value=a8
Key= a7,Value=a7
Key= a6,Value=a6
Key= a5,Value=a5
Key= a4,Value=a4
Key= a3,Value=a3
Key= a2,Value=a2
Key= a1,Value=a1
count = 8
first = a2
Anonymous PL block executed.
```

For More Information

Refer to **COLLECTION Variable Declaration**.

28.7 COLLECTION Variable Declaration

Function

It declares a collection variable.

Syntax

```
<declare record variable> ::=
    variable_name <collectionType>
    ;

<Collection Type Definition> ::=
    TYPE <Type-Name> IS TABLE OF <Element-Type> INDEX BY <Index-Type>
    ;
<Element-Type> ::=
      Built-in SQL Data Type
    ¦ User-Defined Type
    ¦ %TYPE
    ¦ %ROWTYPE
<Index-Type> ::=
      INTEGER
    ¦ LONG
    ¦ CHAR(n)
    ¦ VARCHAR(n)
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the declaration section of a PL block.

Syntax Rules and Parameters

- Type-name
  - It specifies the name of a collection type to be used by a user.
- Element-type
  - It specifies the type of an element to be stored in a collection variable.
- Index-type
  - It specifies the data type of a key stored in a collection variable.

Description

It declares a collection type.

Examples

```
gSQL> DECLARE
TYPE rec IS TABLE OF VARCHAR(20) INDEX BY VARCHAR(20);
v1 rec;
v2 VARCHAR(20);
BEGIN
  DBMS_OUTPUT.PUT_LINE( 'first = ' || v1.first);
  DBMS_OUTPUT.PUT_LINE( 'last = '  || v1.last);
  DBMS_OUTPUT.PUT_LINE( 'prior = ' || v1.prior('aaa'));
  DBMS_OUTPUT.PUT_LINE( 'next = '  || v1.next('aaa'));
  DBMS_OUTPUT.PUT_LINE( 'count = ' || v1.count());
  FOR I IN 1 .. 10
  LOOP
      v1('a' || i) := 'a' || i;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE( 'first = ' || v1.first);
  DBMS_OUTPUT.PUT_LINE( 'last = '  || v1.last);
  DBMS_OUTPUT.PUT_LINE( 'count = ' || v1.count());
  DBMS_OUTPUT.PUT_LINE( 'Print all from first to last');
  v2 := v1.first;
  WHILE v2 IS NOT NULL
  LOOP
      DBMS_OUTPUT.PUT_LINE('Key= ' || v2 || ',Value=' || v1(v2));
```

```
      v2 := v1.next(v2);
  END LOOP;
  DBMS_OUTPUT.PUT_LINE( 'Print all from last to first');
  v2 := v1.last;
  WHILE v2 IS NOT NULL
  LOOP
      DBMS_OUTPUT.PUT_LINE('Key= ' || v2 || ',Value=' || v1(v2));
      v2 := v1.prior(v2);
  END LOOP;
END;
/
first =
last =
prior =
next =
count = 0
first = a1
last = a9
count = 10
Print all from first to last
Key= a1,Value=a1
Key= a10,Value=a10
Key= a2,Value=a2
Key= a3,Value=a3
Key= a4,Value=a4
Key= a5,Value=a5
Key= a6,Value=a6
Key= a7,Value=a7
Key= a8,Value=a8
Key= a9,Value=a9
Print all from last to first
Key= a9,Value=a9
Key= a8,Value=a8
Key= a7,Value=a7
Key= a6,Value=a6
Key= a5,Value=a5
Key= a4,Value=a4
Key= a3,Value=a3
Key= a2,Value=a2
Key= a10,Value=a10
Key= a1,Value=a1
```

```
Anonymous PL block executed.
```

Compatibility

The SQL standard does not define it.

For More Information

Refer to **Collection Method Invocation**.

28.8 CONTINUE Statement

Function

It stops currently performing statement list, and performs the next iteration of a superordinate loop statement.

Syntax

```
<continue statement> ::=
    CONTINUE [ label_name ] [ WHEN condition ]
    ;
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the body section of a PL block.

A statement with a target label should be one of the following loop family statements.

- basic loop statement
- for loop statement
- while statement
- forall statement

Syntax Rules and Parameters

- Label name
  - It can have an identifier chain form.
- Condition
  - If it is specified, it returns to a loop statement only when the condition is TRUE.

Description

It stops currently performing statement list, and returns to the superordinate loop statement.

If a label is specified, it returns to the superordinate loop statement of the label name.
If a label is not specified, it returns to the nearest superordinate loop statement.
If multiple superordinate loop statements with the same names exist, then the nearest statement is select ed.
It can return to a loop statement (exist in a nested scope) which is visible in the current location.

If a condition is specified, then it returns only when the condition is TRUE.
If a condition is not specified, then it definitely returns.

Examples

```
DECLARE
  V1 INTEGER := 1;
BEGIN
  <<AAA>>
  WHILE V1 <= 10 LOOP
    DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
    V1 := V1 + 1;
    IF V1 <= 2 THEN
      DBMS_OUTPUT.PUT_LINE( 'CONTINUE' );
      CONTINUE;
    ELSE
      EXIT;
    END IF;
    DBMS_OUTPUT.PUT_LINE( 'END-OF-WHILE' );
  END LOOP AAA;
END;
/
V1 = 1
CONTINUE
V1 = 2
Anonymous PL block executed.
```

Compatibility

〈continue statement〉 statement is similar to 〈iterate statement〉 of the SQL standard.
However, 〈iterate statement〉 statement does not provide WHEN condition feature.

For More Information

Refer to the followings.

- **EXIT Statement**
- **GOTO Statement**

28.9 Cursor FOR LOOP Statement

Function

It performs loops as many times as the number of rows in the result created by a query or a cursor declare d by a user in PSM.

Syntax

```
<Cursor For Loop statement> ::=
       FOR <Variable_Name> IN <Cursor>
       LOOP
            { <SQL procedure statement> ; }...
       END LOOP [ Label_Name ]
       ;
<Cursor> ::=
       < ( Implicit_Cursor_Query ) >
     ¦ < Explict_Cursor_Name > [ ( [ <actual param> ] ) ]

<Implicit_Cursor_Query> ::=
       SELECT statement
     ¦ SELECT_FOR_UPDATE statement
     ¦ INSERT_RETURNING_QUERY statement
     ¦ UPDATE_RETURNING_QUERY statement
     ¦ DELETE_RETURNING_QUERY statement
<actual param> ::=
      ( expression [ , expression ] .. )
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the body of PSM.

Syntax Rules and Parameters

The variable declared within For~Loop is valid only within that loop scope. (The variable can not be refere nced from outside of that Cursor For Loop Block Scope)

When Using Cursor Name

A cursor should already have been declared before performing LOOP by using a cursor name.
For more information about an actual param, refer to **OPEN Statement**.

When Using Cursor Query

It can perform only the query which can be internally processed by using an implicit cursor in GOLDILOCK S such as a select and a returning query.

Description

It performs PSM statements within a loop by turning around loops as many time as the number of results created by a cursor.
If a cursor becomes invalid (e.g. closed) during LOOP, it does not perform the loop and it processes itas a n error.
If an explicit cursor name is specified and the corresponding cursor is already opened, then it is processed as an error.

The variable to which a result of the cursor specified in FOR LOOP clause is returned is automatically creat ed. (It is created as a row type of the result set to be returned by an execution result of a cursor.)
However, if an alias for a select target expression which is not a column of a specific table among results of a user cursor query is not specified, then an error may occur.

Examples

Using Explicit Cursor

```
DECLARE
CURSOR c1 IS SELECT * FROM T1;
BEGIN
```

```
  FOR rec IN C1
  LOOP
    DBMS_OUTPUT.PUT_LINE( 'RowCount=' || c1%rowcount || ',C1=' || rec.c1 || ', C2=' ||
rec.c2);
  END LOOP;
END;
/
RowCount=1,C1=1, C2=1
RowCount=2,C1=2, C2=2
RowCount=3,C1=3, C2=3
RowCount=4,C1=4, C2=4
RowCount=5,C1=5, C2=5
RowCount=6,C1=6, C2=6
RowCount=7,C1=7, C2=7
RowCount=8,C1=8, C2=8
RowCount=9,C1=9, C2=9
RowCount=10,C1=10, C2=10
Anonymous PL block executed.
```

Using Cursor Query

```
BEGIN
  FOR rec IN (select * from t1)
  LOOP
      DBMS_OUTPUT.PUT_LINE( 'RowCount=' || sql%rowcount || ',C1=' || rec.c1 || ', C2=' ||
rec.c2);
  END LOOP;
END;
/
C1=1, C2=1
C1=2, C2=2
C1=3, C2=3
C1=4, C2=4
C1=5, C2=5
C1=6, C2=6
C1=7, C2=7
C1=8, C2=8
C1=9, C2=9
C1=10, C2=10
Anonymous PL block executed.
```

For More Information

Refer to the followings.

- **Explicit Cursor Declaration and Definition**
- **GOTO Statement**
- **EXIT Statement**

28.10 Cursor Variable Declaration

Function

It declares a cursor variable in DECLARE section of PSM.

Syntax

```
<cursor variable declaration> ::=
    variable_name <type>
    ;
<cursor type definition> ::=
      TYPE <type_name> IS REF CURSOR [ RETURN <return type> ]
<return type> ::=
      <table_name ¦ view_name ¦ cursor_name ¦ cursor_variable > % ROWTYPE
    ¦ <record_variable_name> % TYPE
    ¦ <record_type_name>
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the declaration section of PSM.

Syntax Rules and Parameters

Specifying the initial value of a cursor variable or assigning a cursor variable is allowed only between curs or variables.

Description

A cursor variable is operated like as a pointer indicating a cursor which is not dependent on a specific curs or.

Examples

```
DECLARE
TYPE rec IS RECORD (V1 VARCHAR(20), V2 VARCHAR(20));
TYPE cv IS REF CURSOR RETURN rec;
BEGIN
    NULL;
END;
/
Anonymous PL block executed.
```

For More Information

Refer to the followings.

- **OPEN Statement**
- **FETCH Statement**
- **CLOSE Statement**

28.11 DELETE Statement Extension

Function

It can store the result in RETURNING INTO clause by using a record type variable of PSM.

Syntax

```
<PSM delete statement extension: searched> ::=
    DELETE [ FROM ] table_name [ [ AS ] alias_name ]
        [ WHERE <search condition> ]
        [ <result offset clause> ]
        [ <fetch limit clause> ]
        [ <returning into clause> ]
    ;
<delete statement: positioned> ::=
    DELETE [ FROM ] table_name [ [ AS ] alias_name ]
        WHERE CURRENT OF cursor_name
    ;
<result offset clause> ::=
    OFFSET skip_count [ ROW | ROWS ]
<fetch limit clause> ::=
      <fetch first clause>
    | <limit clause>
<fetch first clause> ::=
    FETCH [ FIRST | NEXT ] [ row_count ] [ ROW ONLY | ROWS ONLY ]
<limit clause>
    LIMIT { fetch_row_count | offset_row_count, fetch_row_count | ALL }
<returning into clause> ::=
    { RETURN | RETURNING } { * | { <value expression> [ [AS] alias_name] } [, ...] } INTO
variable_name [, ...]
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the body section of PSM.

Syntax Rules and Parameters

If a variable to be returned through RETURNING INTO is a record type, then it can not be used by mixing t ogether with a different variable type.

Description

It can store the result in RETURNING INTO clause by using a record type variable of PSM.

Examples

```
gSQL> CREATE TABLE T1( C1 VARCHAR(20), C2 VARCHAR(20));
Table created.
gSQL> COMMIT;
Commit complete.
gSQL> INSERT INTO T1 VALUES ('AAA', 'BBB'), ('BBB', 'CCC'), ('CCC', 'DDD');
3 rows created.
gSQL> COMMIT;
Commit complete.
gSQL> DECLARE
  rec t1%ROWTYPE;
BEGIN
  DELETE FROM T1 WHERE C1 = 'AAA' RETURNING * INTO rec;
  DBMS_OUTPUT.PUT_LINE('SQL%ROWCOUNT=' || SQL%ROWCOUNT);
  DBMS_OUTPUT.PUT_LINE('rec.c1=' || rec.c1 || ', rec.c2=' || rec.c2);
END;
/
SQL%ROWCOUNT=1
rec.c1=AAA, rec.c2=BBB
Anonymous PL block executed.
```

For More Information

Refer to **Deleting Data**.

28.12 EXCEPTION_INIT Pragma

Function

It sets the error code which is to be processed by a user-defined exception.

Syntax

```
< PRAGMA EXCEPTION_INIT > ::=
     PRAGMA EXCEPTION_INIT ( <Exception-Name>, <Internal-ErrorCode> )
     ;
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the declaration section of PSM.

Syntax Rules and Parameters

A predefined exception can not be used in an exception name which is used as an argument. (A predefined exception name can not be declared.)
An exception name which is used as an argument in the same PL block DECLARE clause should be declared in advance. (Declaration of an exception name in different BLOCK can not be referenced.)
<Internal-ErrorCode> should be an internal error code existing within DB SYSTEM. (SUCCESS code can not be set.

Description

A user explicitly declares an exception name corresponding to an error code of DB SYSTEM.

Examples

```
gSQL> DECLARE
user_exception_1 EXCEPTION;
PRAGMA EXCEPTION_INIT( user_exception_1, -17001);
BEGIN
  RAISE USER_EXCEPTION_1;
  EXCEPTION WHEN user_exception_1 THEN DBMS_OUTPUT.PUT_LINE('User Exception_1');
END;
/
User Exception_1
Anonymous PL block executed.
```

Compatibility

Error codes are different each other according to a vendor, so it is not compatible each other.

For More Information

Refer to the followings.

- **Exception Declaration**
- **Exception Handler**

28.13 Exception Declaration

Function

It declares an exception name within a PL block.

Syntax

```
< Exception-Declaration Statement > ::=
     <Exception-Name>  EXCEPTION
     ;
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the declaration section of PSM.

Syntax Rules and Parameters

It can not declare a predefined exception name.
Duplicated declarations are not allowed in DECLARE clause of the same SCOPE.

Description

A user explicitly declares an exception.

Examples

```
DECLARE
user_exception_1 EXCEPTION;
```

```
user_exception_2 EXCEPTION;
user_exception_3 EXCEPTION;
BEGIN
  RAISE USER_EXCEPTION_2;
  EXCEPTION WHEN user_exception_1 THEN DBMS_OUTPUT.PUT_LINE('User Exception_1');
            WHEN user_exception_2 THEN DBMS_OUTPUT.PUT_LINE('User Exception_2');
            WHEN user_exception_3 THEN DBMS_OUTPUT.PUT_LINE('User Exception_3');
END;
/
User Exception_2
Anonymous PL block executed.
```

Compatibility

An exception declaration of the standard SQL is as follows, but GOLDILOCKS supports the syntax as abov e.

```
<condition declaration> ::=
DECLARE <condition name> CONDITION [ FOR <sqlstate value> ]
```

For More Information

Refer to the followings.

- **Exception Declaration**
- **EXCEPTION_INIT Pragma**

28.14 Exception Handler

Function

It performs an operation defined for an exception which is explicitly occurred by a user or an operation de fined for an implicit error due to a DB SYSTEM error occurred during performing PL/ SQL.

Syntax

```
< Exception Handler Statement > ::=
       EXCEPTION < Exception_When_List >
       ;
< Exception_When_List > ::=
       WHEN < Exception_Name_List > THEN <excutable statement list>  [ WHEN OTHERS THEN
<excutable statement list> ]
< Exception_Name_List > ::=
        <Exception_Name> [ { OR <Exception_Name> }... ]
```

Invocation and Access Rules

It can be used within a PL block.

Syntax Rules and Parameters

OTHERS (predefined exception) can not be specified together with another exception name by using OR. Duplicated specifying of OTHERS (predefined exception) is not allowed in an exception handler, and OTH ERS should be specified at the last.

Description

Exception Types

Type	Definer	Has error	Has name	Raise implicitly	Raise explicitly
Predefined	System	Yes	Yes	Yes	Optionally
User-defined	User	If user assign	If user assign	No	Yes

A predefined exception has an exception name and an error code which are specified in advance in GOL DILOCKS.
Other exceptions are classified into an internally defined exception and a user-defined exception. An inter nally defined exception is that a user sets the internal error code name of GOLDILOCKS differently from a predefined exception name, and a user-defined exception is that only the exception name is declared wit hout specifying a separate error code.

Predefined Exception

Table 28-6 Predefined exception type

Name	Description
CASE_NOT_FOUND	It can not satisfy all conditions of CASE WHEN, or ELSE clause is not defined.
DUP_VAL_ON_INDEX	INDEX duplicated error occurred.
INVALID_CURSOR	A cursor status is incorrect.
INVALID_NUMBER	It can not be converted to a number.
NO_DATA_FOUND	SELECT statement returns zero data.
ROWTYPE_MISMATCH	Field types of two RowType variables are different each other.
TOO_MANY_ROWS	It returns two or more rows.
VALUE_ERROR	It is an error such as type mismatch and invalid casting.
ZERO_DIVIDE	It tries dividing by 0.
OTHERS	It includes errors which are not defined in a predefined.

Examples

```
gSQL> DECLARE
V1 INTEGER := 0;
BEGIN
   DBMS_OUTPUT.PUT_LINE('Step1');
   V1 := 1 / 0;
```

```
   DBMS_OUTPUT.PUT_LINE('Step2');
   EXCEPTION WHEN OTHERS THEN DBMS_OUTPUT.PUT_LINE( 'Exception V1=' || V1);
END;
/
Step1
Exception V1=0
Anonymous PL block executed.
```

Compatibility

It does not support the SQL standard grammar.

For More Information

Refer to the followings.

- **EXCEPTION_INIT Pragma**
- **Exception Declaration**

28.15 EXECUTE IMMEDIATE Statement

Function

It executes a dynamic SQL within PSM.

Syntax

```
<EXECUTE IMMEDIATE statement> ::=
    EXECUTE IMMEDIATE <dynamic-sql> [ <binding-parameters> ]
    ;
<dynamic-sql> ::=
      single_quote_string
    ¦ psm_variable
<binding-parameters> ::=
      <into-clause>
    ¦ <using-clause>
    ¦ <returning-into-clause>
    ¦ <into-clause> <using-clause>
    ¦ <using-clause> <returning-into-clause>
<into-clause> ::=
    INTO psm_variable [ {, psm_variable} ... ]
<using-clause> ::=
    USING [ <Bind-Type> ] <expression> [ {, [ <Bind-Type> ] <expression>} ... ]
<Bind-Type> ::=
     IN
   ¦ OUT
   ¦ IN OUT
<returing-into-clause> ::=
    RETURNING INTO psm_variable [ {, psm_variable} ... ]
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the body section of a PL block.

Syntax Rules and Parameters

Dynamic SQL

- single_quote_string: It is an SQL statement enclosed with a single quote ('). (double_quote_string is recognized as a variable in PSM.)
- Psm-variable: It is an SQL statement stored in a variable which is used in PSM.

The following is an example of expressing a data by using quote(s) in an SQL statement to be performed.

```
EXECUTE IMMEDIATE 'INSERT INTO t1 VALUES ( ''Tom'' ) '; -- Tom
EXECUTE IMMEDIATE 'INSERT INTO t1 VALUES ( ''Tom''''House'') '; -- Tom'House
EXECUTE IMMEDIATE 'INSERT INTO t1 VALUES ( ''''''Tom'') ';   -- 'Tom
EXECUTE IMMEDIATE 'INSERT INTO t1 VALUES ( CHR(39) || ''TOM'' ) '; -- 'Tom
```

A marker (? or :V1) is used in a location where a user input a variable in a dynamic SQL.
An SQL statement specified in a dynamic SQL should be valid.

INTO Clause

The result of performing a dynamic SQL exists, and it is not bound through a marker, but the result of processing an SQL statement is returned. (It is internally a form of an implicit cursor fetch.)
The syntax is as follows.

```
EXECUTE IMMEDIATE 'SELECT ... FROM .. WHERE ...';
EXECUTE IMMEDIATE 'INSERT ... RETURNING ...';
EXECUTE IMMEDIATE 'UPDATE ... RETURNING ...';
EXECUTE IMMEDIATE 'DELETE ... RETURNING ...';
```

USING Clause

It lists a variable or an expression in USING clause (IN can be omitted.) as many as the number of input variables used in a dynamic SQL.
If the result of performing a dynamic SQL exists, then it lists variables as many as the number of result columns in USING OUT clause. (when returning the results to INTO clause)
The result can be returned in the following syntax by using USING OUT.

```
EXECUTE IMMEDIATE 'SELECT x, y, z INTO :v1, :v2, :v3 ...';
EXECUTE IMMEDIATE 'INSERT INTO ...  RETURNING C1, C2 INTO :V1, :V2';
EXECUTE IMMEDIATE 'UPDATE T1 SET .. RETURNING C1, C2 INTO :V1, :V2';
```

```
EXECUTE IMMEDIATE 'DELETE FROM ...  RETURNING C1, C2 INTO :V1, :V2';
```

RETURNING Clause

If INSERT/UPDATE/DELETE RETURNING INTO statement is used as a dynamic SQL, the result is returned b y binding a variable specified in USING clause in OUT-mode in GOLDILOCKS.
The same result can be returned by using RETURNING-INTO clause for the compatibility with other DBMS.

Other Rules

- DDL/ DCL can not use any BIND clause. (INTO, USING, RETURNING clause)
- It is used as OUT in INTO clause and RETURNING INTO clause, so a separate bind type can not be spec ified nor it be simultaneously used together.
- IN BIND type can use only a scalar type variable.
- OUT BIND type can use a record type variable, but a scalar type or a record type can not be listed bein g mixed together.
- A result can not be returned being divided through INTO clause and USING OUT or RETURNING INTO.

Description

- If a dynamic SQL returns the result as a variable specified in INTO clause
  - A query returning two or more results causes TOO_MANY_ROWS exception.
  - If the result is zero, it causes NO_DATA_FOUND exception.
- If a dynamic SQL returns the result as a variable specified in USING or RETURNING clause
  - If two or more variables which are not ARRAY are returned, it causes TOO_MANY_ROWS excepti on.
  - If the result is zero, an error does not occur.
- The followings are results of which DDL/ DCL performs implicit cursor SQL%Attribute variables.
  - SQL%ROWCOUNT = 0
  - SQL%ISOPEN = FALSE
  - SQL%FOUND = FALSE
  - SQL%NOTFOUND = TRUE
- Other dynamic SQLs store SQL%Attribute values corresponding to the processing results of that SQL statement.

Examples

```
gSQL> DECLARE
V1 INTEGER;
V2 VARCHAR(20);
BEGIN
    V1 := 1;
    V2 := 'abcdef';
    DBMS_OUTPUT.PUT_LINE('#INSERT');
    EXECUTE IMMEDIATE 'insert into t1 values (:a1, :a2)' USING v1, v2;
    EXECUTE IMMEDIATE 'select c1, c2 from t1 where c1 = 1' INTO v1, v2;
    DBMS_OUTPUT.PUT_LINE('C1='|| v1 || ', C2=' || v2);
    V1 := 1;
    V2 := 'xyz';
    DBMS_OUTPUT.PUT_LINE('#UPDATE');
    EXECUTE IMMEDIATE 'update t1 set c2 = :a1 where c1 = :a2' USING V2, V1;
    V1 := 1;
    V2 := '';
    DBMS_OUTPUT.PUT_LINE('#SELECT');
    EXECUTE IMMEDIATE 'select c1, c2 from t1 where c1 = :a1' INTO v1, v2 USING v1;
    DBMS_OUTPUT.PUT_LINE('C1='|| v1 || ', C2=' || v2);
    V1 := 1;
    V2 := '';
    DBMS_OUTPUT.PUT_LINE('#DELETE');
    EXECUTE IMMEDIATE 'delete from t1 where c1 = :a1' USING v1;
END;
/
#INSERT
C1=1, C2=abcdef
#UPDATE
#SELECT
C1=1, C2=xyz
#DELETE
Anonymous PL block executed.
```

28.16 EXIT Statement

Function

It exits a loop statement which has the given label among superordinate loop statements, then performs the next statement.

Syntax

```
<exit statement> ::=
    EXIT [ label_name ] [ WHEN condition ]
    ;
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the body section of a PL block.

Syntax Rules and Parameters

- Label name
  - It is a label name of a loop statement to be exited.
  - It can be in an identifier chain form.
- Condition
  - If a condition is specified, then it can exit a loop statement only when that condition is TRUE.

Description

- It stops currently performing statement list and exits a superordinate loop statement.
- A statement with a target label should be one of the following loop family statements.
  - basic loop statement
  - for loop statement

    - while statement
    - forall statement
- If a label is specified, then it exits a superordinate loop statement which has that label name.
- If a label is not specified, then it exits the nearest superordinate loop statement.
- If multiple superordinate loop statements with the same label names exist, then the nearest statemen t is selected.
- It can exit only a loop statement (exist in a nested scope) which is visible in the current location.

- If a condition is specified, then it exits only when the condition is TRUE.
- If a condition is not specified, then it definitely exits.

Examples

```
gSQL> DECLARE
  V1 INTEGER := 1;
BEGIN
  <<AAA>>
  WHILE V1 <= 10 LOOP
    DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
    EXIT;
    V1 := V1 + 1;
  END LOOP AAA;
END;
/
V1 = 1
Anonymous PL block executed.
```

Compatibility

It does not exist in the SQL standard.

28.17 Explicit Cursor Attribute

Function

It returns the status value of a cursor defined in PSM.

Syntax

```
〈Explicit cursor attribute〉 ::=
    cursor_name '%' { ISOPEN ¦ FOUND ¦ NOTFOUND ¦ ROWCOUNT }
```

Invocation and Access Rules

It can be used only in the body section of PSM.

Syntax Rules and Parameters

- Cursor_name
  - It is the name of a cursor whose status value is to be obtained.

Description

- It returns the status value of a given cursor.
  - ISOPEN: It is whether the current cursor is OPEN.
  - FOUND: It is whether the data is returned by the recent fetch.
  - NOTFOUND: It is an opposite of FOUND.
  - ROWCOUNT: It is the number of fetched records after the cursor is recently OPEN.
- The following values are returned according to the cursor status.

Table 28-7 Results according to the performing moment

Attribute name	Before OPEN	After OPEN	After FETCH	After CLOSE
ISOPEN	FALSE	TRUE	TRUE	FALSE

Attribute name	Before OPEN	After OPEN	After FETCH	After CLOSE
FOUND	NULL	NULL	TRUE/FALSE	NULL
NOTFOUND	NULL	NULL	TRUE/FALSE	NULL
ROWCOUNT	NULL	NULL	N (the number)	NULL

Examples

```
gSQL> CREATE TABLE T1 ( I1 INTEGER );
Table created.
gSQL> COMMIT;
Commit complete.
gSQL> DECLARE
  CURSOR C1 IS SELECT * FROM T1;
  V1 INTEGER := 0;
  V2 INTEGER := 0;
  TOTAL INTEGER := 0;
BEGIN
  FOR I IN 1..100 LOOP
    INSERT INTO T1 VALUES( I );
  END LOOP;
  COMMIT;
  IF NOT C1%ISOPEN THEN
    OPEN C1;
  END IF;
  LOOP
    FETCH C1 INTO V1;
    EXIT WHEN C1%NOTFOUND;
    TOTAL := TOTAL + V1;
    V2 := V1;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE( 'COUNT = ' || C1%ROWCOUNT || ' TOTAL = ' || TOTAL );
  CLOSE C1;
END;
/
COUNT = 100 TOTAL = 5050
Anonymous PL block executed.
```

Compatibility

It is not defined in the SQL standard.

28.18 Explicit Cursor Declaration and Definition

Function

It declares a cursor in DECLARE section of PSM.

Syntax

```
<cursor declaration> ::=
    CURSOR cursor_name [ <cursor param spec> ] RETURN rowtype
    ;
<cursor param spec> ::=
      ( <cursor param decl> [ , <cursor param decl> ] .. )
<cursor param decl> ::=
      param_name [ IN ] datatype [ { ':=' ¦ DEFAULT } expression ]
<cursor definition> ::=
    CURSOR cursor_name [ <cursor param spec> ] [ RETURN rowtype ]
    IS <cursor query>
    ;
<cursor query> ::==
    select statement
    ¦ select for update_statement
    ¦ insert returning query statement
    ¦ update returning query statement
    ¦ delete returning query statement
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the declaration section of a PL block.

Syntax Rules and Parameters

Cursor Name

It is a cursor name to be declared.
The length of a cursor name should be shorter than 128 bytes.
It should be a unique name in that scope.

RowType

It defines a record type of a cursor.
The number of select targets specified when defining a cursor should be same, and the data type should be compatible.
If a rowtype is not specified, then a rowtype which is appropriate to a SELECT target of select_statement specified when defining a cursor is automatically specified.

Param Name

It is a name which distinguishes parameters within a specific cursor.
It should be a unique name in that cursor.
If the name is as same as a name of another variable which can be referenced within a scope, then a para meter of that cursor is preferentially referenced.

DataType

It specifies the data type of the corresponding parameter.
It can use all built-in types provided by GOLDILOCKS and types defined within PSM.
However, a statement which restricts a scope (precision/ scale) can not be specified in a built-in type, but it is internally specified as the maximum scope of the corresponding data type.

Cursor Query

- The cursor can execute the following queries.
  - Select statement
  - Select For Update statement
  - Insert Returning statement
  - Update Returning statement
  - Delete Returning statement

- SELECT INTO statement is not available.

Description

- <explicit cursor declaration> statement declares or defines a cursor.
  - Cursor declaration: It declares only a name and format of a cursor.
  - Cursor definition: It detailedly defines a name, format of a cursor and SELECT statement to be pe rformed.
- An explicit cursor is used by defining it after declaration. or it can be used by defining it without a dec laration.
- When using an explicit cursor by defining it after declaration, then the cursor name, the parameter sp ecification and the record type definition should exactly match.
- A declaration and a definition of an explicit cursor should exist in the same block.
- An explicit cursor is created when the cursor enters the defined block, and it is automatically CLOSEd and deleted when it exits the block.
- 'MAXIMUM_NAMED_CURSOR_COUNT' property restricts the maximum number of explicit cursors w hich can be created in a specific time.
- The following variables can be used in <cursor query> statement performed by an explicit cursor.
  - Parameter of the cursor
  - All PSM variables which can be referenced in a scope at the time of declaration. (It is not a scope of the time of open.)
  - External bind parameter (in case of an anonymous PL block)
- An explicit cursor performing <cursor query> specified by a user has the following properties.
  - IN_SENSITIVE (Changes by another transaction do not affect it.)
  - NON_SCROLLABLE (It can not fetch the previous record again.)
  - READ_ONLY (Only a read operation is available.)
  - WITH-HOLD (A cursor is not automatically closed even though COMMIT/ ROLLBACK is performe d.)

Examples

```
gSQL> CREATE TABLE t1( c1 INTEGER, c2 VARCHAR( 6 ) );
Table created.
gSQL> COMMIT;
Commit complete.
```

- Insert Returning Statement

```
gSQL>
DECLARE
   var1 t1%ROWTYPE;
   CURSOR cur1( p1 INTEGER, p2 varchar(6) ) RETURN t1%ROWTYPE
      IS INSERT INTO t1 VALUES ( p1, p2 ) RETURNING *;
BEGIN
   OPEN cur1( 1, 'aaa' );

   FETCH cur1 INTO var1;
   DBMS_OUTPUT.PUT_LINE( 'var1.c1 = ' || var1.c1 || ' , var1.c2 = ' || var1.c2 );
   CLOSE cur1;
END;
/
var1.c1 = 1 , var1.c2 = aaa
Anonymous PL block executed.
```

- Select Statement

```
gSQL>
DECLARE
   var1 t1%ROWTYPE;
   CURSOR cur1( p1 INTEGER, p2 varchar(6) ) RETURN t1%ROWTYPE IS
      SELECT * FROM t1 WHERE c1 = p1 AND c2 = p2;
BEGIN
   OPEN cur1( 1, 'aaa' );
   FETCH cur1 INTO var1;
   DBMS_OUTPUT.PUT_LINE( 'var1.c1 = ' || var1.c1 || ' , var1.c2 = ' || var1.c2 );
   CLOSE cur1;
END;
/
var1.c1 = 1 , var1.c2 = aaa
Anonymous PL block executed.
```

- Update Returning Statement

```
gSQL>
DECLARE
   var1 t1%ROWTYPE;
   CURSOR cur1( p1 INTEGER, p2 varchar(6) ) RETURN t1%ROWTYPE IS
      UPDATE t1 SET c1 = p1, c2 = p2 RETURNING *;
BEGIN
```

```
   OPEN cur1( 3, 'ccc' );

   FETCH cur1 INTO var1;
   DBMS_OUTPUT.PUT_LINE( 'var1.c1 = ' || var1.c1 || ' , var1.c2 = ' || var1.c2 );
   CLOSE cur1;
END;
/
var1.c1 = 3 , var1.c2 = ccc
Anonymous PL block executed.
```

- Delete Returning Statement

```
gSQL>
DECLARE
   var1 t1%ROWTYPE;
   CURSOR cur1( p1 INTEGER, p2 varchar(6) ) RETURN t1%ROWTYPE IS
      DELETE FROM t1 WHERE c1 = p1 AND c2 = p2 RETURNING *;
BEGIN
   OPEN cur1( 3, 'ccc' );

   FETCH cur1 INTO var1;
   DBMS_OUTPUT.PUT_LINE( 'var1.c1 = ' || var1.c1 || ' , var1.c2 = ' || var1.c2 );
   CLOSE cur1;
END;
/
var1.c1 = 3 , var1.c2 = ccc
Anonymous PL block executed.
```

Compatibility

It is not defined in the SQL standard.

For More Information

Refer to the followings.

- **OPEN FOR Statement**
- **FETCH Statement**

- CLOSE Statement

28.19 FETCH Statement

Function

It fetches a single record of OPEN cursor.

Syntax

```
<fetch statement> ::=
    FETCH cursor_name <into clause>
    ;
<into clause> ::=
    INTO { variable [ , variable ] .. ¦ record }
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the body section of a PL block.

Syntax Rules and Parameters

- Cursor name
  - It is a cursor name to be fetched.
- Variable
  - It is a scalar type variable or a bind parameter which stores a column value among the fetch results.
- Record
  - It is a record type variable to store the entire single record of which is the fetch result.

Description

It fetches a record from an open cursor, then copies the value to a variable specified in INTO clause.
If a cursor is declared only but not defined, then an error occurs.
The cursor should be open.

A variable type given to INTO clause should be compatible with a data type of the fetched record result.
The number of variables given to INTO clause should be as same as the number of the cursor's SELECT tar gets.
However, if a variable given in INTO clause is a record type, then only a single variable should be specified.
Also, the number of the record variable fields should be as same as the number of SELECT targets.

If a fetch is called when a record to be fetched does not exist, then the value of target variables in INTO cl ause is not altered.

Examples

```
gSQL> CREATE TABLE T1 ( I1 INTEGER );
Table created.
gSQL> COMMIT;
Commit complete.
gSQL> DECLARE
  CURSOR C1 IS SELECT * FROM T1;
  V1 INTEGER := 0;
  V2 INTEGER := 0;
  CNT INTEGER := 0;
  TOTAL INTEGER := 0;
BEGIN
  FOR I IN 1..100 LOOP
    INSERT INTO T1 VALUES( I );
  END LOOP;
  COMMIT;
  OPEN C1;
  LOOP
    FETCH C1 INTO V1;
    CNT := CNT + 1;
    TOTAL := TOTAL + V1;
    V2 := V1;
    EXIT WHEN V1 = 100;
```

```
  END LOOP;
  CLOSE C1;
  DBMS_OUTPUT.PUT_LINE( 'CNT = ' || CNT || ' TOTAL = ' || TOTAL );
END;
/
CNT = 100 TOTAL = 5050
Anonymous PL block executed.
```

Compatibility

It is not defined in the SQL standard.

For More Information

Refer to the followings.

- **OPEN Statement**
- **CLOSE Statement**

28.20 FOR LOOP Statement

Function

As long as an index variable has the given value scope, it performs internal statements by increasing or reversing the index variable by 1.

Syntax

```
<for loop statement> ::=
    FOR index_variable_name IN [ REVERSE ] lower_bound .. upper_bound
    LOOP { <SQL procedure statement> ; }... END LOOP [ loop_name ]
    ;
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the body section of a PL block.

Syntax Rules and Parameters

- Index variable name
  - It is a variable name to be used as an index in FOR statement. Internally, NATIVE_BIGINT type variable is used.
- Lower bound
  - It should be an integer type. If a number with a floating point is used, then the number below decimal point is rounded up while converting the type.
- Upper bound
  - It should be an integer type. If a number with a floating point is used, then the number below decimal point is rounded up while converting the type.

Description

for loop statement performs an internal statement list by increasing or decreasing the index variable value.

- If REVERSE is specified
  - an index variable is decreased by 1 from upper_bound value, and it terminates execution of *for loop* statement when the index variable value becomes smaller than lower_bound.
  - If upper_bound value is smaller than lower_bound then an internal statement list is not performed.
- If REVERSE is not specified
  - an index variable is increased by 1 from lower_bound, and it terminates execution of *for loop* statement when the index variable value becomes bigger than upper_bound.
  - If lower_bound value is bigger than upper_bound, then an internal statement list is not performed.

Examples

```
gSQL> BEGIN
  FOR I IN 0 .. 5 LOOP
    DBMS_OUTPUT.PUT_LINE( 'I = ' || I );
  END LOOP;
END;
/
I = 0
I = 1
I = 2
I = 3
I = 4
I = 5
Anonymous PL block executed.
gSQL> BEGIN
  FOR I IN REVERSE 0 .. 5 LOOP
    DBMS_OUTPUT.PUT_LINE( 'I = ' || I );
  END LOOP;
END;
/
I = 5
I = 4
I = 3
```

```
I = 2
I = 1
I = 0
Anonymous PL block executed.
```

Compatibility

It is not defined in the SQL standard.

For More Information

Refer to the followings.

- **CONTINUE Statement**
- **EXIT Statement**
- **GOTO Statement**

28.21 Function Declaration and Definition

Function

It declares and defines a function.

Syntax

```
<function declaration> ::=
        FUNCTION <function name> [ ( <parameter list> ) ] <return clause>
        ;

<function definition> ::=
        FUNCTION <function name> [ ( <parameter list> ) ] <return clause>
        { IS ¦ AS }
        <item declaration>
        BEGIN
        <pl statement list>
        END [ <function name> ]
        ;

<parameter list> ::=
      <parameter name> [ <parameter mode> ] <datatype> [ <parameter default> ] [ , ... ]

<parameter mode> ::=
      IN
    ¦ OUT
    ¦ IN OUT

<parameter default> ::=
      { := ¦ DEFAULT } <value expression>

<return clause> ::=
      RETURN <datatype>
    ¦ RETURN TABLE ( <table function column list> )

<table function column list> ::=
```

```
    <column name> <datatype> [ , ... ]
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the declaration section of a PL block.

Syntax Rules and Parameters

function name

It is a name of function to be created in a PL block, and it should be a unique name in a PL block.
In other words, PL item and function declared in PL block can not have the same name.
The length of a function name should be shorter than 128 bytes.

parameter name

It defines a parameter name of the function.
The name of each parameter should be unique in a function.
In other words, the function's parameter and PL item can not have the same name.
The length of a parameter name should be shorter than 128 bytes.
The maximum number of parameters available in a single function is limitless.

parameter mode

It sets each parameter mode.
The parameter modes are IN, OUT, and IN OUT.
If the parameter mode is not specified, the default mode is *IN*.

parameter default

It is the default value of the parameter.
The parameter with the specified parameter default can be omitted when executing the function.
If the parameter is not specified but omitted, then the default value is <value expression> specified when defining the parameter.
The datatype of <value expression> should be the datatype of the parameter.
All parameters defined after the parameter having <parameter default> should have <parameter default>.

return clause

It defines the return type of the function.
It is defined as follows in <return clause>.

- RETURN <datatype>
  - It defines the datatype of the return value returned by the function.
- RETURN TABLE (<table function column list>)
  - It defines the table type of the returned result set.

table function column list

It is the column name of the result set returned by the table function.
The length of a column name should be shorter than 128 bytes.
The number of columns are limitless.
Each column name is unique in <table function column list>.
The column name can be as same as the parameter name and the declare item name.
The column defined in <table function column list> can not be referenced in PL block of the function.

Item Declaration

It declares items such as a local variable to be used within a function.
It can declare all items which can be declared in a PL block.

PL Stmt List

It is a body section of a function, and it lists PL statements to be performed.

Description

It declares and defines the function as follows.

- PL block (e.g. anonymous block, procedure and function's block, block statement's block)
  - It is one of PL block's item and it declares and defines the function.
  - The function declared and defined in a PL block is a nested function.
  - Nested function is available only within the PL block range.
- Package specification
  - It declares the public function of the package.
  - The public function of the package is available in the database.

- Package body
  - It defines the public function declared in the package specification.
  - It declares and defines the private function of the package.
  - The private function of the package is available only within the package range.

The usage of the function is as same as that of schema-level function.

Examples

```
gSQL> CREATE TABLE t_score( c_grade INTEGER, c_score INTEGER );

Table created.

gSQL> INSERT INTO t_score VALUES ( 1 , 98 ) , ( 1 , 97 ) , ( 1 , 99 ),
                                 ( 2 , 95 ) , ( 2 , 98 ) , ( 2 , 92 ),
                                 ( 3 , 98 ) , ( 3 , 96 ) , ( 3 , 94 );

9 rows created.

gSQL> COMMIT;

Commit complete.
```

- Nested function

```
gSQL>
DECLARE
  v_max_score INTEGER;

  FUNCTION nestedfunc RETURN INTEGER;

  FUNCTION nestedfunc RETURN INTEGER AS
    v_max INTEGER;
  BEGIN
    SELECT MAX( c_score )
      INTO v_max
      FROM t_score;

    RETURN v_max;
  END;
```

```
BEGIN
  v_max_score := nestedfunc;

  DBMS_OUTPUT.PUT_LINE( 'Max Score : ' || v_max_score );
END;
/
Max Score : 99
Anonymous PL block executed.
```

- Package function

```
gSQL>
CREATE OR REPLACE PACKAGE pkg1 AS
  -- Declare Package Public Function
  FUNCTION func1( p_grade INTEGER, p_option VARCHAR ) RETURN INTEGER;
END;
/

Package created.
gSQL>
CREATE OR REPLACE PACKAGE BODY pkg1 AS
  -- Declare Package Private Function
  FUNCTION func2( p_grade INTEGER, p_option VARCHAR ) RETURN INTEGER;

  -- Define Package Public Function
  FUNCTION func1( p_grade INTEGER, p_option VARCHAR ) RETURN INTEGER AS
    var INTEGER;
  BEGIN
    var := func2( p_grade, p_option );

    RETURN var;
  END;

  -- Define Package Private Function
  FUNCTION func2( p_grade INTEGER, p_option VARCHAR ) RETURN INTEGER AS
    var INTEGER;
  BEGIN
    IF p_option = 'MIN' THEN
      SELECT MIN(c_score) INTO var FROM t_score WHERE c_grade = p_grade;
    ELSIF p_option = 'MAX' THEN
      SELECT MAX(c_score) INTO var FROM t_score WHERE c_grade = p_grade;
```

```
    ELSIF p_option = 'AVG' THEN
      SELECT AVG(c_score) INTO var FROM t_score WHERE c_grade = p_grade;
    ELSE
      var := -1;
    END IF;

    RETURN var;
  END;
END;
/

Package created.

gSQL> SELECT pkg1.func1(  1 , 'MAX' ) FROM DUAL;

PKG1.FUNC1(  1 , 'MAX' )
------------------------
                      99

1 row selected.
```

Compatibility

It is as same as a schema-level function.

For More Information

Refer to **CREATE FUNCTION**.

28.22 GOTO Statement

Function

It tries to jump into the nearest statement which has a given label among statements accessible from the current location.

Syntax

```
<goto statement> ::=
    GOTO label_name
    ;
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the body section of a PL block.

Syntax Rules and Parameters

- Label_name
  - It is a label name of a statement in which is to be jumped.
  - It can be in an identifier chain form.

Description

It starts performing by jumping into a statement which has the corresponding label name.
If multiple candidate statements exist, then it jumps into the nearest statement.
It can jump only to a statement (exist in a nested scope) which is visible in the current location.
Both forward jump and backward jump are possible.

Examples

```
gSQL> DECLARE
  V1 INTEGER := 0;
BEGIN
  <<LABEL1>>
  IF V1 > 0 THEN
    GOTO LABEL2;
  END IF;
  V1 := V1 + 1;
  DBMS_OUTPUT.PUT_LINE('a');
  GOTO LABEL1;
  DBMS_OUTPUT.PUT_LINE('b');
  <<LABEL2>>
  DBMS_OUTPUT.PUT_LINE('c');
END;
/
a
c
Anonymous PL block executed.
```

Compatibility

It is not defined in the SQL standard.

For More Information

Refer to the followings.

- **EXIT Statement**
- **CONTINUE Statement**

28.23 IF Statement

Function

It performs a statement list corresponding to the condition returning TRUE among the given conditions.

Syntax

```
<if statement> ::=
    IF <search condition> <if statement then clause>
    [ <if statement elsif clase> ]
    [ <if statement else clase> ]
    END IF
    ;
<if statement then clause> ::=
    THEN <executable statement list>
<if statement elsif clause> ::=
    ELSIF <search condition> THEN <executable statement list>
<if statement elsif clause> ::=
    ELSE <executable statement list>
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the body section of a PL block.

Syntax Rules and Parameters

- Search condition
  - It is an expression which can be finally evaluated as a boolean type.
- Executable statement list
  - It is a list of all statements supported by GOLDILOCKS PSM.

Description

Like as CASE statement, it performs statement lists of IF, ELSIF clauses returning TRUE by evaluating conditions.
If it can not satisfy any condition and <if statement else clause> exists, then it performs the corresponding statement.
ELSIF clauses evaluates conditional expressions in order, and stops evaluating after finding the TRUE conditional clause.

Examples

```
gSQL> DECLARE
  V1 INTEGER := 10;
BEGIN
  IF V1 > 0 THEN
    DBMS_OUTPUT.PUT_LINE( 'POSITIVE' );
  ELSIF V1 = 0 THEN
    DBMS_OUTPUT.PUT_LINE( 'ZERO' );
  ELSE
    DBMS_OUTPUT.PUT_LINE( 'NEGATIVE' );
  END IF;
END;
/
POSITIVE
Anonymous PL block executed.
```

Compatibility

<if statement> statement is as same as a syntax and an operation of the SQL standard.

Table 28-8 SQL standard compatibility

Feature ID	Description	Compatibility
P002	Computational completeness	O

28.24 Implicit Cursor Attribute

Function

It returns the status value of an implicit cursor defined in PSM.

Syntax

```
<Implicit cursor attribute> ::=
    SQL '%' { ISOPEN ¦ FOUND ¦ NOTFOUND ¦ ROWCOUNT }
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the body section of a PL block.

Description

- It stores the result of SQL statement which was executed just before.
  - ISOPEN: It is whether the cursor is OPEN, and it is always set to FALSE.
  - FOUND: It is whether the data is returned by the SQL result of just before.
  - NOTFOUND: It is an opposite of FOUND.
  - ROWCOUNT: It is the number of records affected by the SQL result of just before.

Examples

```
DECLARE
V1 INTEGER;
BEGIN
    SELECT COUNT(*) INTO V1 FROM T1;
    DBMS_OUTPUT.PUT_LINE('COUNT RET    = ' ¦¦ V1);
    DBMS_OUTPUT.PUT_LINE('SQL%ISOPEN   = ' ¦¦ SQL%ISOPEN);
```

```
    DBMS_OUTPUT.PUT_LINE('SQL%FOUND    = ' || SQL%FOUND);
    DBMS_OUTPUT.PUT_LINE('SQL%NOTFOUND = ' || SQL%NOTFOUND);
    DBMS_OUTPUT.PUT_LINE('SQL%ROWCOUNT = ' || SQL%ROWCOUNT);
END;
/
COUNT RET    = 0
SQL%ISOPEN   = FALSE
SQL%FOUND    = TRUE
SQL%NOTFOUND = FALSE
SQL%ROWCOUNT = 1
Anonymous PL block executed.
```

Compatibility

It is not defined in the SQL standard.

28.25 INSERT Statement Extension

Function

It is an extended feature of an insert statement to input data by specifying record type variable supported in PSM in VALUES clause.

Syntax

```
<PSM Insert Statement Extension Statement> ::=
    INSERT INTO table_name [ ( column_name [, ...] ) ]
           <Insert_source>
           [ <Returning_into_clause> ]
    ;
<Insert_source> ::=
      <value-list>
    ¦ <from_subquery>
    ¦ <from_default>
<from subquery> ::=
    <query_expression>
<from default> ::=
    DEFAULT VALUES
<Value-List> ::=
      VALUES <Value_item> [ , ...]
<value-Item> ::=
     PSM-Record-Type-Variable
     ¦  ( { <value expression> ¦ DEFAULT } [, ...] )
<Returning_into_clause> ::=
      [ RETURN ¦ RETURNING ] { * ¦ { <value_expression> [ [AS] alias_name ] } [, ...] INTO
variable_name [, ...]
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the body section of a PL block.

A PSM insert extension statement can not be used in an original SQL statement of EXECUTE IMMEDIATE.

Syntax Rules and Parameters

It is operated as same as the basic syntax of an insert statement. However, a feature specifying PSM recor d type variables are added other than a feature consecutively listing existing value expressions in parenthe ses in value item.

A PSM record type variable should be specified when using a variable without parentheses in Value_Item. When using it in an insert extension statement form, a variable which is not a record type can not be use d being mixed.

Description

It stores a record by using a record type variable of PSM other than a general insert statement, or obtains a result through returning into.
For more information about insert, refer to the following example.

Examples

```
gSQL> DECLARE
  rec t1%ROWTYPE;
BEGIN
    rec.i1 := 'AAA';
    rec.i2 := 'BBB';
    rec.i3 := 'CCC';
    INSERT INTO t1 (i1, i2, i3) VALUES rec ;
END;
/
Anonymous PL block executed.
gSQL> SELECT * FROM t1;
I1  I2  I3
--- --- ---
AAA BBB CCC
```

Compatibility

It is not defined in the SQL standard.

28.26 INSERT INTO ... UPDATE Statement Extension

Function

It is the extended function of *insert into .. update* statement. It creates a new row in a table by specifying the record type variable supported by PSM in a values clause, or updates the row by specifying the variable in a set clause.

Syntax

```
<PSM Upsert Statement Extention Statement > ::=
    INSERT INTO table_name [ ( column_name [, ...] ) ]
        <insert source>
        <duplicate key clause>
        [ <Returning_into_clause> ]
    ;
<insert source> ::=
      <values-list>
    ¦ <from subquery>
    ¦ <from_default>
<Value-List> ::=
      VALUES <Value_item> [ , ...]
<value-Item> ::=
         PSM-Record-Type-Variable
     ¦  ( { <value expression> ¦ DEFAULT } [, ...] )
<from subquery> ::=
    <query expression>
<from default> ::=
    DEFAULT VALUES
<duplicate key clause>
    ON DUPLICATE KEY { DO NOTHING ¦ <do update clause> }
<do update clause> ::=
    [DO] UPDATE <target-list>
<target-List> ::=
        SET <set clause> [, ...]
```

```
      ¦ SET ROW = <psm_variable>
<set clause> ::=
      column_name = { <value expression> ¦ DEFAULT }
    ¦ ( column_name [, ...] ) = ( { <value expression> ¦ DEFAULT } [, ...] )
    ¦ ( column_name [, ...] ) = ( <query expression> )
<returning_into_clause> ::=
      [ RETURN ¦ RETURNING ] { * ¦ { <value_expression> [ [AS] alias_name ] } [, ...] INTO
variable_name [, ...]
```

Invocation and Access Rules

It can be used only within PSM. (e.g. procedure, function, package)
It can be used only in the body section of a PL block.

Syntax Rules and Parameters

- <values clause>
  - The function specifying PSM record type variable other than consecutively listing existing value ex pressions in parentheses in the value item has been added.
  - PSM record type variable should be specified when using the variable without parentheses in the value item.
  - When describing by extending the value item, then the variable other than the record type variab le is not allowed.
- <target list>
  - <PSM_Variable> used in SET ROW statement should be the variable declared as the record type.
- <returning into clause>
  - <Variable> used in RETURNING INTO statement does not need to be a record type. However, if it is specified as a record type, then it can not be used together with other data type variables nor c an list two or more record type variables.

Description

Returning into statement in an upsert statement stores the inserted result when insert is executed, and it stores the updated result when update is executed.

Examples

- Execute insert.

```
gSQL> CREATE TABLE t1( c1 INTEGER UNIQUE, c2 INTEGER, c3 INTEGER );
Table created.
gSQL> DECLARE
  v_rec t1%ROWTYPE;
BEGIN
  v_rec.c1 := 1;
  v_rec.c2 := 1;
  v_rec.c3 := 1;

  INSERT INTO t1 VALUES v_rec ON DUPLICATE KEY UPDATE SET c1 = c1 + 1;
END;
/
Anonymous PL block executed.
gSQL> SELECT * FROM t1;
C1 C2 C3
-- -- --
 1  1  1
1 row selected.
```

- Execute update.

```
gSQL> CREATE TABLE t1( c1 INTEGER UNIQUE, c2 INTEGER, c3 INTEGER );
Table created.
gSQL> INSERT INTO t1 VALUES ( 1 , 1 , 1 );
1 row created.
gSQL> DECLARE
  v_rec t1%ROWTYPE;
BEGIN
  v_rec.c1 := 2;
  v_rec.c2 := 2;
  v_rec.c3 := 2;
  INSERT INTO t1 VALUES (1, 1, 1) ON DUPLICATE KEY UPDATE SET ROW = v_rec;
END;
/
gSQL> SELECT * FROM t1;
C1 C2 C3
```

```
-- -- --
 2  2  2
1 row selected.
```

- Returning statement of when executing insert

```
gSQL> CREATE TABLE t1( c1 INTEGER UNIQUE, c2 INTEGER, c3 INTEGER );
Table created.
gSQL> DECLARE
  v_rec1 t1%ROWTYPE;
  v_rec2 t1%ROWTYPE;
BEGIN
  v_rec1.c1 := 1;
  v_rec1.c2 := 1;
  v_rec1.c3 := 1;
  INSERT INTO t1 VALUES v_rec1 ON DUPLICATE KEY UPDATE SET c1 = c1 + 1 RETURNING * INTO
v_rec2;
  DBMS_OUTPUT.PUT_LINE( 'v_rec2.c1 : ' || v_rec2.c1 );
  DBMS_OUTPUT.PUT_LINE( 'v_rec2.c2 : ' || v_rec2.c2 );
  DBMS_OUTPUT.PUT_LINE( 'v_rec2.c3 : ' || v_rec2.c3 );
END;
/
v_rec2.c1 : 1
v_rec2.c2 : 1
v_rec2.c3 : 1
Anonymous PL block executed.
gSQL> SELECT * FROM t1;
C1 C2 C3
-- -- --
 1  1  1
1 row selected.
```

- Returning statement of when executing update

```
gSQL> CREATE TABLE t1( c1 INTEGER UNIQUE, c2 INTEGER, c3 INTEGER );
Table created.
gSQL> INSERT INTO t1 VALUES ( 1 , 1 , 1 );
1 row created.
gSQL> DECLARE
  v_rec1 t1%ROWTYPE;
  v_rec2 t1%ROWTYPE;
BEGIN
```

```
  v_rec1.c1 := 2;
  v_rec1.c2 := 2;
  v_rec1.c3 := 2;
  INSERT INTO t1 VALUES (1, 1, 1) ON DUPLICATE KEY UPDATE SET ROW = v_rec1 RETURNING * INTO
v_rec2;
  DBMS_OUTPUT.PUT_LINE( 'v_rec2.c1 : ' || v_rec2.c1 );
  DBMS_OUTPUT.PUT_LINE( 'v_rec2.c2 : ' || v_rec2.c2 );
  DBMS_OUTPUT.PUT_LINE( 'v_rec2.c3 : ' || v_rec2.c3 );
END;
/
v_rec2.c1 : 2
v_rec2.c2 : 2
v_rec2.c3 : 2
Anonymous PL block executed.
gSQL> SELECT * FROM t1;
C1 C2 C3
-- -- --
 2  2  2
1 row selected.
```

Compatibility

It is not defined in the SQL standard.

28.27 NULL Statement

Function

It is a statement without any feature.

Syntax

```
<null statement> ::=
    NULL
    ;
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the body section of a PL block.

Description

It is a statement without any feature, and used to set a label of a specific location.

Examples

```
gSQL> BEGIN
  FOR i in 1..10 LOOP
    DBMS_OUTPUT.PUT_LINE( i );
    IF i > 5 THEN
      GOTO label1;
    END IF;
  END LOOP;
  <<label1>>
  NULL;
```

```
END;
/
1
2
3
4
5
6
Anonymous PL block executed.
```

Compatibility

It is not defined in the SQL standard.

28.28 OPEN Statement

Function

It executes SELECT statement of a cursor defined in PSM.

Syntax

```
<open statement> ::=
    OPEN cursor_name [ <actual param spec> ]
    ;
<actual param spec> ::=
      ( expression [ , expression ] .. )
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the body section of a PL block.

Syntax Rules and Parameters

- Cursor_name
  - It is the name of a cursor to be opened.

Description

It executes SELECT or SELECT ... FOR UPDATE statement of a defined cursor.
If a cursor is declared only but is not defined, then an error occurs.
Values of actual parameters should be compatible with the data type of those parameters.

The number of actual parameters should be same as the number of parameters of a cursor.
If it is smaller than the number of parameters of a cursor, then the default value should be specified in all

other parameters.

Examples

```
gSQL> CREATE TABLE T1 ( I1 INTEGER, I2 VARCHAR(10) );
Table created.
gSQL> COMMIT;
Commit complete.
gSQL> INSERT INTO T1 VALUES( 1, 'AAA' );
1 row created.
gSQL> INSERT INTO T1 VALUES( 2, 'BBB' );
1 row created.
gSQL> INSERT INTO T1 VALUES( 3, 'CCC' );
1 row created.
gSQL> COMMIT;
Commit complete.
gSQL> DECLARE
  CURSOR C1( A1 INTEGER := 1, A2 VARCHAR DEFAULT 'AAA') IS SELECT * FROM T1 WHERE I1 = A1 AND
I2 = A2;
  V1 INTEGER;
  V2 VARCHAR(10);
BEGIN
  OPEN C1( 1 );
  FETCH C1 INTO V1, V2;
  DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 || ' V2 = ' || V2 );
  CLOSE C1;
END;
/
V1 = 1 V2 = AAA
Anonymous PL block executed.
```

Compatibility

It is not defined in the SQL standard.

For More Information

Refer to the followings.

- CLOSE Statement
- FETCH Statement

28.29 OPEN FOR Statement

Function

It opens a single cursor by executing SELECT statement through a cursor variable defined in PSM.

Syntax

```
<open statement> ::=
    OPEN cursor_variable_name FOR <cursor_query>
    ;
<cursor_query> ::==
      <static_cursor_query>
    ¦ <dynamic_cursor_query> [ <using_clause> ]
<static_cursor_query> ::==
      select_statement
    ¦ select_for_update_statement
    ¦ insert_returning_query_statement
    ¦ update_returning_query_statement
    ¦ delete_returning_query_statement
<dynamic_cursor_query>
      single_quote_string
    ¦ psm_variable
<using_clause> ::=
    USING [ <bind_type> ] <expression> [ {, [ <bind_type> ] <expression>} ... ]
<bind_type> ::=
     IN
   ¦ OUT
   ¦ IN OUT
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the body section of a PL block.

Syntax Rules and Parameters

- 〈cursor_variable_name〉
  - It is the name of cursor variable.
- 〈cursor_query〉
  - A cursor query can use both a static cursor query and a dynamic cursor query.
  - The followings are cursor queries.
    - select statement
    - select for update statement
    - insert returning statement
    - update returning statement
    - delete returning statement
  - A dynamic cursor query is used by enclosing sql with a single quoted (') or by storing sql in a psm variable.
  - A using clause is available when writing a dynamic cursor query.
- 〈using_clause〉
  - If a bind variable exists when writing a dynamic cursor query, list variables or expressions in using clause as many as the number of bind variables.
  - The variable in using clause can describe the bind type of IN/ OUT/ IN OUT. The bind type of whe n writing the dynamic cursor query should be same as the bind type of the bind variable.
  - The bind type of the variable in using clause can be omitted. If omitted, the default bind type is I N.

Description

It executes the cursor query of the defined cursor variable.
If the cursor variable is previously open, then that cursor is executed after it is automatically closed and re opened.

Examples

```
gSQL> CREATE TABLE t1( c1 INTEGER, c2 VARCHAR( 6 ) );
Table created.
gSQL> COMMIT;
Commit complete.
```

- Insert Returning Statement

```
gSQL>
DECLARE
   var1 t1%ROWTYPE;
   curvar1 SYS_REFCURSOR;
BEGIN
   OPEN curvar1 FOR INSERT INTO t1 VALUES ( 1 , 'aaa' ) RETURNING *;
   FETCH curvar1 INTO var1;
   DBMS_OUTPUT.PUT_LINE( 'var1.c1 = ' || var1.c1 || ' , var1.c2 = ' || var1.c2 );
   CLOSE curvar1;
END;
/
var1.c1 = 1 , var1.c2 = aaa
Anonymous PL block executed.
```

- Select Statement

```
gSQL>
DECLARE
   var1 t1%ROWTYPE;
   curvar1 SYS_REFCURSOR;
BEGIN
   OPEN curvar1 FOR SELECT * FROM t1;

   FETCH curvar1 INTO var1;
   DBMS_OUTPUT.PUT_LINE( 'var1.c1 = ' || var1.c1 || ' , var1.c2 = ' || var1.c2 );
   CLOSE curvar1;
END;
/
var1.c1 = 1 , var1.c2 = aaa
Anonymous PL block executed.
```

- Update Returning Statement

```
gSQL>
DECLARE
   var1 t1%ROWTYPE;
   curvar1 SYS_REFCURSOR;
BEGIN
   OPEN curvar1 FOR UPDATE t1 SET c1 = 3, c2 = 'ccc' RETURNING *;

   FETCH curvar1 INTO var1;
```

```
   DBMS_OUTPUT.PUT_LINE( 'var1.c1 = ' || var1.c1 || ' , var1.c2 = ' || var1.c2 );
   CLOSE curvar1;
END;
/
var1.c1 = 3 , var1.c2 = ccc
Anonymous PL block executed.
```

- Delete Returning Statement

```
gSQL>
DECLARE
   var1 t1%ROWTYPE;
   curvar1 SYS_REFCURSOR;
BEGIN
   OPEN curvar1 FOR DELETE FROM t1 RETURNING *;
   FETCH curvar1 INTO var1;
   DBMS_OUTPUT.PUT_LINE( 'var1.c1 = ' || var1.c1 || ' , var1.c2 = ' || var1.c2 );
   CLOSE curvar1;
END;
/
var1.c1 = 3 , var1.c2 = ccc
Anonymous PL block executed.
```

- Dynamic Cursor Query and Using Clause

```
gSQL> INSERT INTO t1 VALUES( 100 , 'BBB' );
gSQL> COMMIT;
gSQL>
DECLARE
  v_int_in     INTEGER;
  v_varchar_in VARCHAR(6);
  v_out        t1%ROWTYPE;
  v_sql        VARCHAR(1024);
  cv           SYS_REFCURSOR;
BEGIN
  v_sql := 'SELECT * FROM t1 WHERE c1 = ? AND c2 = ?';
  v_int_in := 100;
  v_varchar_in := 'BBB';

  OPEN cv FOR v_sql USING IN v_int_in, v_varchar_in;

  FETCH cv INTO v_out;
```

```
  DBMS_OUTPUT.PUT_LINE( 'c1 : ' || v_out.c1 || ' , v_out.c2 : ' || v_out.c2 );

  CLOSE cv;
END;
/
c1 : 100 , v_out.c2 : BBB
Anonymous PL block executed.
```

Compatibility

It is not defined in the SQL standard.

For More Information

Refer to the followings.

- **Cursor Variable Declaration**
- **FETCH Statement**
- **CLOSE Statement**

28.30 Procedure Call

Function

It calls a user-defined procedure, a built-in procedure or a nested procedure.

Syntax

```
<Procedure call> ::=
    proc_name [ ( expr { , expr } ... ) ]
    ;
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the body section of a PL block.

- The following privileges for the corresponding procedure are required to call the user-defined procedure.
  - EXECUTE PROCEDURE
  - (EXECUTE PROCEDURE or CONTROL SCHEMA) ON SCHEMA for the schema to which the procedure belongs
  - EXECUTE ANY PROCEDURE ON DATABASE

Syntax Rules and Parameters

- Proc_name
  - It is a name of a procedure to be executed, and it is used in the following format.

Format	Syntax	Description
Single identifier	procedure_name	It calls the procedure of the given name. It is searched in the following order. 1. nested procedure

Format	Syntax	Description
		2. schema-level procedure
identifier chain	label_name.procedure_name	It calls a nested procedure.
	schema_name.procedure_name	It calls a schema-level procedure.
	package_name.procedure_name	It calls a built-in procedure.

If the given the number and type of arguments which were given in the procedure found first are wrong when searching with the given name (proc_name), then it does not search for another procedure but ca uses an error.

Description

It calls a user-defined procedure, a built-in procedure or a nested procedure which was defined an advan ce.
The argument value in which the default value is defined can be omitted when calling.

When using a procedure variable or bind parameter (?, :V1) in an argument which were defines as OUT o r IN-OUT, then the returned value is obtained.

Examples

```
gSQL> DECLARE
  PROCEDURE PROC1( A1 INTEGER )
    IS
    BEGIN
      PUT_LINE( 'A1 = ' || A1 );
    END;
BEGIN
  PROC1( 100 );
END;
/
A1 = 100
Anonymous PL block executed.
```

Compatibility

The SQL standard requires to use ⟨call statement⟩.

28.31 Procedure Declaration and Definition

Function

It declares and defines a procedure.

Syntax

```
<procedure declaration> ::=
      PROCEDURE <procedure name> [ ( <parameter list> ) ]
      ;
<procedure definition> ::=
      PROCEDURE <procedure name> [ ( <parameter list> ) ]
      { IS ¦ AS }
      <item declaration>
      BEGIN
      <pl statement list>
      END [ <procedure name> ]
      ;

<parameter list> ::=
      <parameter name> [ <parameter mode> ] <datatype> [ <parameter default> ] [ , ... ]
<parameter mode> ::=
      IN
    ¦ OUT
    ¦ IN OUT

<parameter default> ::=
      { := ¦ DEFAULT } <value expression>
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the declaration section of a PL block.

Syntax Rules and Parameters

procedure name

It is a name of procedure to be created in a PL block, and it should be a unique name in a PL block.
In other words, PL item and procedure declared in PL block can not have the same name.
The length of a procedure name should be shorter than 128 bytes.

parameter name

It defines a parameter name of the procedure.
The name of each parameter should be unique in a procedure.
In other words, the procedure's parameter and PL item can not have the same name.
The length of a parameter name should be shorter than 128 bytes.
The maximum number of parameters available in a single procedure is limitless.

parameter mode

It sets each parameter mode.
The parameter modes are IN, OUT, and IN OUT.
If the parameter mode is not specified, the default mode is *IN*.

parameter default

It is the default value of the parameter.
The parameter with the specified parameter default can be omitted when executing the procedure.
If the parameter is not specified but omitted, then the default value is <value expression> specified when defining the parameter.
The datatype of <value expression> should be the datatype of the parameter.
All parameters defined after the parameter having <parameter default> should have <parameter default>.

Item Declaration

It declares items such as a local variable to be used within a function.
It can declare all items which can be declared in a PL block.

PL Stmt List

It is a body section of a procedure, and it lists PL statements to be performed.

Description

It declares and defines the procedure as follows.

- PL block (e.g. anonymous block, procedure and function's block, block statement's block)
  - It is one of PL block's item and it declares and defines the procedure.
  - The function declared and defined in a PL block is a nested procedure.
  - Nested procedure is available only within the PL block range.
- Package specification
  - It declares the public procedure of the package.
  - The public procedure of the package is available in the database.
- Package body
  - It defines the public procedure declared in the package specification.
  - It declares and defines the private procedure of the package.
  - The private procedure of the package is available only within the package range.

The usage of the procedure is as same as that of schema-level procedure.

Examples

- Nested procedure

```
gSQL>
DECLARE
  PROCEDURE proc1( p1 INTEGER ) IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE( 'p1 = ' || p1 );
  END;
BEGIN
  proc1( 100 );
END;
/
p1 = 100
Anonymous PL block executed.
```

- Package procedure

```
gSQL>
CREATE OR REPLACE PACKAGE pkg1 AS
  -- Declare Package Public Function
  PROCEDURE proc1( p1 INTEGER );
END;
/

Package created.
gSQL>
CREATE OR REPLACE PACKAGE BODY pkg1 AS
  -- Declare Package Private Function
  PROCEDURE proc2( p1 INTEGER );

  -- Define Package Public Function
  PROCEDURE proc1( p1 INTEGER ) AS
  BEGIN
    DBMS_OUTPUT.PUT_LINE( 'p1 of proc1 : ' || p1 );

    proc2( p1 * p1 );
  END;

  -- Define Package Private Function
  PROCEDURE proc2( p1 INTEGER ) AS
  BEGIN
    DBMS_OUTPUT.PUT_LINE( 'p1 of proc2 : ' || p1 );
  END;
END;
/
Package created.
gSQL> CALL pkg1.proc1( 10 );
p1 of proc1 : 10
p1 of proc2 : 100
Procedure Call complete.
```

Compatibility

It is as same as a schema-level procedure.

For More Information

Refer to **CREATE PROCEDURE**.

28.32 RAISE Statement

Function

It explicitly generates a user-defined exception.

Syntax

```
〈RAISE Statement〉 ::=
    RAISE  〈Exception-Name〉
    ;
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the body section of a PL block.

Syntax Rules and Parameters

- A name of exception to be raised should be declared in a PL block to which a raise belongs or in DEC LARE clause of a superordinate PL block. However, a predefined exception can be raised without a de claration.
- If a raise statement is used within an exception handler, an exception name can be omitted, in this ca se, the previous exception is spread to the superordinate block.

Description

If it can not be processed in a PL block in which an exception occured, then it is spread to the superordina te PL block.
If an exception to be raised does not exist in a PL block including RAISE statement, nor does exist in all ex ception handlers within a superordinate PL block, then an error occurs.
It is spread from a PL block in which RAISE exception occurred to a superordinate PL block until it is proce

ssed, and it can not be spread to an exception handler of subordinate PL block.

Table 28-9 Propagating user exception

Raise exception	Exception handler SCOPE	Exception handler	Whether to spread it to superordinate
User exception without error code	Same scope	X	It spreads "unhandled exception" err or to a superordinate scope.
User exception with error code	Superordinate scope	X	It spreads a user exception.
User exception with error code	Same scope	X	Itspreads a user defined error code.
User exception with error code	Superordinate scope	X	Itspreads a user defined error code.

Examples

```
gSQL> DECLARE
V1 INTEGER;
exception1   EXCEPTION;
exception100 EXCEPTION;
BEGIN
    DBMS_OUTPUT.PUT_LINE('Step1');
    BEGIN
        RAISE exception1;
        DBMS_OUTPUT.PUT_LINE('Step2');
        EXCEPTION WHEN Exception100 THEN DBMS_OUTPUT.PUT_LINE('in Exception');
    END;
    DBMS_OUTPUT.PUT_LINE('Step3');
    EXCEPTION WHEN Exception1 THEN DBMS_OUTPUT.PUT_LINE('out Exception');
    DBMS_OUTPUT.PUT_LINE('Step4');
END;
/
Step1
out Exception
Step4
Anonymous PL block executed.
```

Compatibility

The SQL standard specifies <handler declaration> and <condition declaration>, but it does not support th e syntax.

For More Information

Refer to the followings.

- **Exception Handler**
- **Exception Declaration**

28.33 Record Variable Declaration

Function

It declares a record type variable in DECLARE section.

Syntax

```
<declare record variable> ::=
    variable_name <recordType>
    ;
<recordType> ::=
     <tableName>%ROWTYPE
   ¦ USER_DEFINED_DATA_TYPE
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the declaration section of a PL block.

Syntax Rules and Parameters

- Variable_name
  - It is a name of variable to be declared.
  - The length of the variable name should be shorter than 128 bytes.
  - It should be a unique name within a scope (PL block body).
  - A variable with the same name can be declared in a PL block body of nested subordinate.
  - It should be used in scope_name.variable_name form to correctly use the variables of duplicated declaration.
  - When using a variable of duplicated declaration without specifying the scope name, then a varia ble of the nearest scope is automatically used.
- Record_type
  - It can use %ROWTYPE and user-defined recordType.
  - For more information about the declaration of recordType, refer to **User Defined Record Type**.

Description

- The declared variable has the following features.
  - It should be a unique name within a scope (PL block body).
  - A variable with the same name can be declared in a PL block body of nested subordinate.
  - It should be used in scope_name.variable_name form to correctly use the variables of duplicated declaration.
  - When using a variable of duplicated declaration without specifying the scope name, then a variable of the nearest scope is automatically used.
- The declared variable is used in that scope and in its subordinate scope, but ':' is not added unlike an embedded SQL.
- The used variable is operated as a bind parameter (INOUT) for that SQL or a PSM control statement.

Examples

```
gSQL> DECLARE
  TYPE MY_REC1 IS RECORD ( F1 INTEGER, F2 VARCHAR(10) );
  V1 MY_REC1;
BEGIN
  V1.F1 := 1;
  V1.F2 := 'AAA';
  INSERT INTO T1 VALUES( V1.F1, V1.F2 );
END;
/
Anonymous PL block executed.
gSQL> COMMIT;
Commit complete.
gSQL> SELECT * FROM T1;
I1 I2
-- ---
 1 AAA
1 row selected.
```

Compatibility

It is not defined in the SQL standard.

28.34 RETURN Statement

Function

It specifies the value of which a function returns, then terminates the function. A procedure does not spe cify the return value, but terminates the procedure.

Syntax

```
<return statement> ::=
    RETURN [ return_value_expr ]
    ;
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the body section of a PL block.
It can not be used in the PL block of the table function.

Syntax Rules and Parameters

- Return_value_expr
  - It is an expression of a value to be returned, and it can be specified only when it is a function.

Description

It terminates a currently performing procedure/ function.
For a function, if RETURN statement is terminated without being performed, or if RETURN statement doe s not have return_value_expr, then an error occurs.
It can not be used in the table function.

Examples

```
gSQL> CREATE OR REPLACE FUNCTION FUNC1( A1 INTEGER, A2 INTEGER )
  RETURN INTEGER
  IS
    V1 INTEGER;
  BEGIN
    SELECT COUNT(*)
      INTO V1
      FROM T1
      WHERE T1.I1 >= A1 AND T1.I1 <= A2;
    RETURN V1;
  END;
  /
Function created.
```

Compatibility

It is specified in the SQL standard, but conformance rules do not exist.

28.35 RETURN TABLE Statement

Function

It returns the result set of executing the cursor variable's cursor query or the select statement, then termi nates the function.

Syntax

```
<return table statement> ::=
    RETURN TABLE ( { <cursor variable name> ¦ <select statement> } )
    ;
```

Invocation and Access Rules

It can be used only in the table function within PSM.
It can be used only in the body section of a table function block.

Syntax Rules and Parameters

- 〈cursor variable name〉
  - It returns the result set by executing the specified cursor variable's cursor query.
  - The cursor variable should be open.
  - The cursor variable's cursor query should have not been fetched.
  - It allows the cursor variable whose cursor query is the select statement only.
- 〈select statement〉
  - It can use the select statement only.
  - It can not use select for update statement and select into statement.

Description

It returns the result set of executing the cursor variable's cursor query or select statement specified in RET URN TABLE statement.
RETURN TABLE statement is available only in the table function.

Examples

```
gSQL> CREATE TABLE t_score( c_grade INTEGER, c_score INTEGER );

Table created.

gSQL> INSERT INTO t_score VALUES ( 1 , 98 ) , ( 1 , 97 ) , ( 1 , 99 ),
                                 ( 2 , 95 ) , ( 2 , 98 ) , ( 2 , 92 ),
                                 ( 3 , 98 ) , ( 3 , 96 ) , ( 3 , 94 );

9 rows created.

gSQL> COMMIT;

Commit complete.
```

- Example of using the cursor variable

```
gSQL>
CREATE FUNCTION func_cursor_variable( p_option VARCHAR )
  RETURN TABLE( f_class NUMBER, f_score NUMBER )
AS
  s_cur SYS_REFCURSOR;
BEGIN
  IF p_option = 'MIN' THEN
    OPEN s_cur FOR SELECT c_grade, MIN(c_score) FROM t_score GROUP BY c_grade;
  ELSIF p_option = 'MAX' THEN
    OPEN s_cur FOR SELECT c_grade, MAX(c_score) FROM t_score GROUP BY c_grade;
  ELSIF p_option = 'AVG' THEN
    OPEN s_cur FOR SELECT c_grade, AVG(c_score) FROM t_score GROUP BY c_grade;
  ELSE
    OPEN s_cur FOR SELECT NULL, NULL FROM dual;
  END IF;
```

```
  RETURN TABLE( s_cur );
END;
/

Function created.

gSQL> COMMIT;

Commit complete.

-- Calculate the lowest score in each grade.
gSQL> SELECT * FROM TABLE( func_cursor_variable( 'MIN' ) );

F_CLASS F_SCORE
------- -------
      1      97
      2      92
      3      94

3 rows selected.
```

- Example of using the select statement

```
gSQL>
CREATE FUNCTION func_select( p_option VARCHAR )
    RETURN TABLE( f_class NUMBER, f_score NUMBER )
AS
BEGIN
  IF p_option = 'MIN' THEN
    RETURN TABLE ( SELECT c_grade, MIN(c_score) FROM t_score GROUP BY c_grade );
  ELSIF p_option = 'MAX' THEN
    RETURN TABLE ( SELECT c_grade, MAX(c_score) FROM t_score GROUP BY c_grade );
  ELSIF p_option = 'AVG' THEN
    RETURN TABLE ( SELECT c_grade, AVG(c_score) FROM t_score GROUP BY c_grade );
  ELSE
    RETURN TABLE ( SELECT NULL, NULL FROM dual );
  END IF;
END;
/
```

```
Function created.

-- Calculate the average score in each grade.
gSQL> SELECT * FROM TABLE( func_select( 'AVG' ) );

F_CLASS F_SCORE
------- -------
      1      98
      2      95
      3      96

3 rows selected.
```

Compatibility

The SQL standard describes it in <return statement> statement.
However, the SQL standard does not define the cursor variable.

28.36 RETURNING INTO clause

Function

The data processed in an insert/ update/ delete is returned to a PSM variable.

Syntax

```
<Insert, Delete Returning_into_clause> ::=
      [ RETURN ¦ RETURNING ] { * ¦ { <value_expression> [ [AS] alias_name ] } [, ...] INTO
variable_name [, ...]
<Update returning into clause> ::=
    { RETURN ¦ RETURNING } [ NEW ¦ OLD ] { * ¦ { <value expression> [ [AS] alias_name] } [,
...] } INTO Variable [, ...]
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the body section of a PL block.

Syntax Rules and Parameters

A record type variable can not be used being mixed with other types.

Description

It stores before/ after record of processing an insert/ update/ delete statement through returning into.

Examples

```
gSQL> DECLARE
  rec t1%ROWTYPE;
BEGIN
    INSERT INTO t1 VALUES (1, 2, 3) RETURNING * INTO rec ;
    UPDATE T1 SET ROW = rec RETURNING * INTO rec;
    DELETE FROM T1 RETURNING * INTO rec;
END;
/
Anonymous PL block executed.
```

Compatibility

It is not defined in the SQL standard.

28.37 %ROWTYPE Attribute

Function

When declaring a variable, it defines the structure and type as same as those of a specific table, a specific cursor or a result set of a cursor variable.

Syntax

```
<rowtype attribute> ::=
    <identifier chain> % ROWTYPE
```

Invocation and Access Rules

- It can be used only within PSM. (e.g. package, procedure, function)
- It can be used only in the declaration section of a PL block.
- When declaring a variable, it can be used only in <data type> section.
- When declaring a record type field, then it can not be used in <data type> section. (It does not suppo rt complex data type.)

Syntax Rules and Parameters

- Identifier chains are as follows.
  - The name of table, view, synonym to be referenced, or the name of a cursor or a cursor variable

Description

- Search order of the reference targets
  - Cursor or cursor variable
  - Base table, view, or synonym
- Reference scope

- When referring by using a row attribute, only name and type of columns in a result set of a table or a cursor is referenced.
- Therefore, it does not refer to NOT NULL constraint or DEFAULT value settings.

Examples

```
gSQL> CREATE TABLE T1 ( I1 INTEGER NOT NULL, I2 VARCHAR(10) );
Table created.
gSQL> INSERT INTO T1 VALUES( 123, '1234567890' );
1 row created.
gSQL> COMMIT;
Commit complete.
gSQL> DECLARE
  V1 T1%ROWTYPE;
BEGIN
  SELECT * INTO V1.I1, V1.I2 FROM T1;
  DBMS_OUTPUT.PUT_LINE( 'V1.I1 = ' || V1.I1 || ' V1.I2 = ' || V1.I2 );
END;
/
V1.I1 = 123 V1.I2 = 1234567890
Anonymous PL block executed.
```

Compatibility

It is not defined in the SQL standard.

28.38 Scalar Variable Declaration

Function

It declares a scalar variable in the declaration section.

Syntax

```
<declare scalar variable> ::=
    variable_name <data type> [ <variable initialize clause> ]
    ;
<variable initialize clause> ::=
      [ NOT NULL ] { DEFAULT | := } <value expression>
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the declaration section of a PL block.

Syntax Rules and Parameters

- Variable_name
  - It is a name of variable to be declared.
  - The length of the variable name should be shorter than 128 bytes.
  - It should be a unique name within a scope (PL block body).
  - A variable with the same name can be declared in a PL block body of nested subordinate.
  - It should be used in scope_name.variable_name form to correctly use the variables of duplicated declaration.
  - When using a variable of duplicated declaration without specifying the scope name, then a variable of the nearest scope is automatically used.
- Data_Type
  - It can use all built-in **Data Type** provided in GOLDILOCKS.
- Value_expression
  - It expresses the initial value to specify in the variable.

    - It can use all constants and expressions supported by GOLDILOCKS except for multi-row function s.

Description

- The declared variable has the following features.
    - It should be a unique name within a scope (PL block body).
    - A variable with the same name can be declared in a PL block body of nested subordinate.
    - It should be used in scope_name.variable_name form to correctly use the variables of duplicated declaration.
    - When using a variable of duplicated declaration without specifying the scope name, then a varia ble of the nearest scope is automatically used.
- The declared variable is used in that scope and in its subordinate scope, but ':' is not added unlike an embedded SQL.
- The used variable is operated as a bind parameter (INOUT) for that SQL or a PSM control statement.

Examples

```
gSQL> CREATE TABLE T1 ( I1 INTEGER, I2 VARCHAR(10) );
Table created.
gSQL> COMMIT;
Commit complete.
gSQL> DECLARE
V1 INTEGER := 100;
V2 INTEGER := -100;
V3 VARCHAR(10) := 'ABC';
BEGIN
  IF V1 > 50 THEN
    INSERT INTO T1 VALUES ( V1, V3 );
  ELSE
    INSERT INTO T1 VALUES ( V2, V3 );
  END IF;
END;
/
Anonymous PL block executed.
gSQL> SELECT * FROM T1;
 I1 I2
--- ---
```

```
100 ABC
1 row selected.
```

Compatibility

- The differences between <declare scalar variable> statement of GOLDILOCKS and that of SQL standard are as follows.
  - <SQL variable declaration> of the SQL standard declares a variable in a PL block body (after BEGIN), but GOLDILOCKS declares a variable in a separate declaration section.
  - The SQL standard can declare multiple variables of the same type by using a single DECLARE statement, but GOLDILOCKS can declare only a single variable by using a single statement.
  - The SQL standard uses only DEFAULT syntax when setting the initial value, but GOLDILOCKS can use an assign sign (:=).

28.39 SELECT INTO Statement

Function

A single row is returned through SELECT.

Syntax

```
<select statement: single row> ::=
    SELECT [ <hint clause> ] [ <set quantifier> ] <select list>
        INTO <select target list>
        <table expression>
    ;
<select target list> ::=
    variable_name [, ...]
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the body section of a PL block.

Syntax Rules and Parameters

The rules are as same as those of a select statement except the rules for INTO clause.

Description

- SELECT INTO is used to return a single record.
  - If the number of results is zero, "NO_DATA_FOUND" exception occurs.
  - If the number of results are two or more, "TOO_MANY_ROWS" exception occurs.
- The result can be returned through a record type variable of PSM.
  - When using a record type variable, it can not be used being mixed with other type variables

Examples

```
gSQL> CREATE TABLE T1 (c1 VARCHAR(20), c2 VARCHAR(20));
Table created.
gSQL> INSERT INTO T1 VALUES ('AAA', 'BBB'), ('BBB', 'CCC');
2 rows created.
gSQL> DECLARE
  v1 VARCHAR(20);
  v2 VARCHAR(20);
BEGIN
  SELECT * INTO v1, v2 FROM T1 WHERE c1 = 'AAA';
  DBMS_OUTPUT.PUT_LINE('V1=' || v1 || ', v2=' || v2);
END;
/
V1=AAA, v2=BBB
Anonymous PL block executed.
```

Compatibility

It is not defined in the SQL standard.

28.40 SQLCODE Function

Function

It returns an error code of a statement which was performed just before in PSM.

Syntax

```
<SQLCODE function> ::= SQLCODE
```

Invocation and Access Rules

It can be used only within PSM.

Syntax Rules and Parameters

It does not have a separate argument.

Description

It returns an error code of a statement performed in PL/ SQL.
A user-defined exception which does not assign an error code returns to 1 at the time when it is processe d by a handler.
When the error is completely processed by an exception handler, it returns to 0.

Examples

```
gSQL> DECLARE
V1 INTEGER;
BEGIN
```

```
    DBMS_OUTPUT.PUT_LINE('SQLCODE=[' || SQLCODE || ']');
    DBMS_OUTPUT.PUT_LINE('SQLERRM=[' || SQLERRM || ']');
END;
/
SQLCODE=[0]
SQLERRM=[[SUNJESOFT][PL/SQL][GOLDILOCKS]successful completion]
Anonymous PL block executed.
```

Compatibility

It is not defined in the SQL standard.

28.41 SQLERRM Function

Function

It returns an error message of a statement which was performed just before in PSM.

Syntax

```
<SQLERRM function> ::= SQLERRM
```

Invocation and Access Rules

It can be used only within PSM.

Syntax Rules and Parameters

It does not have a separate argument.

Description

It returns an error message of a statement performed in PL/ SQL.
A user-defined exception which does not assign an error code returns to a user-defined exception at the time when it is processed by a handler.
When the error is completely processed by an exception handler, it outputs successful completion message.

Examples

```
gSQL> DECLARE
V1 INTEGER;
```

```
BEGIN
    DBMS_OUTPUT.PUT_LINE('SQLCODE=[' || SQLCODE || ']');
    DBMS_OUTPUT.PUT_LINE('SQLERRM=[' || SQLERRM || ']');
END;
/
SQLCODE=[0]
SQLERRM=[[SUNJESOFT][PL/SQL][GOLDILOCKS]successful completion]
Anonymous PL block executed.
```

Compatibility

It is not defined in the SQL standard.

28.42 %TYPE Attribute

Function

When declaring a variable or defining a specific field of RECORD type, it defines the type as same as the column of a specific table or another variable.

Syntax

```
<type attribute> ::=
    <identifier chain> % TYPE
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the declaration section of a PL block.
It can be used only for <data type> section when declaring a variable or a field of a record type.

Syntax Rules and Parameters

- Identifier_chain
  - It is the name of a table column to be referenced or of the existing declared variable (or a field of the variable).

Description

Reference scope

The reference scope according to the referenced object types are as follows.

- When the referenced object is a column of a table
  - It refers to the data type of the column.

    - It does not refer to a constraint (e.g. NOT NULL) of the column.
    - It does not refer to the default initial value of the column.
- When the referenced object is another variable (or a field of a variable)
    - It refers to the data type of the variable (or a field).
    - It refers to a constraint (NOT NULL) of the variable (or a field).
    - It does not refer to the default initial value of the variable (or a field).
- Search order of the referenced target object is as follows.
  1. The name of a variable (or a field)
  2. The name of a column

NOT NULL Constraints Variables References

It does not refer to the initial value when referring to NOT NULL attribute variable, so a new initial value s hould be specified.
Setting an initial value of a field is not supported when using a type attribute for the field of a current rec ord type variable, so NOT NULL type field can not be referenced.

Examples

```
gSQL> DECLARE
  V1 NUMBER(5,2) := 100.01;
  V2 V1%TYPE;
BEGIN
  DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
  DBMS_OUTPUT.PUT_LINE( 'V2 = ' || V2 );
END;
/
V1 = 100.01
V2 =
Anonymous PL block executed.
```

Compatibility

It is not defined in the SQL standard.

28.43 UPDATE Statement Extension

Function

A feature altering a record by using a record type variable is added other than a feature consecutively listi ng the altering target columns of UNDATE statement in PSM.
It stores the result by using a record type variable in UPDATE (searched) RETURNING INTO clause in PSM.

Syntax

```
<update Extension statement : searched> ::=
    UPDATE table_name [ [ AS ] alias_name ]
        <target-list>
        [ WHERE <search condition> ]
        [ <result offset clause> ]
        [ <fetch limit clause> ]
        [ <returning into clause> ]
    ;
<update statement: positioned> ::=
    UPDATE table_name [ [ AS ] alias_name ]
        <Target-List>
        WHERE CURRENT OF cursor_name
    ;
<result offset clause> ::=
    OFFSET skip_count [ ROW ¦ ROWS ]
<fetch limit clause> ::=
      <fetch first clause>
    ¦ <limit clause>
<fetch first clause> ::=
    FETCH [ FIRST ¦ NEXT ] [ row_count ] [ ROW ONLY ¦ ROWS ONLY ]
<limit clause>
    LIMIT { fetch_row_count ¦ offset_row_count, fetch_row_count ¦ ALL }
<returning into clause> ::=
    { RETURN ¦ RETURNING } [ NEW ¦ OLD ] { * ¦ { <value expression> [ [AS] alias_name] } [,
...] } INTO Variable [, ...]
<target-List> ::=
        SET <set clause> [, ...]
```

```
      ¦ SET ROW = <psm_variable>
<set clause> ::=
      column_name = { <value expression> ¦ DEFAULT }
    ¦ ( column_name [, ...] ) = ( { <value expression> ¦ DEFAULT } [, ...] )
    ¦ ( column_name [, ...] ) = ( <query expression> )
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the body section of a PL block.

Syntax Rules and Parameters

- <PSM_Variable> to be used in UPDATE SET ROW syntax shoould be a variable declared as a record ty pe.
- <Variable> to be used in UPDATE RETURNING INTO syntax does not need to be a record type.
  - However, if a record type is specified, then it can not be used being mixed with other data type v ariables nor can two or more record type variables be listed.

Description

It alters the record or stores the result of RETURNING INTO through a record type variable in PSM.

Examples

```
gSQL> CREATE TABLE T1 (C1 VARCHAR(20), C2 VARCHAR(20));
Table created.
gSQL> INSERT INTO T1 VALUES ('AAA', 'BBB'), ('BBB', 'CCC'), ('CCC', 'DDD');
3 rows created.
gSQL> DECLARE
    v1 t1%ROWTYPE;
    v2 t1%ROWTYPE;
BEGIN
    v1.c1 := '1';
    v1.c2 := '2';
```

```
    UPDATE T1 SET ROW = v1 WHERE c1 = 'AAA' RETURNING * INTO v2;
    DBMS_OUTPUT.PUT_LINE('SQL%ROWCOUNT=' || SQL%ROWCOUNT );
    DBMS_OUTPUT.PUT_LINE('v2.c1=' || v2.c1 || ', v2.c2=' || v2.c2);
END;
/
SQL%ROWCOUNT=1
v2.c1=1, v2.c2=2
Anonymous PL block executed.
gSQL> SELECT * FROM T1 ORDER BY C1;
C1  C2
--- ---
1   2
BBB CCC
CCC DDD
3 rows selected.
```

Compatibility

It is not defined in the SQL standard.

28.44 WHILE LOOP Statement

Function

It performs internal statements during <search condition> returns TRUE value.

Syntax

```
<while loop statement> ::=
    WHILE <search condition>
    LOOP { <SQL procedure statement> ; }... END LOOP [ loop_name ]
    ;
```

Invocation and Access Rules

It can be used only within PSM. (e.g. package, procedure, function)
It can be used only in the body section of a PL block.

Syntax Rules and Parameters

- Search conditions are as follows.
  - It is a conditional expression which continues to circle a while loop.
  - It should finally return a boolean type.

Description

while loop statement performs an internal statement list as long as the evaluation result of <search condit ion> is TRUE.

Examples

```
gSQL> DECLARE
V1 integer := 0;
BEGIN
  WHILE V1 < 10 LOOP
    DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
    V1 := V1 + 1;
  END LOOP;
END;
/
V1 = 0
V1 = 1
V1 = 2
V1 = 3
V1 = 4
V1 = 5
V1 = 6
V1 = 7
V1 = 8
V1 = 9
Anonymous PL block executed.
```

Compatibility

<while loop statement> statement is defined as <while statement> in the SQL standard.
<while statement> of the SQL standard performs loop statement as DO ... END WHILE, but GOLDILOCKS performs it as LOOP ... END LOOP.

For More Information

Refer to the followings.

- **CONTINUE Statement**
- **EXIT Statement**
- **GOTO Statement**

29.

PSM SQL References

29.1 ALTER FUNCTION

Function

It recompiles a function.

Syntax

```
<alter function statement> ::=
    ALTER FUNCTION function_name COMPILE
    ;
```

Invocation and Access Rules

One of the following privilege is required to perform <alter function statement>.

- The owner of that function
- (ALTER PROCEDURE or CONTROL SCHEMA) ON SCHEMA for the schema to which the function belo ngs
- ALTER ANY PROCEDURE ON DATABASE

Syntax Rules and Parameters

- function Name
  - It is the function name to be compiled.
  - It can define the schema to which the function belongs, such as schema_name.func_name. If sch ema_name is omitted, the default schema name of the user performing the statement is used.

Description

It recompiles a specified schema-level function.

Examples

```
gSQL> ALTER FUNCTION FUNC1 COMPILE;
Function altered.
```

Compatibility

It is a statement altering characteristics specified when performing CREATE in the SQL standard, and asta tement recreating a plan of a function in GOLDILOCKS.

Table 29-1 SQL standard compatibility

Feature ID	Description	Compatibility
F381	Extended schema manipulation	O

For More Information

Refer to the followings.

- CREATE FUNCTION
- DROP FUNCTION

29.2 ALTER PACKAGE

Function

It recompiles a package.

Syntax

```
<alter package statement> ::=
    ALTER PACKAGE package_name <package compile clause>
    ;
<package compile clause> ::=
    COMPILE [PACKAGE¦SPECIFICATION¦BODY]
```

Invocation and Access Rules

One of the following privilege is required to perform <alter package statement>.

- The owner of that package
- (ALTER PACKAGE or CONTROL SCHEMA) ON SCHEMA for the schema to which the package belong s
- ALTER ANY PACKAGE ON DATABASE

Syntax Rules and Parameters

- Package name
  - It is a name of package to be compiled.
  - It can define the schema to which the package belongs, such as schema_name.package_name. If schema_name is omitted, the default schema name of the user performing the statement is used.

- Package compile clause
  - PACKAGE: It recompiles both the specification and the body. (Default)
  - SPECIFICATION: It recompiles only the specification.
  - BODY: It recompiles only the body.

Description

It recompiles the specified package.
The execution code of the compiled package is stored in the plan cache.

Examples

```
ALTER PACKAGE PKG1 COMPILE;
Package altered.
```

```
ALTER PACKAGE PKG1 COMPILE PACKAGE;
Package altered.
```

```
ALTER PACKAGE PKG1 COMPILE BODY;
Package altered.
```

Compatibility

It is ALTER MODULE statement in the SQL standard.

For More Information

Refer to the followings.

- CREATE PACKAGE
- CREATE PACKAGE BODY
- DROP PACKAGE

29.3 ALTER PROCEDURE

Function

It recompiles a procedure.

Syntax

```
<alter procedure statement> ::=
    ALTER PROCEDURE proc_name COMPILE
    ;
```

Invocation and Access Rules

One of the following privilege is required to perform <alter procedure statement>.

- The owner of that procedure
- (ALTER PROCEDURE or CONTROL SCHEMA) ON SCHEMA for the schema to which the procedure bel ongs
- ALTER ANY PROCEDURE ON DATABASE

Syntax Rules and Parameters

- Proc Name
  - It is the procedure name to be compiled.
  - It can define the schema to which the procedure belongs, such as schema_name.proc_name. If s chema_name is omitted, the default schema name of the user performing the statement is used.

Description

It recompiles a specified schema-level procedure.

Examples

```
gSQL> CREATE OR REPLACE PROCEDURE PROC1( A1 INTEGER )
IS
BEGIN
  INSERT INTO T1 VALUES( A1 );
END;
/
ERR-01000(16409): Warning: Routine definition has compilation errors
ERR-HY000(17032): PSM compilation error :
(1) at (5:15): ERR-17053: schema or table object does not exist
Procedure created.
gSQL> CALL PROC1(1);
ERR-HY000(17032): PSM compilation error :
(1) at (5:15): ERR-17053: schema or table object does not exist
gSQL> CREATE TABLE T1( I1 INTEGER );
Table created.
gSQL> COMMIT;
Commit complete.
gSQL> ALTER PROCEDURE PROC1 COMPILE;
Procedure altered.
gSQL> COMMIT;
Commit complete.
gSQL> CALL PROC1(2);
Procedure Call complete.
gSQL> SELECT * FROM T1;
I1
--
 2
1 row selected.
```

Compatibility

It is a statement altering characteristics specified when performing CREATE in the SQL standard, and asta tement recreating a plan of a procedure in GOLDILOCKS.

Table 29-2 SQL standard compatibility

Feature ID	Description	Compatibility
F381	Extended schema manipulation	O

For More Information

Refer to the followings.

- CREATE PROCEDURE
- DROP PROCEDURE

29.4 CALL Statement

Function

It performs a schema-level procedure or a function.

Syntax

```
<call statement> ::=
    <sql call statement> ¦ <odbc procedure call escape sequence>
    ;
<sql call statement> ::=
    CALL proc_name [ ( value_expr [ , value_expr ] .. ) ]  [ INTO { '?' ¦ { host_param [
indicator_param ] } } ]
<odbc procedure call escape sequence> ::=
    '{' [ ? = ] CALL proc_name [ ( value_expr [ , value_expr ] .. ) ] '}'
```

Invocation and Access Rules

One of the following privilege is required to perform <call statement>.

- The EXECUTE privilege for that procedure
- (EXECUTE PROCEDURE or CONTROL SCHEMA) ON SCHEMA for the schema to which the procedure belongs
- EXECUTE ANY PROCEDURE ON DATABASE

Syntax Rules and Parameters

- proc_name
  - It is a name of a procedure/ function to be executed.
  - It may includes a schema to whichthe procedure belongs such as Schema_name.Proc_name.
- value_expr
  - It expresses an argument value which were transferred to the procedure. It can use a bind param eter such as '?' or ':V1'.

Description

It executes a schema-level SQL procedure or a function by using specified arguments.

A function of <sql call statement> form returns the result value by using a host variable expression or a dy namic bind parameter (?) after INTO clause.

<odbc procedure call escape sequence> form is a standard statement to call PROCEDURE in ODBC/ JDBC, and GOLDILOCKS supports this statement in a server. (It can also be used in a tool such as gsql.) A functi on returns the result value by using assign expressions (? =) at the front.

Examples

Call Procedure

```
gSQL> CREATE OR REPLACE PROCEDURE PROC1
(
  A1 INTEGER
)
IS
BEGIN
  DBMS_OUTPUT.PUT_LINE('A1=' || A1);
END;
/
Procedure created.
gSQL> \var v1 INTEGER;
gSQL> \exec :v1 := 123;
gSQL> CALL PROC1(:v1);
A1=123
Procedure Call complete.
```

Call Function

```
CREATE OR REPLACE FUNCTION FUNC1
(
  A1 INTEGER
)
RETURN INTEGER
```

```
IS
BEGIN
    return A1;
END;
/
Function created.
gSQL> \var v1 INTEGER;
gSQL> \var v2 INTEGER;
gSQL> \exec :v1 := 123;
gSQL> CALL FUNC1(:v1) INTO :v2;
Procedure Call complete.
gSQL> \print v2;
 V2
---
123
```

Compatibility

The SQL standard allows only the call for a PROCEDURE, so it does not define below [INTO] clause.

29.5 CREATE FUNCTION

Function

It defines a schema-level function.

Syntax

```
<create function statement> ::=
    CREATE [ OR REPLACE ] FUNCTION <function name> [ ( <parameter list> ) ]
       <return clause>
       [ <function characteristics> ]
       { IS ¦ AS }
       <item declaration>
       BEGIN
       <pl statement list>
       END [ <function name> ]
    ;
<parameter list> ::=
      <parameter name> [ <parameter mode> ] <datatype> [ <parameter default> ] [ , ... ]

<parameter mode> ::=
      IN
    ¦ OUT
    ¦ IN OUT

<parameter default> ::=
      { := ¦ DEFAULT } <value expression>

<return clause> ::=
      RETURN <datatype>
    ¦ RETURN TABLE ( <table function column list> )

<table function column list> ::=
      <column name> <datatype> [ , ... ]

<function characteristics> ::=
```

```
    DETERMINISTIC
  ¦ AUTHID CURRENT_USER
  ¦ AUTHID DEFINER
```

Invocation and Access Rules

The user should satisfy the following conditions to perform <create function statement>.

- One of the following privileges is required to create a function.
  - (CREATE PROCEDURE or CONTROL SCHEMA) ON SCHEMA for the schema to which the functio n belongs
  - CREATE ANY PROCEDURE ON DATABASE
- If a function already exists when using OR REPLACE clause, then one of the following privileges drop ping the existing function is required.
  - The owner of that function
  - (DROP PROCEDURE or CONTROL SCHEMA) ON SCHEMA for the schema to which the function b elongs
  - DROP ANY PROCEDURE ON DATABASE
- The user who performed the statement becomes the owner of the created function.

Syntax Rules and Parameters

OR REPLACE

It replaces an existing function with a new function when the function already exists.

function name

It is a name of function to be created, and it should be a unique name in a schema.
It can define the schema to which the function belongs, such as schema_name.function_name. If schema _name is omitted, the default schema name of the user performing the statement is used.
The length of a function name should be shorter than 128 bytes.

parameter name

It defines a parameter name of the function.
The name of each parameter should be unique in a function.
In other words, the function's parameter and PL item can not have the same name.

The length of a parameter name should be shorter than 128 bytes.
The maximum number of parameters available in a single function is limitless.

parameter mode

It sets each parameter mode.
The parameter modes are IN, OUT, and IN OUT.
If the parameter mode is not specified, the default mode is *IN*.

parameter default

It is the default value of the parameter.
The parameter with the specified parameter default can be omitted when executing the function.
If the parameter is not specified but omitted, then the default value is <value expression> specified when defining the parameter.
The datatype of <value expression> should be the datatype of the parameter.
All parameters defined after the parameter having <parameter default> should have <parameter default>.

return clause

It defines the return type of the function.
It is defined as follows in <return clause>.

- RETURN <datatype>
  - It defines the datatype of the return value returned by the function.
- RETURN TABLE (<table function column list>)
  - It defines the table type of the returned result set.

table function column list

It is the column name of the result set returned by the table function.
The length of a column name should be shorter than 128 bytes.
The number of columns are limitless.
Each column name is unique in <table function column list>.
The column name can be as same as the parameter name and the declare item name.
The column defined in <table function column list> can not be referenced in PL block of the function.

function characteristics

It defines options to perform a function. It should be specified only once per a single item.

- DETERMINISTIC: The function always returns the same result value when the same argument values a re input.
- AUTHID CURRENT_USER: SQLs which are being performed during performing a function are interpre ted and performed according to an authorization of an invoker.
- AUTHID DEFINER: SQLs which are being performed during performing a function are interpreted and performed according to an authorization of a definer.

Item Declaration

It declares items such as a local variable to be used within a function.
It can declare all items which can be declared in a PL block.

PL Stmt List

It is a body section of a function, and it lists PL statements to be performed.
It can not use a bind parameter such as '?' or ':V1'. within a function.

Description

It defines a schema-level SQL function. The created fucntion can be called from all expressions.

The definition of a function can be viewed in ROUTINES table of INFORMATION_SCHEMA. The definition of a function parameter can be viewed in PARAMETERS table of INFORMATION_SCHEMA.

If a function becomes temporarily unstable due to absences of related objects, then it can try to recreate a plan by using **ALTER FUNCTION** statement.

The created function can be dropped by using **DROP FUNCTION** statement.
The maximum number of functions to be created is not limited. Therefore, they can be created as many a s the storage space is available.

Examples

```
gSQL> CREATE OR REPLACE FUNCTION FUNC1( A1 INTEGER, A2 INTEGER )
  RETURN INTEGER
  IS
    V1 INTEGER;
  BEGIN
    SELECT COUNT(*)
      INTO V1
      FROM T1
      WHERE T1.I1 >= A1 AND T1.I1 <= A2;
    RETURN V1;
  END;
  /
Function created.
```

Compatibility

The SQL standard does not define OR REPLACE clause.

Table 29-3 SQL standard compatibility

Feature ID	Description	Compatibility
T471	Result sets return value	X
T341	Overloading of SQL-invoked functions and SQL-invoked procedures	X
S023	Basic structured types	X
S241	Transform functions	X
S024	Enhanced structured types	X
T571	Array-returning external SQL-invoked functions	X
T572	Multiset-returning external SQL-invoked functions	X
S201	SQL routines on arrays	X
S202	SQL-invoked routines on multisets	X
T323	Explicit security for external routines	X
S231	Structured type locators	X
S232	Array locators	X
S233	Multiset locators	X
T041	Basic LOB data type support	X
S027	Create method by specific method name	X
T041	Basic LOB data type support	X

Feature ID	Description	Compatibility
T324	Explicit security for SQL routines	O
T326	Table functions	O
T651	SQL-schema statements in SQL routines	X
T652	SQL-dynamic statements in SQL routines	O
T653	SQL-schema statements in external routines	X
T654	SQL-dynamic statements in external routines	X
T655	Cyclically dependent routines	X
T272	Enhanced savepoint management	X
T522	Default values for IN parameters of SQL-invoked procedures	O
B121	Routine language Ada	X
B122	Routine language C	X
B123	Routine language COBOL	X
B124	Routine language Fortran	X
B125	Routine language MUMPS	X
B126	Routine language Pascal	X
B127	Routine language PL/I	X
B128	Routine language SQL	O
B129	Routine language Ada: VARCHAR and NUMERIC support	X

For More Information

Refer to the followings.

- DROP FUNCTION
- ALTER FUNCTION

29.6 CREATE PACKAGE

Function

It defines the spec about public items to be used in a package.

Syntax

```
<create package statement> ::=
    CREATE [ OR REPLACE ]
        PACKAGE package_name
        [ <package_characteristics> ]
        { IS ¦ AS }
       <item_declaration>
    END
    ;
<package_characteristics> ::=
    AUTHID CURRENT_USER ¦ AUTHID DEFINER
<item_declaration> ::=    <variable declaration>
                        ¦ <cursor declaration>
                        ¦ <user-defined type declaration>
                        ¦ <function declaration>
                        ¦ <procedure declaration>
                        ¦ <user exception declaration>
                        ¦ <cursor definition>
```

Invocation and Access Rules

The user should satisfy the following conditions to perform <create package statement>.

- One of the following privileges is required to create a package.
  - (CREATE PACKAGE or CONTROL SCHEMA) ON SCHEMA for the schema to which the package b elongs
  - CREATE ANY PACKAGE ON DATABASE
- If a package already exists when using OR REPLACE clause, then one of the following privileges drop ping the existing package is required.

    - The owner of that package
    - (DROP PACKAGE or CONTROL SCHEMA) ON SCHEMA for the schema to which the package bel ongs
- The user who performed the statement becomes the owner of the created package.

Syntax Rules and Parameters

OR REPLACE

It replaces an existing package specification when the package already exists.

PACKAGE NAME

It is a name of package to be created, and it should be a unique name in a schema.
It can define the schema to which the package belongs, such as schema_name.package_name. If schema _name is omitted, the default schema name of the user performing the statement is used.
The length of a package name should be shorter than 128 bytes.

Package Characteristics

It defines options to perform a package. It should be specified only once per a single item.

- AUTHID CURRENT_USER: SQLs which are being performed during performing items in a package are interpreted and performed according to an authorization of an invoker.
- AUTHID DEFINER: SQLs which are being performed during performing items in a package are interpr eted and performed according to an authorization of a definer.

Item Declaration

It declares a public variable, a type cursor, a cursor variable, a function spec, and a procedure spec which are accessible from out of the package.
A function and a procedure in the package should be defined through *create package body* statement.
The cursor which was declared in a package without SQL should be defined through *create package bod y* statement.
Also, a function, a procedure, an argument of a cursor and the returning type in a package should be as s ame as those defined in a body statement.

Description

It creates the schema-level package specification.
All public items in the created package can be referred by another procedure/ function/ package/ anony mous block.

The information about package creation can be viewed in MODULES table in DEFINITION_SCHEMA or IN FORMATION_SCHEMA.
The list of each public procedure/ function in the package can be viewed in ROUTINES table in DEFINITIO N_SCHEMA or INFORMATION_SCHEMA.
The definition about parameters of each public procedure/ function in the package can be viewed in PAR AMETERS table in DEFINITION_SCHEMA or INFORMATION_SCHEMA.
Private procedure/ function can be viewed in MODULE_BODY table.
If the information about objects referred by a procedure/ function/ variable in the package are temporaril y unstable, try to recompile it by using <alter package> statement.

Examples

```
CREATE OR REPLACE PACKAGE PKG1
IS
  V1 INTEGER;
  PROCEDURE PROC1;
  FUNCTION FUNC1 RETURN INTEGER;
END;
/
Package created.
```

Compatibility

It is CREATE MODULE statement in the SQL standard.

For More Information

Refer to the followings,

- CREATE PACKAGE BODY

- DROP PACKAGE
- ALTER PACKAGE

29.7 CREATE PACKAGE BODY

Function

It creates the definition about procedure/ function/ cursors to be used in the package.

Syntax

```
<create package body statement> ::=
    CREATE [ OR REPLACE ] PACKAGE BODY package_name
    { IS ¦ AS }
       <item_declaration>
       <cursor_definition>
       <routine_definition>
    [ BEGIN <initialization part> ]
    END
    ;
<item_declaration> ::=    VARIABLE_DECLARATION
                        ¦ CURSOR_DECLARATION
                        ¦ USER_DEFINED_TYPE_DEFINITION
                        ¦ FUNCTION_DECLARATION
                        ¦ PROCEDURE_DECLARATION
                        ¦ USER_EXCEPTION_DECLARATION
<routine_definition> ::= FUNCTION_DEFINITION
                       ¦ PROCEDURE_DEFINITION
<initialization part> ::= <pl_stmt_list>
```

Invocation and Access Rules

The user should satisfy the following conditions to perform <create package body statement>.

- One of the following privileges is required to create a package body.
  - (CREATE PACKAGE or CONTROL SCHEMA) ON SCHEMA for the schema to which the package b elongs
  - CREATE ANY PACKAGE ON DATABASE
- The package spec should be defined in advance.

- If a package body already exists when using OR REPLACE clause, then one of the following privileges dropping the existing package is required.
  - The owner of that package
  - (DROP PACKAGE or CONTROL SCHEMA) ON SCHEMA for the schema to which the package bel ongs
  - DROP ANY PACKAGE ON DATABASE
- The user who performed the statement becomes the owner of the created package.

Syntax Rules and Parameters

OR REPLACE

It replaces an existing package body definition when the package body already exists.

PACKAGE NAME

It is a name of package body to be created, and the name as same as the name used in creating a packag e spec should be used. It should be a unique name in a schema.
It can define the schema to which the package belongs, such as schema_name.package_name. If schema _name is omitted, the default schema name of the user performing the statement is used.
The length of a package name should be shorter than 128 bytes.

Item Declaration

It declares PSM identifiers (variable, type, cursor, function spec, procedure spec, exception) to be used in a package body.
Routines declared in a package spec should be defined in a package body.
The cursor which was declared in a package spec without SQL should be defined in a package body.
Also, a function, a procedure, an argument of a cursor and the returning type in a package should be as s ame as those defined in a body statement.

Initialization Part

It describes statements which are performed only once to initialize internal variables while creating a pack age instance.

Description

It creates the schema-level package specification.
All public items in the created package can be referred by another procedure/ function/ package/ anony mous block.

The information about package body creation can be viewed in MODULES_BODY table in DEFINITION_SC HEMA or INFORMATION_SCHEMA.
The list of each private items in a package body can not be viewed in DEFINITION_SCHEMA or INFORMAT ION_SCHEMA.
If the information about objects referred by a procedure/ function/ variable in the package are temporaril y unstable, try to recompile it by using <alter package body> statement.

The created package body can be dropped by using <drop package> or <drop package body> statement.

Examples

```
CREATE OR REPLACE PACKAGE PKG1
IS
  V1 INTEGER;
  PROCEDURE PROC1;
  FUNCTION FUNC1 RETURN INTEGER;
END;
/
Package created.
CREATE OR REPLACE PACKAGE BODY PKG1
IS
  FUNCTION FUNC1 RETURN INTEGER
  IS
  BEGIN
      RETURN V1;
  END;
  PROCEDURE PROC1
  IS
  BEGIN
      IF V1 IS NULL
      THEN
          V1 := 10;
      ELSE
```

```
          V1 := V1 + 10;
      END IF;
  END;
END;
/
Package created.
```

Compatibility

The SQL standard does not define it.

For More Information

Refer to the followings.

- CREATE PACKAGE
- DROP PACKAGE
- ALTER PACKAGE

29.8 CREATE PROCEDURE

Function

It defines a schema-level procedure.

Syntax

```
<create procedure statement> ::=
      CREATE [ OR REPLACE ] PROCEDURE <procedure name> [ ( <parameter list> ) ]
      { IS ¦ AS }
      <item declaration>
      BEGIN
      <pl statement list>
      END [ <procedure name> ]
      ;

<parameter list> ::=
      <parameter name> [ <parameter mode> ] <datatype> [ <parameter default> ] [ , ... ]
<parameter mode> ::=
      IN
    ¦ OUT
    ¦ IN OUT

<parameter default> ::=
      { := ¦ DEFAULT } <value expression>
<procedure characteristics> ::=
      AUTHID CURRENT_USER
    ¦ AUTHID DEFINER
```

Invocation and Access Rules

The user should satisfy the following conditions to perform <create procedure statement>.

- One of the following privileges is required to create a procedure.

    - (CREATE PROCEDURE or CONTROL SCHEMA) ON SCHEMA for the schema to which the procedure belongs
    - CREATE ANY PROCEDURE ON DATABASE
- If a procedure already exists when using OR REPLACE clause, then one of the following privileges dropping the existing procedure is required.
    - The owner of that procedure
    - (DROP PROCEDURE or CONTROL SCHEMA) ON SCHEMA for the schema to which the procedure belongs
    - DROP ANY PROCEDURE ON DATABASE
- The user who performed the statement becomes the owner of the created procedure.

Syntax Rules and Parameters

OR REPLACE

It replaces an existing procedure with a new function when the procedure already exists.

procedure name

It is a name of procedure to be created, and it should be a unique name in a schema.
It can define the schema to which the procedure belongs, such as schema_name.procedure_name. If schema_name is omitted, the default schema name of the user performing the statement is used.
The length of a procedure name should be shorter than 128 bytes.

parameter name

It defines a parameter name of the procedure.
The name of each parameter should be unique in a procedure.
In other words, the procedure's parameter and PL item can not have the same name.
The length of a parameter name should be shorter than 128 bytes.
The maximum number of parameters available in a single procedure is limitless.

parameter mode

It sets each parameter mode.
The parameter modes are IN, OUT, and IN OUT.
If the parameter mode is not specified, the default mode is *IN*.

parameter default

It is the default value of the parameter.
The parameter with the specified parameter default can be omitted when executing the procedure.
If the parameter is not specified but omitted, then the default value is <value expression> specified when defining the parameter.
The datatype of <value expression> should be the datatype of the parameter.
All parameters defined after the parameter having <parameter default> should have <parameter default>.

procedure characteristics

It defines options to perform a procedure. It should be specified only once per a single item.

- AUTHID CURRENT_USER: SQLs which are being performed during performing a procedure are interp reted and performed according to an authorization of an invoker.
- AUTHID DEFINER: SQLs which are being performed during performing a procedure are interpreted a nd performed according to an authorization of a definer.

Item Declaration

It declares items such as a local variable to be used within a procedure.
It can declare all items which can be declared in a PL block.

PL Stmt List

It is a body section of a procedure, and it lists PL statements to be performed.
It can not use a bind parameter such as '?' or ':V1'. within a procedure.

Description

It defines a schema-level SQL procedure. The created procedure can be called from CALL statement, anon ymous block, or other procedure/ function.

The definition of a procedure can be viewed in ROUTINES table of INFORMATION_SCHEMA. The definitio n of a procedure parameter can be viewed in PARAMETERS table of INFORMATION_SCHEMA.

If a procedure becomes temporarily unstable due to absences of related objects, then it can try to recreat e a plan by using **ALTER PROCEDURE** statement.

The created procedure can be dropped by using **DROP PROCEDURE** statement.
The maximum number of procedures to be created is not limited. Therefore, they can be created as many as the storage space is available.

Examples

```
gSQL> CREATE OR REPLACE PROCEDURE PROC1( A1 INTEGER, A2 INTEGER )
  IS
    V1 INTEGER;
  BEGIN
    SELECT COUNT(*)
      INTO V1
      FROM T1
      WHERE T1.I1 >= A1 AND T1.I1 <= A2;
    DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
  END;
  /
Procedure created.
```

Compatibility

The SQL standard does not define the following clauses.

Table 29-4 SQL standard compatibility

Feature ID	Description	Compatibility
T471	Result sets return value	X
T341	Overloading of SQL-invoked functions and SQL-invoked procedures	X
S023	Basic structured types	X
S241	Transform functions	X
S024	Enhanced structured types	X
T571	Array-returning external SQL-invoked functions	X
T572	Multiset-returning external SQL-invoked functions	X
S201	SQL routines on arrays	X
S202	SQL-invoked routines on multisets	X
T323	Explicit security for external routines	X
S231	Structured type locators	X
S232	Array locators	X
S233	Multiset locators	X

Feature ID	Description	Compatibility
T041	Basic LOB data type support	X
S027	Create method by specific method name	X
T041	Basic LOB data type support	X
T324	Explicit security for SQL routines	O
T326	Table functions	O
T651	SQL-schema statements in SQL routines	X
T652	SQL-dynamic statements in SQL routines	O
T653	SQL-schema statements in external routines	X
T654	SQL-dynamic statements in external routines	X
T655	Cyclically dependent routines	X
T272	Enhanced savepoint management	X
T522	Default values for IN parameters of SQL-invoked procedures	O
B121	Routine language Ada	X
B122	Routine language C	X
B123	Routine language COBOL	X
B124	Routine language Fortran	X
B125	Routine language MUMPS	X
B126	Routine language Pascal	X
B127	Routine language PL/I	X
B128	Routine language SQL	O
B129	Routine language Ada: VARCHAR and NUMERIC support	X

For More Information

Refer to the followings.

- DROP PROCEDURE
- ALTER PROCEDURE

29.9 DROP FUNCTION

Function

It drops a function.

Syntax

```
<drop function statement> ::=
    DROP FUNCTION [ IF EXISTS ] func_name
    ;
```

Invocation and Access Rules

One of the following privilege is required to perform <drop function statement>.

- The owner of that function
- (DROP PROCEDURE or CONTROL SCHEMA) ON SCHEMA for the schema to which the function belo ngs
- DROP ANY PROCEDURE ON DATABASE

Syntax Rules and Parameters

IF EXISTS

Even when the function does not exist, an error does not occur.

FUNC NAME

It is the function name to be dropped.
It can define the schema to which the function belongs, such as schema_name.func_name. If schema_na me is omitted, the default schema name of the user performing the statement is used.

Description

It drops a specified schema-level function.

Examples

```
gSQL> CREATE OR REPLACE FUNCTION FUNC1
RETURN INTEGER
 IS
    V1 INTEGER;
  BEGIN
    V1 := 10;
    RETURN V1;
  END;
  /
Function created.
COMMIT;
Commit complete.
gSQL> DROP FUNCTION FUNC1;
Function dropped.
```

Compatibility

The SQL standard does not define the following clauses.

Table 29-5 SQL standard compatibility

Feature ID	Description	Compatibility
F032	CASCADE drop behavior	X
S024	Enhanced structured types	X

For More Information

Refer to the followings.

- CREATE FUNCTION

- ALTER FUNCTION

29.10 DROP PACKAGE

Function

It drops a package (of only body or both spec/ body).

Syntax

```
<drop package statement> ::=
    DROP PACKAGE [BODY] [ IF EXISTS ] package_name
    ;
```

Invocation and Access Rules

One of the following privilege is required to perform <drop package statement>.

- The owner of that package
- (DROP PACKAGE or CONTROL SCHEMA) ON SCHEMA for the schema to which the package belongs
- DROP ANY PACKAGE ON DATABASE

Syntax Rules and Parameters

BODY

It drops only the body object in the package of the given name. If the keyword *BODY* is not specified it dr ops both the package specification and the body.

IF EXISTS

Even when the package does not exist, an error does not occur.

PACKAGE NAME

It is the package name to be dropped.
It can define the schema to which the package belongs, such as schema_name.package_name. If schema _name is omitted, the default schema name of the user performing the statement is used.

Description

It drops the specified package object.

Examples

```
CREATE PACKAGE PKG1
  IS
    V1 INTEGER;
    FUNCTION FUNC1 (A1 INTEGER) RETURN INTEGER;
  END;
  /
Package created.
DROP PACKAGE IF EXISTS PKG1;
Package dropped.
```

Compatibility

It is DRO MODULE statement in the SQL standard.

For More Information

Refer to the followings.

- CREATE PACKAGE
- CREATE PACKAGE BODY
- ALTER PACKAGE

29.11 DROP PROCEDURE

Function

It drops a procedure.

Syntax

```
<drop procedure statement> ::=
    DROP PROCEDURE [ IF EXISTS ] proc_name
    ;
```

Invocation and Access Rules

One of the following privilege is required to perform <drop procedure statement>.

- The owner of that procedure
- (DROP PROCEDURE or CONTROL SCHEMA) ON SCHEMA for the schema to which the procedure bel ongs
- DROP ANY PROCEDURE ON DATABASE

Syntax Rules and Parameters

IF EXISTS

Even when the procedure does not exist, an error does not occur.

PROC NAME

It is the procedure name to be dropped.
It can define the schema to which the procedure belongs, such as schema_name.proc_name. If schema_ name is omitted, the default schema name of the user performing the statement is used.

Description

It drops a specified schema-level procedure.

Examples

```
gSQL> CREATE OR REPLACE PROCEDURE PROC1( A1 INTEGER, A2 INTEGER )
  IS
    V1 INTEGER;
  BEGIN
    SELECT COUNT(*)
      INTO V1
      FROM T1
      WHERE T1.I1 >= A1 AND T1.I1 <= A2;
    DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
  END;
  /
Procedure created.
COMMIT;
Commit complete.
gSQL> DROP PROCEDURE PROC1;
Procedure dropped.
```

Compatibility

The SQL standard does not define the following clauses.

Table 29-6 SQL standard compatibility

Feature ID	Description	Compatibility
F032	CASCADE drop behavior	X
S024	Enhanced structured types	X

For More Information

Refer to the followings.

- CREATE PROCEDURE
- ALTER PROCEDURE

Part V.

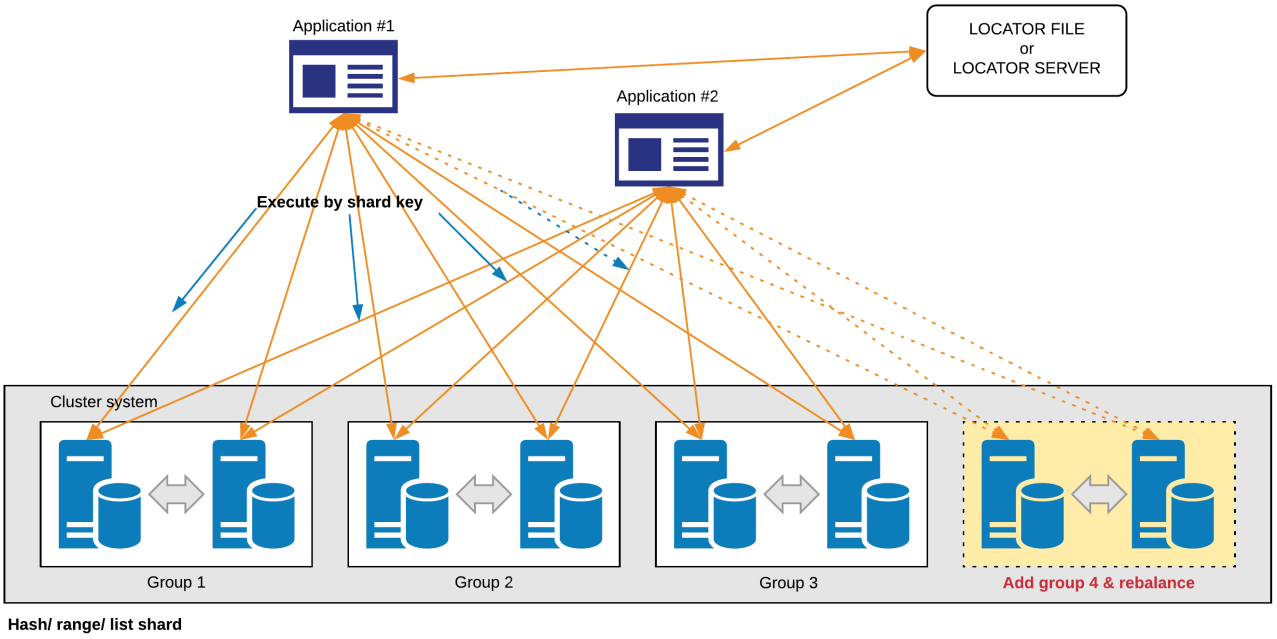
Developer Manual

30.

Database Connection

30.1 Features

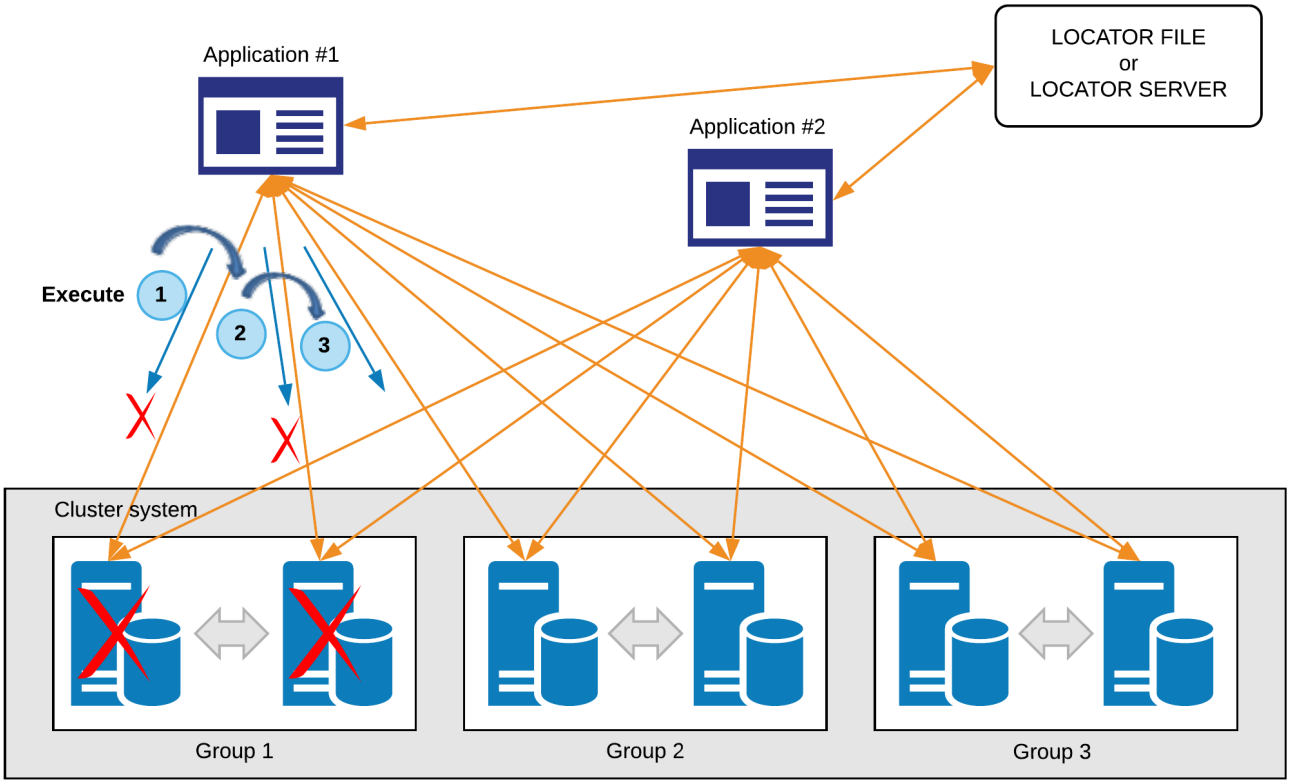
Figure 1 GLOBAL CONNECTION



Global connection feature is a method of optimizing transaction performance in consideration of data loc ality.
A general connection is a connection with a single member, but global connection is a connection with al l members. An application using the global connection makes the member with most data accessed by q ueries perform the query, so the performance is upgraded.
Hash, range, list sharding method can use the global connection, and the application does not need to b e altered at that moment.

When the online scale-out is performed, a user does not need to consider a new node but the the applica tion automatically connects to a new node and operates it.

Figure 2 GLOBAL CONNECTION HA (high availability)



If an error occurs on the selected node when performing SQL, then the SQL is performed through another node in the same group. If errors occur on all nodes in the selected group, then the SQL is performed through another group.

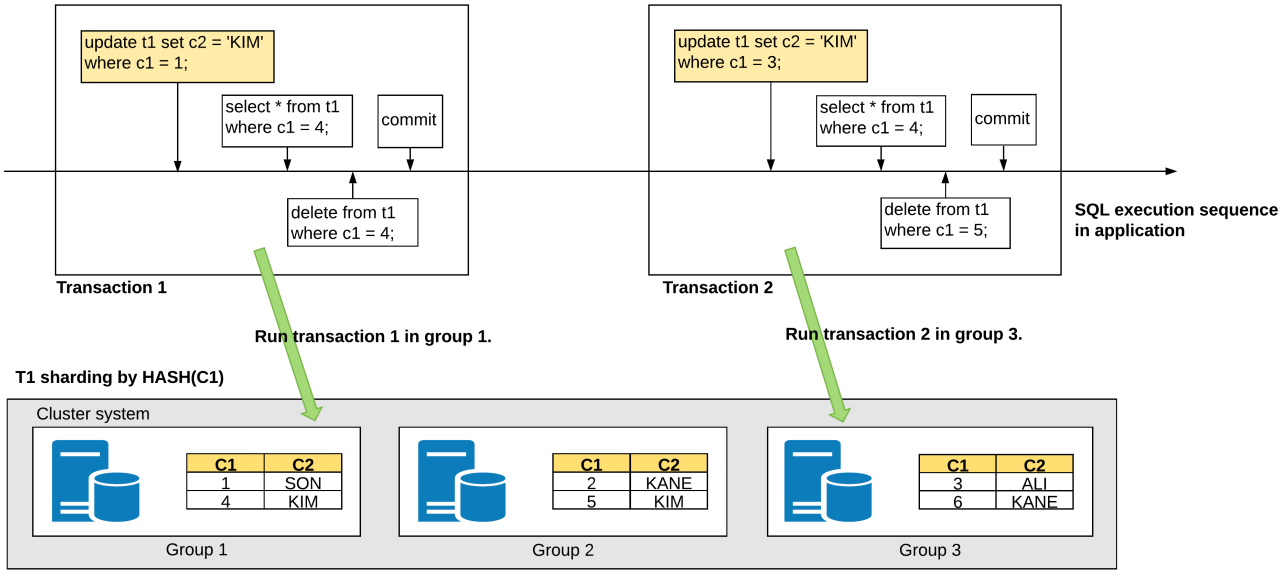
When a node is recovered from an error, the it is automatically connected to the node again online. Moreover, a user can make an application connect again by using the following syntax.

```
ALTER SYSTEM RECONNECT GLOBAL CONNECTION
```

30.2 Selecting Execution Member

An execution member is selected per transaction. When DML query is first performed without a transaction, a group is selected according to the sharding key, and an execution member in the group is selected according to LOCALITY_MEMBER_POLICY property. Then, all queries are processed in the selected member until COMMIT or ROLLBACK. If the first query can not select the appropriate group according to a sharding key, then the group is selected according to LOCALITY_GROUP_POLICY property.

Figure 3 Selecting a member

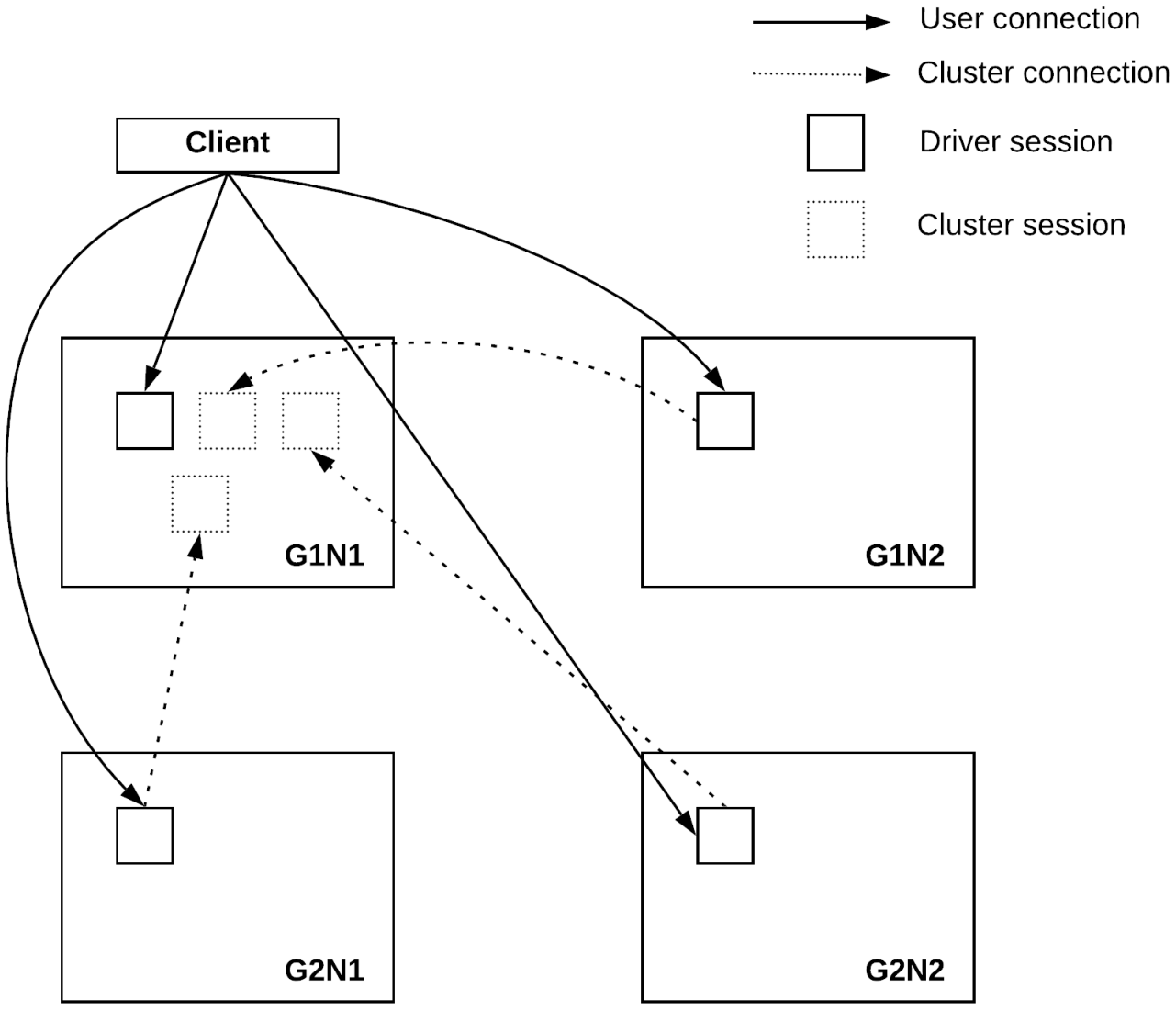


In case of transaction1 in the figure above, UPDATE query is performed in the group1 according to the sharding key, so all queries between COMMITs are performed in group1. In case of transaction2, UPDATE query is performed in group3, and all queries are performed in group3 before COMMIT even when further queries are not appropriate to perform in group3.

The read-only query (SELECT) without a transaction selects an execution member according to a sharding key like as other queries, but further queries are not always performed in the previously selected member.
In other words, all queries included in a transaction is performed only in a single member, but a query irrelevant to a transaction selects a member per a member.

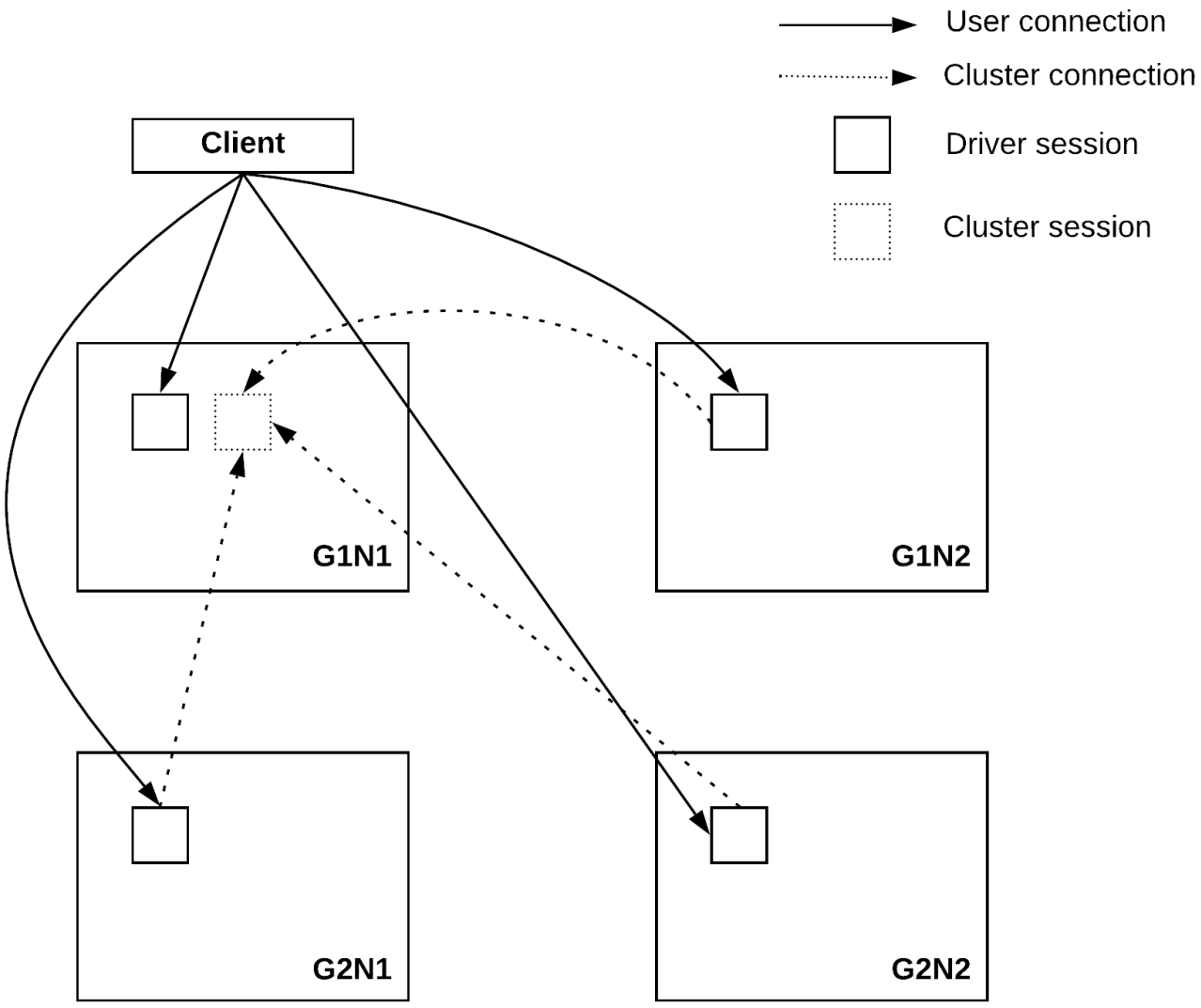
30.3 Global Session

Figure 4 Cluster session derived from global connection



The session directly connected to an application is called as a driver session, and a session from a driver se ssion to another member is called as a cluster session. Global connection creates a driver session in all me mbers, and creates cluster sessions in another member when it is needed. Global connection can make m ultiple cluster sessions in a single member.

Figure 5 Global session



Global session is a feature for a cluster sessions created in the global connection shares a single session to improve the resource efficiency. When the global connection does not use a global session, then the clust er session increases according to increasing groups and members. However, when it uses the global sessi on, then the cluster session does not increase even when groups and members increase.

The global session feature is available only in the global connection, but it is not available in the general c onnection.

30.4 Constraints

When using the global connection, a session dependent object is not available in SQL statement, a sessio n dependent clause nor is a session dependent function available.

Also, if SQLs accessing to multiple cluster nodes exist in a single transaction, then SQLs in the transaction uses the cluster node in common which were selected by the sharding key of the first SQL.

The query in the global connection is performed in consideration of data locality only when it is performe d with prepare execute. However, it is performed in an arbitrary node when it is performed with direct ex ecute.

Session Dependent Object

When using a session dependent object in SQL statement, the global connection is not available.

- Global temporary table

Session Dependent Clause

When using a session dependent clause in SQL statement, the global connection is not available.

- All @domain-related statements

Session Dependent Function and Pseudo Column

When using session dependent information in SQL statement, the global connection is not available.

- CURRVAL(sequence), sequence.CURRVAL
- UUID()
- VERSION()
- SESSION_ID()
- SESSION_SERIAL()
- USER_ID()
- LAST_IDENTITY_VALUE()
- STATEMENT_VIEW_SCN()
- STATEMENT_VIEW_SCN_GCN()
- STATEMENT_VIEW_SCN_DCN()
- STATEMENT_VIEW_SCN_LCN()
- LOCAL_GROUP_ID()
- LOCAL_MEMBER_ID()
- LOCAL_GROUP_NAME()

- LOCAL_MEMBER_NAME()
- CLUSTER_GROUP_ID
- CLUSTER_GROUP_NAME
- CLUSTER_MEMBER_ID
- CLUSTER_MEMBER_NAME
- CLUSTER_SHARD_ID

When Using Global Session

Data Definition Language (DDL) is not available in SQL statement.

30.5 Settings

For more information, refer to the followings.

- **Setting global connection in ODBC**
- **Setting global connection in JDBC**

31.

ODBC

31.1 Overview of GOLDILOCKS ODBC Driver

Concepts of GOLDILOCKS ODBC Driver

Open Database Connectivity (ODBC) is a specifications for a database Application Programming Interface (API). Microsoft ODBC version 3.0 is based on International Standards Organization/ International Electro mechanical Commission (ISO/ IEC) and the recommended specifications of Call Level Interface (CLI) of X/ open. ODBC supports the SQL statements by using C library functions. The application implements the O DBC features by calling these functions.

ODBC architecture has four components which perform the following features.

Component	Description
Application	It calls ODBC function which communicated with an ODBC data source and sends an SQL statement and processes the result set.
Driver manager	It manages the communication between an application and all ODBC drivers which ar e used by the application.
Driver	It processes all ODBC calls from applications, and connects to data source, and submit s the SQL statement to the data source in the application, and returns results to the a pplication. If necessary, the driver converts the ODBC SQL sent form the application to the default SQL which is used in the data source.
Data source	It includes all information which the driver needs to access the data in the DBMS.

The following operations can be executed by using ODBC applications.

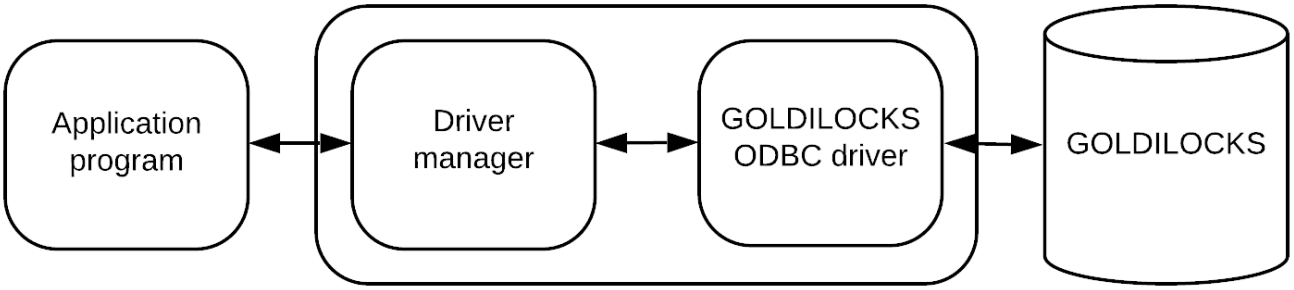
- Connecting to a data source
- Sending SQL statements to the data source
- Processing the results of an SQL statement in the data source
- Handling errors and messages
- Terminating the connection to the data source

Overview of ODBC Components

GOLDILOCKS ODBC Driver Including Driver Manager

The following is a software architecture of when the driver manager is included in the system. In this case, the application should be linked to the driver manager library.

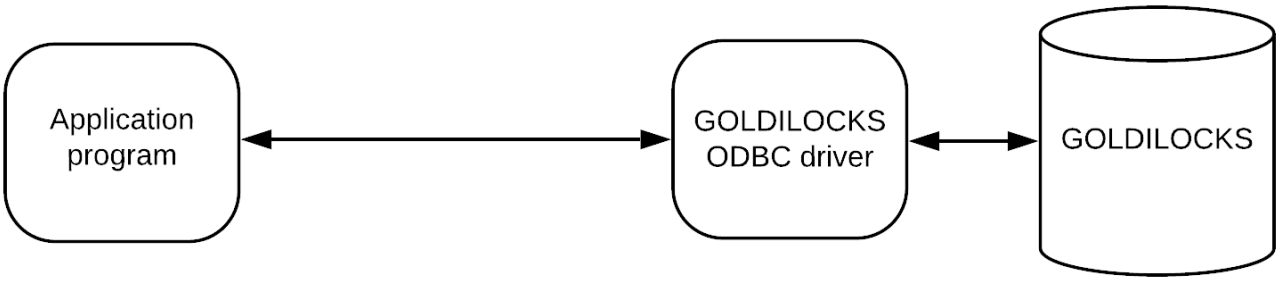
Figure 1 GOLDILOCKS ODBC driver including the driver manager



GOLDILOCKS ODBC Driver Not Including Driver Manager

The following is an architecture of when the application does not include a driver manager and uses GOL DILOCKS ODBC driver. In this case, the application should be linked to GOLDILOCKS ODBC driver library.

Figure 2 GOLDILOCKS ODBC driver not including the driver manager



Using GOLDILOCKS ODBC Driver

Header File

goldilocks.h file installed on $GOLDILOCKS_HOME/include should be included to execute GOLDILOCKS ODBC driver. This file defines constant and type of GOLDILOCKS ODBC driver, and provides a function pr ototype of GOLDILOCKS ODBC driver function.

Library

The application which does not use a driver manager should be linked to a static or shared version of GO LDILOCKS ODBC driver library.

UNIX

Table 31-1 GOLDILOCKS UNIX ODBC driver libraries

File name	Description
libgoldilocks.a	It is a static version of library including DA and CS.
libgoldilocksa.a	It is a static version of DA dedicated library.
libgoldilocksas.so	It is a shared version of DA dedicated library.
libgoldilocksc.a	It is a static version of CS dedicated library.
libgoldilockscs-ul32.so	It is a shared version of 64-bit CS dedicated library recognizing SQLLEN to 4 bytes.
libgoldilockscs-ul64.so	It is a shared version of 64-bit CS dedicated library recognizing SQLLEN to 8 bytes.
libgoldilockscs.so	It is a shared version of 32-bit CS dedicated library.
libgoldilockss.so	It is a shared version of library including DA and CS.

Windows

GOLDILOCKS Windows ODBC driver libraries provide CS libraries only.

Table 31-2 GOLDILOCKS Windows ODBC driver libraries

File name	Description
goldilockscs-ul64.dll	It is a shared version of 64-bit CS dedicated library recognizing SQLLEN to 8 bytes.
goldilockscs.dll	It is a shared version of 32-bit CS dedicated library.
goldilockssetup32.dll	It is a setup library for 32-bit ODBC driver manager.
goldilockssetup64.dll	It is a setup library for 64-bit ODBC driver manager.

31.2 Data Source Configuration

DSN Configuration on UNIX

odbcinst.ini File

odbcinst.ini file is a configuration file for installed ODBC driver.

- unixODBC

```
% odbcinst -j
unixODBC 2.3.2
DRIVERS............: /etc/odbcinst.ini
SYSTEM DATA SOURCES: /etc/odbc.ini
FILE DATA SOURCES..: /etc/ODBCDataSources
USER DATA SOURCES..: /home/goldilocks/.odbc.ini
SQLULEN Size.......: 8
SQLLEN Size........: 8
SQLSETPOSIROW Size.: 8
```

- iODBC

```
% iodbc-config --odbcinstini
/etc/odbcinst.ini
```

ODBC Driver Specification

ODBC driver specification section in odbcinst.ini file specifies the driver property values and list. The regist ered information section is under the driver name in each driver installed.

```
[driver_name]
Description = driver_description
Driver = driver_library_path
Setup = setup_library_path
FileUsage = file_usage
```

The following table describes keywords in the driver specification section.

Keyword	Description
Description	It is a string which describes the driver.

Keyword	Description
Driver	It is a driver library path.
Setup	It is a setup library path.
FileUsage	It is a character which displays how to directly process the file in DSN by the file-based driver.

The following is an example of information of GOLDILOCKS ODBC driver specifications.

```
[GOLDILOCKS ODBC Driver]
Description= GOLDILOCKS ODBC Driver
Driver = /home/goldilocks/home/lib/libgoldilockscs-ul64.so
Setup = /home/goldilocks/home/lib/libgoldilockscs-ul64.so
FileUsage = 0
```

odbc.ini File

odbc.ini file is the configuration file for the DSN connected by the application, and it is divided into a user DSN and system DSN. Typically, a user DSN file is ~ / .odbc.ini file, and a system DSN file is /etc/odbc.ini.

- unixODBC

```
% odbcinst -j
unixODBC 2.3.4
DRIVERS............: /etc/odbcinst.ini
SYSTEM DATA SOURCES: /etc/odbc.ini
FILE DATA SOURCES..: /etc/ODBCDataSources
USER DATA SOURCES..: /home/goldilocks/.odbc.ini
SQLULEN Size.......: 8
SQLLEN Size........: 8
SQLSETPOSIROW Size.: 8
```

- iODBC

```
% iodbc-config --odbcini
/etc/odbc.ini
```

Data Source Specification

Data source specification section of the odbc.ini file describes DSN.

```
[data_source_name]
Driver = driver_name
PROTOCOL = {DA ¦ TCP ¦ IPC}
```

```
CS_MODE = {default ¦ dedicated ¦ shared}
HOST = host_address
PORT = port_no
PREFER_IPV6 = {0 ¦ 1}
CHARSET = {SQL_ASCII ¦ UTF8 ¦ UHC ¦ GB18030}
TCP_NODELAY = {0 ¦ 1}
ALTERNATE_SERVERS = (HOST=ADDRESS1:PORT=PORT1,HOST=ADDRESS2:PORT=PORT2)
CONNECTION_RETRY_COUNT = retry_count
CONNECTION_RETRY_DELAY = retry_delay
FAILOVER_TYPE = {CONNECTION ¦ SESSION}
FAILOVER_GRANULARITY = {0 ¦ 1 ¦ 2}
FAILOVER_ROUTING_POLICY = {0 ¦ 1}
DATE_FORMAT = date_format_string
TIME_FORMAT = time_format_string
TIME_WITH_TIME_ZONE_FORMAT = timetz_format_string
TIMESTAMP_FORMAT = timestamp_format_string
TIMESTAMP_WITH_TIME_ZONE_FORMAT = timestamptz_format_string
CHAR_LENGTH_UNITS = {BYTE ¦ OCTETS ¦ CHAR ¦ CHARACTERS}
ENABLE_SQLDESCRIBEPARAM = {0 ¦ 1}
ENABLE_SQLBINDPARAMETER_CONSISTENCY_CHECK = {0 ¦ 1}
USE_TARGETTYPE = {0 ¦ 1 ¦ 2}
LOCATOR_DSN = locator_dsn_name
LOCATOR_SERVICE = locator_service_name
LOCALITY_AWARE_TRANSACTION = {0 ¦ 1}
LOCALITY_GROUP_POLICY = {0 ¦ 1 ¦ 2}
LOCALITY_GROUP_PATH = group_name1, group_name2, group_name3
LOCALITY_MEMBER_POLICY = {0 ¦ 1 ¦ 2 ¦ 3 ¦ 4}
LOCALITY_MEMBER_PATH = member_name1,member_name2, member_name3
DB_HOME = database_home_path
PACKET_COMPRESSION_THRESHOLD = packet_compression_threshold
USE_GLOBAL_SESSION = {0 ¦ 1}
CONNECTION_TIMEOUT = connection_timeout
LOGIN_TIMEOUT = login_timeout
TRACE = {0 ¦ 1}
TRACEFILE = file_path_name
TRACE_POLICY={DEFAULT ¦ ERROR}
INCLUDE_SYNONYMS = {0 ¦ 1}
DOT_NET_FOR_ODBC = {0 ¦ 1}
[locator_dsn_name]
FILE = location_file_name
HOST = IP address(v4)
```

```
PORT = locator_port
CONNECTION_TIMEOUT = second
ALTERNATE_LOCATORS = (HOST=ADDRESS1:PORT=PORT1,HOST=ADDRESS2:PORT=PORT2)
```

The following table describes keywords in the data source specification section.

Table 31-3 Keywords in the data source specification section

Keyword	Description
data_source_name	It is the data source specified in the data source section.
Driver	It is a driver name installed on odbcinst.ini.
PROTOCOL	It is a connection type with the server. • DA: It directly accesses to the server without communication. • TCP: It communicates with the server through TCP socket. • IPC: It communicates with the server through the shared memory, and it is available only in the same device of the server. It uses TCP connection to transfer IPC information when accessing for the first time, so HOST and PORT should be set. It can access the server only in dedicated mode.
CS_MODE	It sets whether to connect as dedicated mode or shared mode. If it is not set, the default mode is determined depending on the configuration (DEFAULT_CS_MODE) of the listener.
HOST	It is a host IP address or a host name.
PORT	It is a connection port number.
PREFER_IPV6	If the host parameter is the host name, then IPv6 takes precedence over other IP addresses.
TCP_NODELAY	It is a socket TCP_NODELAY option.
UID	It is a user ID.
PWD	It is a user password.
CHARSET	It is a client character set.
ALTERNATE_SERVERS	It is a server list which attempts to connect when the failover occurs. Each servers is separated by a comma (,). To disable the failover feature, set ALTERNATE_SERVERS as a white space.
CONNECTION_RETRY_COUNT	It is the number of times which the driver attempts to connect to the server when the connection fails.
CONNECTION_RETRY_DELAY	It is the server connection retry interval (in seconds) when the connection fails.
FAILOVER_TYPE	• CONNECTION: When the connection fails, it is connected to ALTERNATE_SERVERS. • SESSION: When the connection fails or the connection is disconnected during operating the statement, it is connected to ALTERNATE_SERVERS and the statement is restored. The statement is executed after the failover if the connection is disconnected when a transaction is not in progress.
FAILOVER_GRANULARITY	• 0: The failover proceeds even when an error occurs. • 1: The failover fails when an error except for SQLExecute (), SQLExecDirect ()occ

Keyword	Description
	urs during the failover). • 2: The failover fails when an error occurs.
DATE_FORMAT	It is a DATE type string.
TIME_FORMAT	It is a TIME type string.
TIME_WITH_TIME_ZONE_FORMAT	It is a TIME WITH TIME ZONE type string.
TIMESTAMP_FORMAT	It is a TIMESTAMP type string.
TIMESTAMP_WITH_TIME_ZONE_FORMAT	It is a TIMESTAMP WITH TIME ZONE type string.
CHAR_LENGTH_UNITS	It is the unit of ColumnSize when ParameterType in SQLBindParameter() is SQL_CHAR, SQL_VARCHAR. • BYTE, OCTETS: Byte unit • CHAR, CHARACTERS: Character unit
ENABLE_SQLDESCRIBEPARAM	It determines whether to enable SQLDescribeParam(). • 0: The driver does not support SQLDescribeParam(). • 1: The driver returns SQL_VARCHAR for all parameters.
ENABLE_SQLBINDPARAMETER_CONSISTENCY_CHECK	It determines whether to check ColumnSize and DecimalDigits in SQLBindParameter(). • 0: It does not check ColumnSize and DecimalDigits. • 1: It checks ColumnSize and DecimalDigits.
USE_TARGETTYPE	It sets the type information which is to be received together when receiving a column type through communication. • 0: It receives only the column type. • 1: It receives the column type and column name. • 2: It receives the column type and all information about the column.
LOCATOR_DSN	It is Data Source Name (DSN) which specifies a location information.
LOCATOR_SERVICE	It gets the connection information from a service hint and glocator.
LOCALITY_AWARE_TRANSACTION	It determines whether to use GLOBAL CONNECTION. • 0: It does not use GLOBAL CONNECTION. • 1: It uses GLOBAL CONNECTION.
LOCALITY_GROUP_POLICY	It determines how to select a group if neither of groups are available, or two or more groups are available when using GLOBAL CONNECTION. • 0: It randomly selects the group. • 1: It sequentially selects groups which exist in LOCALITY_GROUP_PATH setting. If neither of groups in LOCALITY_GROUP_PATH are not available, it randomly selects the group. • 2: It sequentially selects groups. It always selects groups in an order of they are connected to the driver.
LOCALITY_GROUP_PATH	It defines the list of selected groups when the available group is not a single one when using GLOBAL CONNECTION. Each group is distinguished with comma (,). e.g. G1,G2,G3
	It determines how to select a member in the selected group when using GLOBAL CO

Keyword	Description
LOCALITY_MEMBER_POLICY	NNECTION. • 0: DML : MASTER / SELECT : MASTER • 1: DML : ANY / SELECT : ANY • 2: DML : MASTER / SELECT : ANY • 3: DML : MASTER / SELECT : SLAVE • 4: It sequentially selects members which exist in LOCALITY_MEMBER_PATH setting. If neither of members in LOCALITY_MEMBER_PATH are not available, it uses the MASTER in the selected group.
LOCALITY_MEMBER_PATH	It defines the list of members to be used in the selected group when using GLOBAL CONNECTION. Each member is distinguished with comma (,). e.g. G1N1,G2N1,G3N1,G1N2,G2N2,G3N2
DB_HOME	It sets the home directory of the database. The default value uses $GOLDILOCKS_HOME environment variable.
PACKET_COMPRESSION_THRESHOLD	It compresses the communication data when the size of the communication data to be sent to the server is bigger than PACKET_COMPRESSION_THRESHOLD. The range of the set value is 32 ~ 2113929216.
USE_GLOBAL_SESSION	It is whether to use GLOBAL SESSION. • 0: It does not use GLOBAL SESSION. • 1: uses GLOBAL SESSION.
CONNECTION_TIMEOUT	It is waiting time (in seconds) for the response after the request.
LOGIN_TIMEOUT	It is waiting time (in seconds) for the login request to be completed.
TRACE	It sets whether to use trace in ODBC API. • 0: It does not use the trace. • 1: It uses the trace.
TRACEFILE	It is the name of the trace file. When the relative path is input, then it is based on the directory in which the program currently runs. The default value is 'odbc_trace.log'.
TRACE_POLICY	It is the trace policy. • DEFAULT: It records both the function parameter and the result. • ERROR: It records the log when the function fails. The default value is DEFAULT.
INCLUDE_SYNONYMS	It sets whether to include the synonym object in SQLGetColumns(). • 0: It does not include the synonym object. • 1: It includes the synonym object.
DOT_NET_FOR_ODBC	It is whether to use ODBC for .NET Framework. • 0: It does not change its usage. • 1: It replaces SQL_DESC_BASE_COLUMN_NAME, SQL_DESC_NAME properties, in SQLGetDescField() and SQLColAttribute() with SQL_DESC_LABEL property.

Table 31-4 Location

Keyword	Description
FILE	Location file name

Keyword	Description
HOST	glocator ip address
PORT	glocator port number
CONNECTION_TIMEOUT	Connection timeout with glocator (second)
ALTERNATE_LOCATORS	If glocator does not respond, it gets the connection information by using ALTERNATE_LOCATORS.

- If properties of FILE, HOST, PORT are all set in LOCATOR_DSN, then the FILE property is prior to others. For more information, refer to **Location File**.

- LOCATOR_SERVICE property enables to connect to a server belonging to LOCATOR_SERVICE. Servers other than the connected server become ALTERNATE_SERVERS.
  If FAILOVER_TYPE is not set, then FAILOVER_TYPE is a session.
  If FAILOVER_GRANULARITY is not set, then FAILOVER_GRANULARITY is 1.
  For more information, refer to **glocator** and **gloctl**.

The following is an example of DSN configuration of GOLDILOCKS.

```
[GOLDILOCKS]
Driver = GOLDILOCKS ODBC Driver
PROTOCOL = TCP
CS_MODE = SHARED
HOST = 192.168.0.10
PORT = 22581
CHARSET = UTF8
TCP_NODELAY = 1
ALTERNATE_SERVERS = (HOST=192.168.0.11:PORT=22581,HOST=192.168.0.12:PORT=22581)
CONNECTION_RETRY_COUNT = 3
CONNECTION_RETRY_DELAY = 1
FAILOVER_TYPE = SESSION
FAILOVER_GRANULARITY = 0
FAILOVER_ROUTING_POLICY = 0
DATE_FORMAT = YYYY-MM-DD
TIME_FORMAT = HH24:MI:SS.FF6
TIME_WITH_TIME_ZONE_FORMAT = HH24:MI:SS.FF6 TZH:TZM
TIMESTAMP_FORMAT = YYYY-MM-DD HH24:MI:SS.FF6
TIMESTAMP_WITH_TIME_ZONE_FORMAT = YYYY-MM-DD HH24:MI:SS.FF6 TZH:TZM
CHAR_LENGTH_UNITS = CHARACTERS
ENABLE_SQLDESCRIBEPARAM = 1
```

```
ENABLE_SQLBINDPARAMETER_CONSISTENCY_CHECK = 1
USE_TARGETTYPE = 0
INCLUDE_SYNONYMS = 0
PACKET_COMPRESSION_THRESHOLD = 2113929216
LOCALITY_AWARE_TRANSACTION = 0
LOCALITY_GROUP_POLICY = 0
LOCALITY_GROUP_PATH = G1,G2,G3
LOCALITY_MEMBER_POLICY = 0
LOCALITY_MEMBER_PATH = G1N1,G2N1,G3N1,G1N2,G2N2,G3N2
USE_GLOBAL_SESSION = 0
CONNECTION_TIMEOUT = 0
LOGIN_TIMEOUT = 0
LOCATOR_DSN = LOCATOR
LOCATOR_SERVICE = S1
TRACE = 1
TRACEFILE = /home/test/log/mytrace.log
TRACE_POLICY = DEFAULT
DOT_NET_FOR_ODBC = 0
[LOCATOR]
FILE = /home/test/.location.ini
HOST = 127.0.0.1
PORT = 42581
ALTERNATE_LOCATORS=(HOST=127.0.0.1:PORT=42582,HOST=127.0.0.1:PORT=42583)
```

DSN Configuration on Windows

ODBC data source manager can add or set up DSN on Windows.

Figure 3 Creating new data source

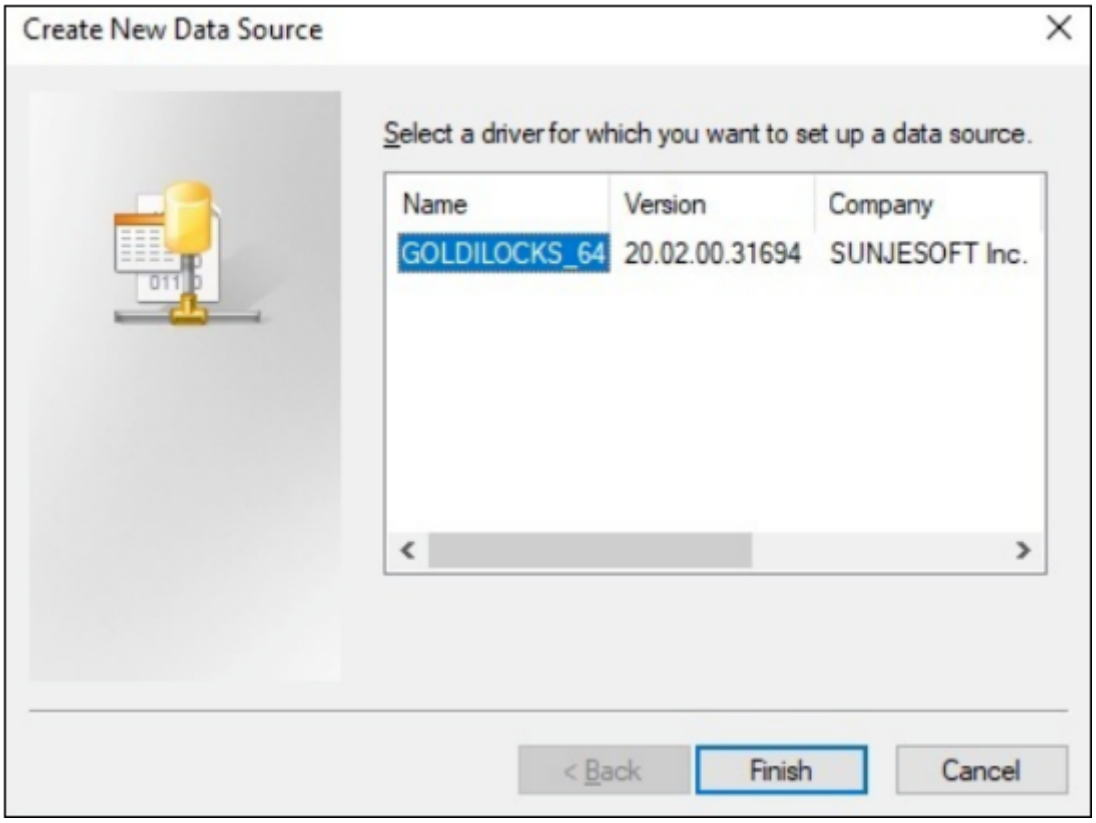
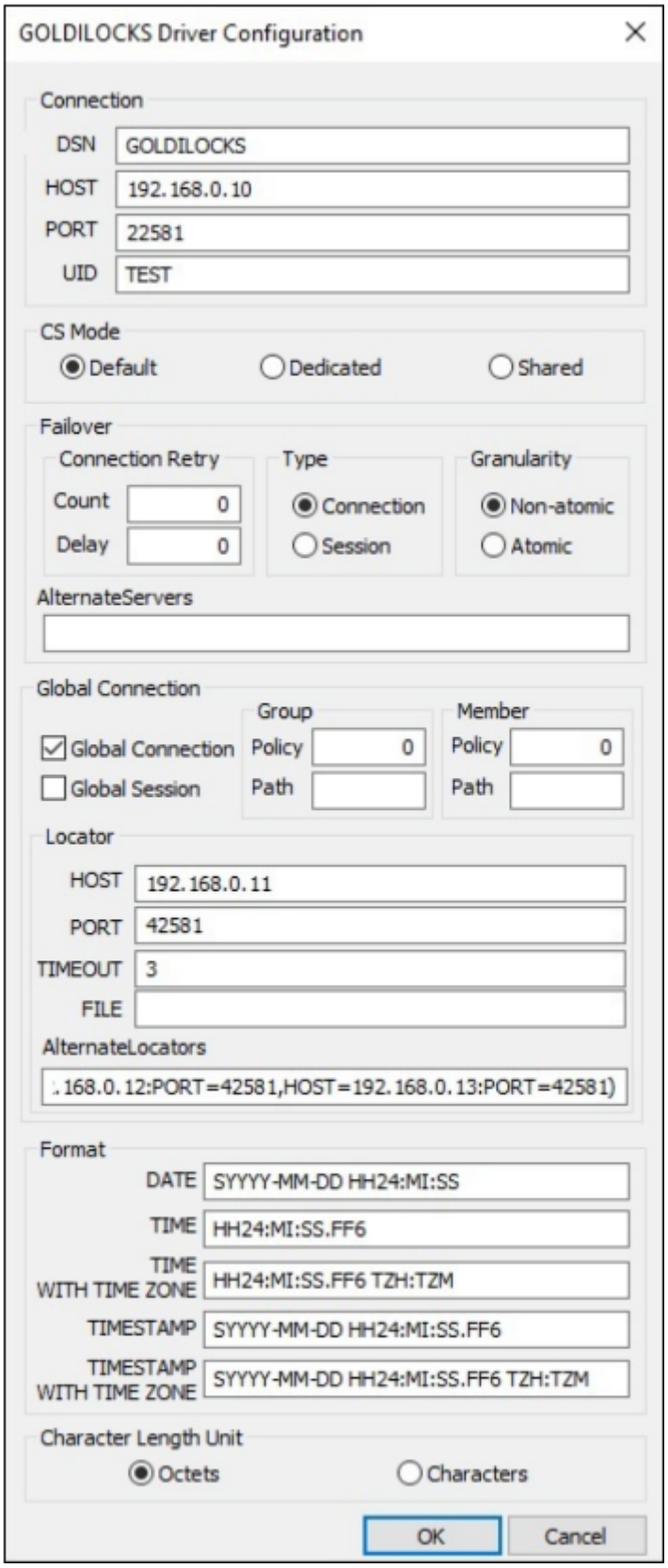


Figure 4 Configuring ODBC driver



The following table describes keywords for DSN configuration.

Table 31-5 Keywords for DSN configuration

Keyword	Description
DSN	It is a data source name.
HOST	It is a host IP address or a host name.
PORT	It is a connection port number.
UID	It is a user ID.
CS_MODE	It sets whether to connect as dedicated mode or shared mode. If it is not set, the default mode is determined depending on the configuration (DEFAULT_CS_MODE) of the listener.
ALTERNATE_SERVERS	It is a server list which attempts to connect when the failover occurs. Each servers is separated by a comma (,). To disable the failover feature, set ALTERNATE_SERVERS as a white space.
CONNECTION_RETRY_COUNT	It is the number of times which the driver attempts to connect to the server when the connection fails.
CONNECTION_RETRY_DELAY	It is the server connection retry interval (in seconds) when the connection fails.
FAILOVER_TYPE	• CONNECTION: When the connection fails, it is connected to ALTERNATE_SERVERS. • SESSION: When the connection fails or the connection is disconnected during operating the statement, it is connected to ALTERNATE_SERVERS and the statement is restored. The statement is executed after the failover if the connection is disconnected when a transaction is not in progress.
FAILOVER_GRANULARITY	• Non-atomic: The failover proceeds even when an error occurs. • Atomic: The failover fails when an error occurs.
DATE_FORMAT	It is a DATE type string.
TIME_FORMAT	It is a TIME type string.
TIME_WITH_TIME_ZONE_FORMAT	It is a TIME WITH TIME ZONE type string.
TIMESTAMP_FORMAT	It is a TIMESTAMP type string.
TIMESTAMP_WITH_TIME_ZONE_FORMAT	It is a TIMESTAMP WITH TIME ZONE type string.
CHAR_LENGTH_UNITS	It is the unit of ColumnSize when ParameterType in SQLBindParameter() is SQL_CHAR, SQL_VARCHAR. • BYTE, OCTETS: Byte unit • CHAR, CHARACTERS: Character unit
LOCALITY_AWARE_TRANSACTION	It sets whether to use GLOBAL CONNECTION. • It does not use GLOBAL CONNECTION. • It uses GLOBAL CONNECTION.
USE_GLOBAL_SESSION	It sets whether to use GLOBAL SESSION. • It does not use GLOBAL SESSION. • It uses GLOBAL SESSION.
	If an available group does not exist or two or more groups can be selected when using GLOBAL CONNECTION, it sets how to select a group.

Keyword	Description
LOCALITY_GROUP_POLICY	• 0: It arbitrarily selects a group. • 1: It sequentially selects groups existing in LOCALITY_GROUP_PATH settings. If all groups in LOCALITY_GROUP_PATH are not available, then it arbitrarily selects a group. • 2: It sequentially selects groups. It selects a group in a sequence of connected to the driver everytime.
LOCALITY_GROUP_PATH	It specifies a list of selected groups if more than one group can be selected when using GLOBAL CONNECTION. Each group is separated with a comma (,). e.g. G1,G2,G3
LOCALITY_MEMBER_POLICY	It determines how to select a member within a selected group when using GLOBAL CONNECTION. • 0: DML : MASTER / SELECT : MASTER • 1: DML : ANY / SELECT : ANY • 2: DML : MASTER / SELECT : ANY • 3: DML : MASTER / SELECT : SLAVE • 4: It sequentially selects members in LOCALITY_MEMBER_PATH settings. If all members in LOCALITY_MEMBER_PATH are not available, then it arbitrarily selects MASTER in the selected group.
LOCALITY_MEMBER_PATH	It specifies a list of members to be used within a selected group when using GLOBAL CONNECTION. Each member is separated with a comma (,). e.g. G1N1,G2N1,G3N1,G1N2,G2N2,G3N2
LOCATOR_HOST	glocator ip address
LOCATOR_PORT	glocator port number
LOCATOR_CONNECTION_TIMEOUT	Connection timeout with glocator (second)
ALTERNATE_LOCATORS	If glocator does not respond, then it gets the connection information by using ALTERNATE_LOCATORS.
TRACE	It sets whether to use trace in ODBC API. • 0: It does not use the trace. • 1: It uses the trace.
TRACEFILE	It is the name of the trace file. When the relative path is input, then it is based on the directory in which the program currently runs. The default value is 'odbc_trace.log'.
TRACE_POLICY	It is the trace policy. • DEFAULT: It records both the function parameter and the result. • ERROR: It records the log when the function fails. The default value is DEFAULT.
DOT_NET_FOR_ODBC	It is whether to use ODBC for .NET Framework. • 0: It does not change its usage. • 1: It replaces SQL_DESC_BASE_COLUMN_NAME, SQL_DESC_NAME properties, in SQLGetDescField() and SQLColAttribute() with SQL_DESC_LABEL property.

31.3 GLOBAL CONNECTION

GLOBAL CONNECTION of which an application selects and performs an appropriate node for a query pro cessing in cluster environment is supported.

Settings

GLOBAL CONNECTION is available only when PROTOCOL is TCP. In this case, LOCALITY_AWARE_TRANS ACTION property value should be set together with LOCATOR file or LOCATOR server. USE_GLOBAL_SES SION property value should be set to 1 to use the global session feature.

- .odbc.ini of when using DSN

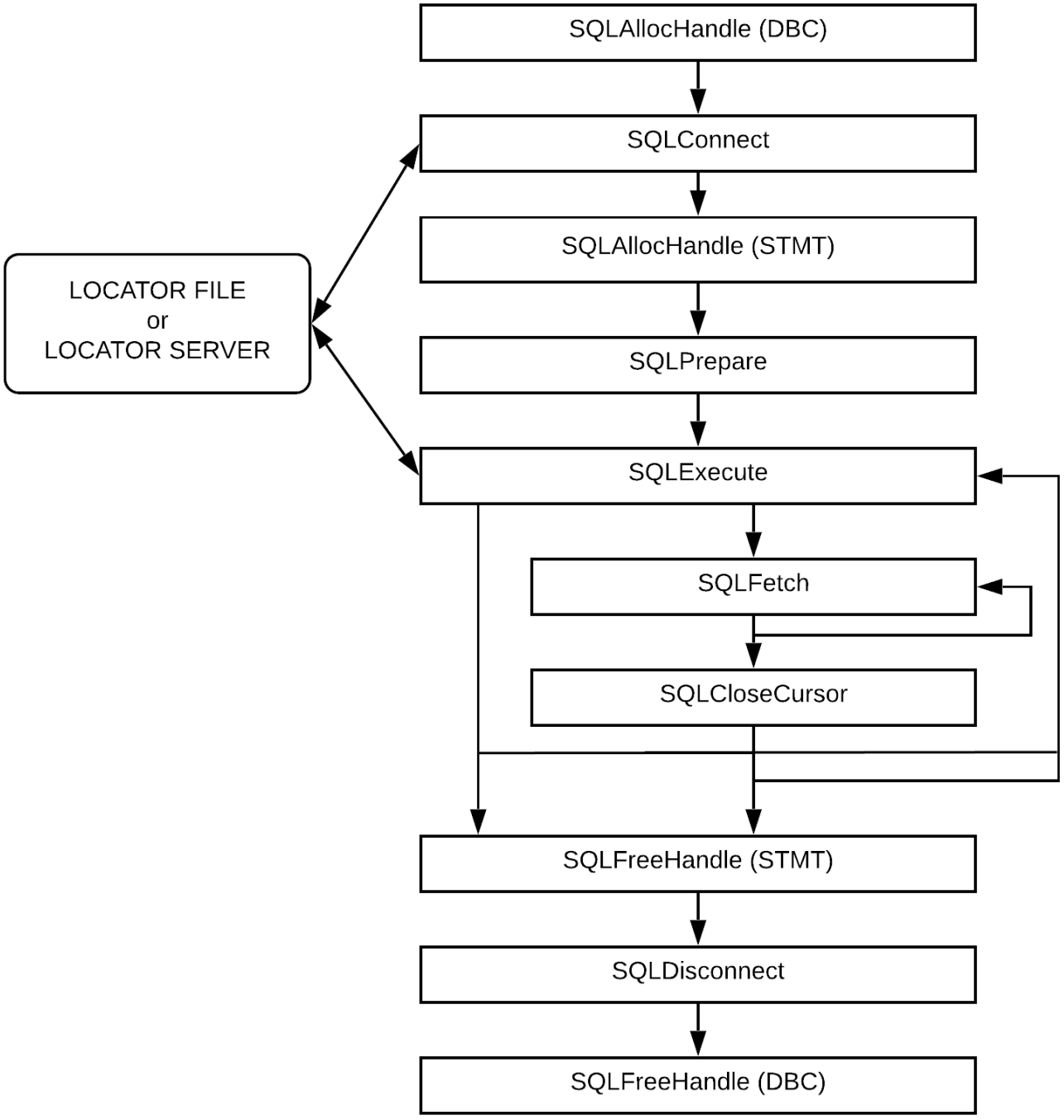
```
[GOLDILOCKS]
PROTOCOL = TCP
HOST = 192.168.0.1
PORT = 22581
UID = TEST
PWD = test
LOCALITY_AWARE_TRANSACTION = 1
LOCATOR_DSN = LOCATOR
[LOCATOR]
FILE = /home/goldilocks/.location.ini
```

- Using connection string

```
SQLDriverConnect( dbc,
                  NULL,
                  (SQLCHAR*)"PROTOCOL=TCP;HOST=192.168.0.1;PORT=22581;UID=TEST;PWD=test;LOCALI
TY_AWARE_TRANSACTION=1;LOCATOR_HOST=192.168.0.2;LOCATOR_PORT=42581",
                  SQL_NTS,
                  NULL,
                  0,
                  NULL,
                  SQL_DRIVER_NOPROMPT );
```

Processing GLOBAL CONNECTION

Figure 5 Basic steps of applying GLOBAL CONNECTION



1. SQLAllocHandle (DBC)
   It allocates a connection handle.

2. SQLConnect
   It connects to a server which was given server information from a user, and obtains the information a bout the cluster system. Then builds the cluster system information through LOCATOR file or LOCAT

OR server, and connects to all nodes of cluster system.

3. SQLAllocHandle(STMT)
   It allocates a statement to each of all connected nodes.

4. SQLPrepare
   It prepares to execute SQL in all connected nodes.

5. SQLExecute
   If the information about the cluster system is not built, an application builds the information about th e cluster system through LOCATOR file or LOCATOR server, then connects to all nodes in the cluster s ystem.
   When connecting to a new node by adding a node to cluster, all statements of other nodes are equal ly created in that node, and prepares to execute SQL.

If the information about a sharding key has already built, an application selects an appropriate node and performs a query by using the sharding key information.
If the information about a sharding key is not built, an application builds the sharding key information of the SQL from an arbitrary server, then selects a node and performs a query.

If an error occurs on the selected node, an appropriate node is selected again, then performs a query.

If the information about a sharding key is altered after SQLExecute, then it deletes the built information about a sharding key
If the information about the cluster system is altered such as adding or deleting a cluster node after SQLE xecute, then it deletes the built cluster system information.

6. SQLFetch
   It brings data from the node on which SQL was executed.

7. SQLCloseCursor
   It closes a cursor from the node on which SQL was executed.

8. SQLFreeHandle(STMT)
   It releases a statement from all connected nodes.

9. SQLDisconnect
   It releases connections with all nodes.

10. SQLFreeHandle(DBC)
    It releases a connection handle.

Handling GLOBAL CONNECTION Exception

When using GLOBAL CONNECTION, if an error occurs on the selected node during the operation, then it is operated as follows according to the transaction occurrence and SELECT progress.

- When a transaction did not occur

If an error occurs on the selected node when a transaction did not occur, then it selects another node wit hin ODBC and executes the query. Though an error occurred on the selected node, the query was normal ly executed on another node, so it does not transfer an error to a user.

- When a transaction occurred or SELECT is in progress

If an error occurs on the selected node when a transaction occurred or SQLFetch() is in progress, then OD BC can not proceeds the current operation any more so it transfers 19068(Retry the transactional operati ons) error. If 19068 error occurs, then a user should perform the transaction or SELECT again.

```
if( !SQL_SUCCEEDED(SQLPrepare( sStmt,
                               (SQLCHAR*)"INSERT INTO T1 VALUES ( ? )",
                               SQL_NTS )) )
{
    goto stmt_error;
}
trans_retry:
if( !SQL_SUCCEEDED(SQLExecute( sStmt )) )
{
    SQLGetDiagRec( SQL_HANDLE_STMT,
                   sStmt,
                   1,
                   sSQLState,
                   &sNativeError,
                   sMessageText,
                   sizeof(sMessageText),
                   &sTextLength );
    if( sNativeError == 19068 )
    {
        goto trans_retry;
    }

    goto stmt_error;
}
```

```
if( !SQL_SUCCEEDED(SQLPrepare( sStmt,
                               (SQLCHAR*)"SELECT * FROM T1 WHERE C1 = ?",
                               SQL_NTS )) )
{
    goto stmt_error;
}
trans_begin:
sReturn = SQLExecute( sStmt );
if( sRetrun == SQL_ERROR )
{
    SQLGetDiagRec( SQL_HANDLE_STMT,
                   sStmt,
                   1,
                   sSQLState,
                   &sNativeError,
                   sMessageText,
                   sizeof(sMessageText),
                   &sTextLength );
    if( sNativeError == 19068 )
    {
        goto trans_retry;
    }

    goto stmt_error;
}
while( 1 )
{
    sReturn = SQLFetch( sStmt );
    if( sReturn == SQL_NO_DATA )
    {
        SQLCloseCursor( sStmt );
        break;
    }
    else if( sReturn == SQL_ERROR )
    {
        SQLGetDiagRec( SQL_HANDLE_STMT,
                       sStmt,
                       1,
                       sSQLState,
                       &sNativeError,
                       sMessageText,
```

```
                        sizeof(sMessageText),
                        &sTextLength );
        if( sNativeError == 19068 )
        {
            goto trans_retry;
        }

        goto stmt_error;
    }
    ...
}
```

- When committing or rolling back a transaction

If an error occurs on the selected node when committing a transaction, then ODBC checks whether the tr ansaction has been committed before the error occurred through another node. Though an error occurre d on the selected node, if the transaction was normally committed, then it does not transfer an error. Als o, if an error occurred on the selected node when a transaction has not been committed, then it transfers 19068(Retry the transactional operations) error. If 19068 error occurs, then a user should perform the tra nsaction again.

If an error occurs on the selected node when rolling back a transaction, then ODBC does not transfer an e rror. It is because the transaction already has been rolled back due to the node error.

```
if( !SQL_SUCCEEDED(SQLSetConnectAttr( sDbc,
                                      SQL_AUTOCOMMIT,
                                      (SQLPOINTER)SQL_AUTOCOMMIT_OFF,
                                      0 ) ))
{
    goto dbc_error;
}
if( !SQL_SUCCEEDED(SQLPrepare( sStmt,
                               (SQLCHAR*)"INSERT INTO T1 VALUES ( ? )",
                               SQL_NTS )) )
{
    goto stmt_error;
}
trans_retry:
if( !SQL_SUCCEEDED(SQLExecute( sStmt )) )
{
    SQLGetDiagRec( SQL_HANDLE_STMT,
                   sStmt,
```

```
                    1,
                    sSQLState,
                    &sNativeError,
                    sMessageText,
                    sizeof(sMessageText),
                    &sTextLength );
    if( sNativeError == 19068 )
    {
        goto trans_retry;
    }

    goto stmt_error;
}
if( !SQL_SUCCEEDED(SQLEndTran( SQL_HANDLE_DBC,
                               sDbc,
                               SQL_COMMIT)) )
{
    SQLGetDiagRec( SQL_HANDLE_DBC,
                    sDbc,
                    1,
                    sSQLState,
                    &sNativeError,
                    sMessageText,
                    sizeof(sMessageText),
                    &sTextLength );
    if( sNativeError == 19068 )
    {
        goto trans_retry;
    }

    goto stmt_error;
}
```

Constraints of GLOBAL CONNECTION

- Use SQLPrepare() and SQLExecute() to select a node appropriate for the query.

Use SQLPrepare() and SQLExecute() to select a node appropriate for the query by using GLOBAL CONNE CTION. SQLExecDirect() does not have an information required to select a node appropriate for the query, so the node is selected according to OCALITY_GROUP_POLICY and LOCALITY_MEMBER_POLICY propert

y.

- Use SQLEndTran() to commit or rollback a transaction.

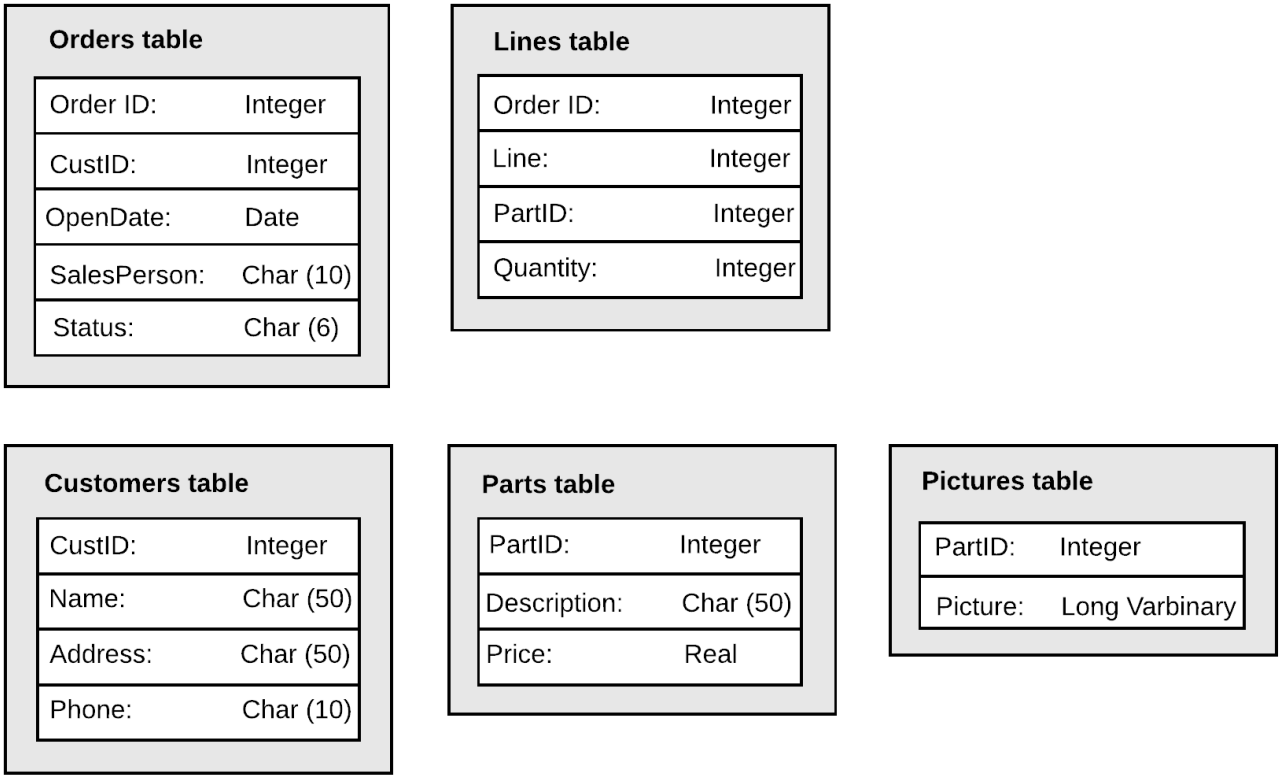
If committing or rolling back with SQL statement when using GLOBAL CONNECTION, then the status cha nge of the transaction is not detected. It is mandatory to use SQLEndTran() to commit or rollback the tran saction.

- Global session feature does not support Data Definition Language (DDL) among SQL statements.

31.4 Catalog Function

All databases have schemas of how to store the data in the database. For example, a simple sales order d atabase will have the schemas shown in the following figure, and the ID columns are used to connect the tables.

Figure 6 Schema of sales orders



The schema is stored in the set of system tables which is called as a database catalog along with other inf ormation such as privileges. This is also known as a data dictionary.

Applications can find this schema by calling the catalog functions. Catalog functions return the informati on to the result set, and typically they are implemented by SELECT statements for the tables in the catalo g.

Using Catalog Data

Applications use catalog data in various ways. The followings are some common usages.

Configuring SQL statements at the time of execution

The vertical applications such as the order input application include the hard-coded SQL statements. The tables and columns used by the application were previously fixed, and the application access these tables. For example, the ordering application typically has one parameterized statement to add a new order to the system.

A general application such as a spreadsheet program using ODBC for collecting the data sometimes configures the SQL statement based on the input from a user at the time of execution. This application may request a user for the format to use tables and columns. However, if the list of tables or columns selected by a user is shown to the application, it will be easier to the user. The application will call catalog functions such as SQLTables and SQLColumns to configure these lists.

Configuring SQL statements during the development

The application development environments allow the developers to create database queries during developing the program. And then the queries are hard-coded and embedded into the application.

These environments can also create a list of what were selected by the developer by using the SQLTables and SQLColumns. The environments find out and display the relationships among the tables automatically selected by using SQLPrimaryKeys and SQLForeignKeys. Then they find out and emphasize the index fields by using SQLStatistics, so the developers can create queries effectively.

Configuring a Cursor

The application, driver, middleware which provides a scroll cursor, find out column(s) which is the only column of identifying a row by using SQLSpecialColumns. The program can configure a keyset including the values of these columns for the collected rows. The application will use these values to collect the latest data for the rows by scrolling backwards.

Catalog Function on ODBC

ODBC includes the following catalog functions.

Function	Description
SQLTables	It returns a list of catalogs, schemas, tables or table types in the data source.
SQLColumns	It returns a list of columns in one or more tables.
SQLStatistics	It returns a list of statistics for a single table. It also returns a list of indexes linked to the table.
SQLSpecialColumns	It returns a list of columns which is the only column of identifying a row in a table. Also,

Function	Description
	it returns a list of columns in the table, and they are automatically updated.
SQLPrimaryKeys	It returns a list of columns which configure the primary key of a table.
SQLForeignKeys	It returns a list of foreign keys in a table or it returns a list of foreign keys in another table referring that table.
SQLTablePrivileges	It returns a list of privileges associated with one or more tables.
SQLColumnPrivileges	It returns a list of privileges associated with one or more columns in a table.
SQLProcedures	It is not supported by the driver.
SQLProcedureColumns	It is not supported by the driver.
SQLGetTypeInfo	It returns a list of SQL data types supported by the data source. These data types are generally used in CREATE TABLE, ALTER TABLE statements.

Data Returning of Catalog Function

Each catalog function returns the data as a result set. The result set is not different from any other result set. It is usually hard-coded into the driver, or it is created by the predefined statement such as the parameterized SELECT statement stored in the procedure of data source.

The result set for each catalog function is described in *For More Information* paragraph of each function in this user manual. The result set can include the columns specified in the driver after the column selected last besides the listed columns. These columns are described in a user manual for the driver.

The applications bind the columns specified in the driver based on the end of result set. They calculate the number of columns specified in the driver as the number of last columns which are smaller than the number of columns after the required column. This saves the trouble of changing the application when a new column is added in future version or ODBC driver. To operate this schema, drives should add the columns specified in the new driver before the columns specified in the old driver to prevent changing the row number based on the end of result set.

Even when they include special characters, they do not quote the identifiers returned in the result set. For example, if the Accounts Payable table's identifier quote character which is specified in the driver and returned by SQL GetInfo is a double quotes ("), and the Accounts Payable table has a Customer Name column, then TABLE_NAME column value in the rows which is returned by SQLColumns is Accounts Payable, but it is not "Accounts Payable", and COLUMN_NAME column value is Customer Name, but it is not "Customer Name".
The application collects the names of customers from the Accounts Payable table as follows.

```
SELECT "Customer Name" FROM "Accounts Payable"
```

Catalog functions are based on SQL-like model in connection based on the user name and password, and their data is returned only to the users with the proper privilege. The file password protection which is ina

ppropriate to this model is defined by the drivers.

Most result set returned by the catalog functions can not be updated, and the application should not exp ect to change the database schema by updating the data in the result set.

Arguments of Catalog Function

Pattern Value Argument

Some arguments in catalog function accept the search pattern, like as TableName argument in SQLTables. The arguments accept the search pattern if SQL_ATTR_METADATA_ID attribute is set to SQL_FALSE. The arguments do not accept the search pattern if SQL_ATTR_METADATA_ID attribute is set to SQL_TRUE.

Search pattern letters have the following features.

- An underscore(_) represents any single character.
- Percent sign (%) represents zero or more characters.
- An escape character is specified in the driver and it is used to include the percent sign, underscore, es cape character as literals. If an escape character is prior to a non-special character, it does not have a ny special meaning. But if an escape character is prior to a special character, it is a special character. F or example, "₩a" is treated as two characters consisting of "₩" and "a", but "₩%" refers to "%".

An escape character is returned by using SQL_SEARCH_PATTERN_ESCAPE option in SQLGetInfo. To inclu de that character as a literal in an argument which accepts search patterns, it should be prior to any unde rscore, percent sign, or escape character.

The following table describes how to use search patterns.

Search pattern	Description
%A%	It is all identifiers which contain A.
ABC_	It is all four letter characters which start with ABC.
ABC₩_	It assumes the escape character a backslash(₩), and the identifier is ABC_.
₩₩%	It assumes the escape character a backslash(₩), and the identifier which start with a backslash(₩) .

⚠ Be cautious when using a escape character in an argument which accepts a search pattern. This is particularly TRUE for the underscore(_) which is generally used as the identifier.
It is a common mistake in the application that the value returned by one catalog function is pass ed to the search pattern argument of another catalog function.
For example, if the application gets MY_TABLE table from the result set of SQLTables and passe s it to SQLColumns to retrieve the column list of MY_TABLE, then, the application will get the c olumns of all tables such as MY_TABLE, MY1TABLE, MY2TABLE instead of getting the columns

of MY_TABLE because they match the search pattern MY_TABLE.

ODBC 2.x driver does not support the search pattern for CatalogName argument of SQL tables. ODBC 3.x driver supports the search pattern within the argument if the environment attribute SQL_ATTR_ODBC_VERSION is set to SQL_OV_ODBC3. The argument does not accept the search pattern if this property is set to SQL_OV_ODBC2.

Passing a NULL pointer to the search pattern argument does not force the argument to search. NULL pointer and the search patterns % (any character) are equivalent. However, a zero-length search pattern is matched with an empty string ("").

31.5 Non-standard Data Type

This chapter describes GOLDILOCKS data types which are not defined in ODBC standard.

C Data Type

Table 31-6 GOLDILOCKS C data type

C type identifier	ODBC C typedef	C type
SQL_C_BOOLEAN	SQLBOOLEAN	unsigned char
SQL_C_LONGVARBINARY	SQL_LONG_VARIABLE_LENGTH_STRUCT	[a]
SQL_C_LONGVARCHAR	SQL_LONG_VARIABLE_LENGTH_STRUCT	[a]
SQL_C_REFCURSOR	SQLHSTMT	void *
SQL_C_TYPE_TIME_WITH_TIMEZONE	SQL_TIME_WITH_TIMEZONE_STRUCT	[b]
SQL_C_TYPE_TIMESTAMP_WITH_TIMEZONE	SQL_TIMESTAMP_WITH_TIMEZONE_STRUCT	[c]
SQL_C_WLONGVARCHAR	SQL_WLONG_VARIABLE_LENGTH_STRUCT	[d]

- [a]

```
typedef struct tagLONG_VARIABLE_LENGTH_STRUCT
{
    SQLBIGINT   len;
    SQLCHAR   * arr;
} LONG_VARIABLE_LENGTH_STRUCT;
```

- [b]

```
typedef struct tagTIME_WITH_TIMEZONE_STRUCT
{
   SQLUSMALLINT hour;
   SQLUSMALLINT minute;
   SQLUSMALLINT second;
   SQLUINTEGER  fraction;
   SQLSMALLINT  timezone_hour;
   SQLSMALLINT  timezone_minute;
```

```
} TIME_WITH_TIMEZONE_STRUCT;
```

- [c]

```
typedef struct tagTIMESTAMP_WITH_TIMEZONE_STRUCT
{
   SQLSMALLINT  year;
   SQLUSMALLINT month;
   SQLUSMALLINT day;
   SQLUSMALLINT hour;
   SQLUSMALLINT minute;
   SQLUSMALLINT second;
   SQLUINTEGER  fraction;
   SQLSMALLINT  timezone_hour;
   SQLSMALLINT  timezone_minute;
} TIMESTAMP_WITH_TIMEZONE_STRUCT;
```

- [d]

```
typedef struct tagWLONG_VARIABLE_LENGTH_STRUCT
{
    SQLBIGINT   len;
    SQLWCHAR  * arr;
} WLONG_VARIABLE_LENGTH_STRUCT;
```

SQL_C_BOOLEAN

It is used to describe boolean value.

- SQL_TRUE
- SQL_FALSE

```
SQLBOOLEAN boolean   = SQL_TRUE;
SQLLEN     indicator = 0;
SQLBindParameter( stmt,
                  1,
                  SQL_PARAM_INPUT,
                  SQL_C_BOOLEAN,
                  SQL_BOOLEAN,
                  0,
                  0,
                  &boolean,
```

```
                0,
                &indicator );
```

SQL_C_LONGVARBINARY

It has SQLCHAR buffer size and SQLCHAR buffer pointer.

SQLCHAR buffer has a binary data.

The actual length of the binary data is transferred through an indicator.

```
SQL_LONG_VARIABLE_LENGTH_STRUCT   longvarbinary;
SQLLEN                            indicator;
SQLCHAR                         * buffer;
buffer = (SQLCHAR*)malloc(128);
longvarbinary.arr = buffer;
longvarbinary.len = 128;
buffer[0] = 0x1;
buffer[1] = 0x2;
buffer[2] = 0x3;
indicator = 3;
SQLBindParameter( stmt,
                  1,
                  SQL_PARAM_INPUT,
                  SQL_C_LONGVARBINARY,
                  SQL_VARBINARY,
                  128,
                  0,
                  &longvarbinary,
                  0,
                  &indicator );
```

SQL_C_LONGVARCHAR

It has SQLCHAR buffer size and SQLCHAR buffer pointer.

SQLCHAR buffer has a string data.

The actual length of the string data is transferred through an indicator.

```
SQL_LONG_VARIABLE_LENGTH_STRUCT   longvarchar;
SQLLEN                            indicator;
SQLCHAR                         * buffer;
```

```
buffer = (SQLCHAR*)malloc(128);
longvarchar.arr = buffer;
longvarchar.len = 128;
buffer[0] = 'A';
buffer[1] = 'B';
buffer[2] = 'C';
indicator = 3;
SQLBindParameter( stmt,
                  1,
                  SQL_PARAM_INPUT,
                  SQL_C_LONGVARCHAR,
                  SQL_VARCHAR,
                  128,
                  0,
                  &longvarchar,
                  0,
                  &indicator );
```

SQL_C_REFCURSOR

It can use REF CURSOR through the allocated SQLHSTMT.

```
SQLHSTMT stmt    = SQL_NULL_HSTMT;
SQLHSTMT refstmt = SQL_NULL_HSTMT;
SQLLEN   refind  = 0;
SQLCHAR  job[10];
SQLLEN   jobind  = 0;
SQLAllocHandle( SQL_HANDLE_STMT,
                dbc,
                &stmt );
SQLAllocHandle( SQL_HANDLE_STMT,
                dbc,
                &refstmt );
SQLPrepare( stmt,
            (SQLCHAR*)"CALL EMP_BY_JOB_REFCURSOR(?, ?)",
            SQL_NTS );
SQLBindParameter( stmt,
                  1,
                  SQL_PARAM_INPUT,
                  SQL_C_CHAR,
                  SQL_VARCHAR,
```

```
                  10,
                  0,
                  job,
                  sizeof(job),
                  &jobind );
SQLBindParameter( stmt,
                  2,
                  SQL_PARAM_OUTPUT,
                  SQL_C_REFCURSOR,
                  SQL_REFCURSOR,
                  0,
                  0,
                  refstmt,
                  0,
                  &refind );
strcpy( (char*)job, "SALESMAN" );
jobind = strlen((char*)job);
SQLExecute( stmt );
SQLFetch( refstmt );
```

SQL_C_TYPE_TIME_WITH_TIMEZONE

It can describe TIME WITH TIMEZONE.

- hour
  - hour
- minute
  - minute
- second
  - second
- fraction
  - 1/1,000,000,000 second
    - It is describable maximum 1/1,000,000 second in GOLDILOCKS.
    - The micro second should be converted to the nano second.
- timezone_hour
  - It is the time of TIMEZONE.
- timezone_minute
  - It is the minute of TIMEZONE.

```
SQL_TIME_WITH_TIMEZONE_STRUCT timetz;
SQLLEN                        indicator = 0;
```

```
timetz.hour            = 10;
timetz.minute          = 20;
timetz.second          = 30;
timetz.fraction        = 123456000;
timetz.timezone_hour   = 9;
timetz.timezone_minute = 0;
SQLBindParameter( stmt,
                  1,
                  SQL_PARAM_INPUT,
                  SQL_C_TYPE_TIME_WITH_TIMEZONE,
                  SQL_TYPE_TIME_WITH_TIMEZONE,
                  0,
                  6,
                  &timetz,
                  0,
                  &indicator );
```

SQL_C_TYPE_TIMESTAMP_WITH_TIMEZONE

It can describe TIMESTAMP WITH TIMEZONE.

- year
  - year
- month
  - month
- day
  - day
- hour
  - hour
- minute
  - minute
- second
  - second
- fraction
  - 1/1,000,000,000 second
    - It is describable maximum 1/1,000,000 second in GOLDILOCKS.
    - The micro second should be converted to the nano second.
- timezone_hour
  - it is the hour of TIMEZONE.
- timezone_minute
  - It is the minute of TIMEZONE.

```
SQL_TIMESTAMP_WITH_TIMEZONE_STRUCT timestamptz;
SQLLEN                             indicator = 0;
timestamptz.year            = 2010;
timestamptz.month           = 9;
timestamptz.day             = 1;
timestamptz.hour            = 10;
timestamptz.minute          = 20;
timestamptz.second          = 30;
timestamptz.fraction        = 123456000;
timestamptz.timezone_hour   = 9;
timestamptz.timezone_minute = 0;
SQLBindParameter( stmt,
                  1,
                  SQL_PARAM_INPUT,
                  SQL_C_TYPE_TIMESTAMP_WITH_TIMEZONE,
                  SQL_TYPE_TIMESTAMP_WITH_TIMEZONE,
                  0,
                  6,
                  &timestamptz,
                  0,
                  &indicator );
```

SQL_C_WLONGVARCHAR

It has SQLWCHAR buffer size and SQLWCHAR buffer pointer.

SQLWCHAR buffer has a unicode string data.

The actual length of the unicode string data is transferred through an indicator.

```
SQL_WLONG_VARIABLE_LENGTH_STRUCT  wlongvarchar;
SQLLEN                            indicator;
SQLWCHAR                         * buffer;
buffer = (SQLWCHAR*)malloc(128 * sizeof(SQLWCHAR));
wlongvarchar.arr = buffer;
wlongvarchar.len = 128;
buffer[0] = 'A';
buffer[1] = 'B';
buffer[2] = 'C';
indicator = 3 * sizeof(SQLWCHAR);
SQLBindParameter( stmt,
                  1,
```

```
                SQL_PARAM_INPUT,
                SQL_C_WLONGVARCHAR,
                SQL_VARCHAR,
                128,
                0,
                &wlongvarchar,
                0,
                &indicator );
```

SQL Data Type

Table 31-7 GOLDILOCKS SQL data type

SQL type identifier	GOLDILOCKS data type	GOLDILOCKS type description
SQL_BOOLEAN	BOOLEAN	It stores TRUE or FALSE.
SQL_REFCURSOR	REF CURSOR	It is the cursor variable of PSM.
SQL_TYPE_TIME_WITH_TIME ZONE	TIME(p) WITH TIME ZO NE	It is TIME WITH TIMEZONE. p of precision stands for the s econds precision.
SQL_TYPE_TIMESTAMP_WIT H_TIMEZONE	TIMESTAMP(p) WITH TI ME ZONE	It is TIMESTAMP WITH TIMEZONE. p of precision stands f or the seconds precision.
SQL_ROWID	ROWID	It is the record identifier.

SQL_BOOLEAN

It is BOOLEAN type of GOLDILOCKS.

SQL_REFCURSOR

It is REF CURSOR type of GOLDILOCKS PSM.

SQL_TYPE_TIME_WITH_TIMEZONE

It is TIME(p) WITH TIME ZONE type of GOLDILOCKS.

The range of fractional seconds precision is 0 ~ 6.

SQL_TYPE_TIMESTAMP_WITH_TIMEZONE

It is TIMESTAMP(p) WITH TIME ZONE type of GOLDILOCKS.

The range of fractional seconds precision is 0 ~ 6.

SQL_ROWID

It is ROWID type of GOLDILOCKS.

```
SQLCHAR  rowid[24];
SQLLEN   indicator = 0;
strcpy( (char*)rowid, "AAAAAAAAYfCAACAAAADiAAA" );
indicator = strlen((char*)rowid);
SQLBindParameter( stmt,
                  1,
                  SQL_PARAM_INPUT,
                  SQL_C_CHAR,
                  SQL_ROWID,
                  0,
                  0,
                  rowid,
                  0,
                  &indicator );
```

31.6 ODBC API References

SQLAllocConnect

Conformance

Introduced version: ODBC 1.0

Overview

SQLAllocConnect function is replaced by SQLAllocHandle function in ODBC 3.x.
For more information, refer to **SQLAllocHandle**.

Syntax

```
SQLRETURN SQLAllocConnect(
    SQLHENV   EnvironmentHandle,
    SQLHDBC * ConnectionHandlePtr);
```

Arguments

EnvironmentHandle

[Input] It is the environment handle.

ConnectionHandlePtr

[Output] It is the pointer of the connection handle to be newly allocated.

SQLAllocEnv

Conformance

Introduced version: ODBC 1.0

Overview

SQLAllocEnv function is replaced by SQLAllocHandle function in ODBC 3.x.
For more information, refer to **SQLAllocHandle**.

Syntax

```
SQLRETURN SQLAllocEnv(
    SQLHENV * EnvironmentHandlePtr);
```

Arguments

EnvironmentHandlePtr

[Output] It is the pointer of the environment handle to be newly allocated.

SQLAllocHandle

Conformance

Introduced version: ODBC 3.0
Standards compliance: ISO 92

Overview

SQLAllocHandle allocates the environment handle, the connection handle, or the statement handle.

Syntax

```
SQLRETURN SQLAllocHandle(
    SQLSMALLINT   HandleType,
    SQLHANDLE     InputHandle,
    SQLHANDLE *   OutputHandlePtr);
```

Arguments

HandleType

[Input] It is the handle type allocated by SQLAllocHandle and it should be one of SQL_HANDLE_DBC, SQL_HANDLE_ENV, SQL_HANDLE_STMT.

InputHandle

[Input] If HandleType is SQL_HANDLE_ENV, it is SQL_NULL_HANDLE.

If HandleType is SQL_HANDLE_DBC, it should be an environment handle.

If it is SQL_HANDLE_STMT, it should be a connection handle.

OutputHandlePtr

[Output] It is the newly allocated handle pointer.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_INVALID_HANDLE, SQL_ERROR

Diagnosis

SQLSTATE	Error	Description
08003	Connection not open	It is not connected and HandleType is one of SQL_HANDLE_STMT and SQL_HANDLE_DESC.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY009	Invalid use of null pointer	OutputHandlePtr argument is a null pointer.
HY010	Function sequence error	HandleType argument is SQL_HANDLE_DBC, and SQLSetEnvAttr is not called for setting SQL_ODBC_VERSION environment attribute.
HY014	Limit on the number of handles exceeded	It limits the number of allocated handles.
HY092	Invalid attribute/option identifier	HandleType argument is not one of SQL_HANDLE_ENV, SQL_HANDLE_DBC, SQL_HANDLE_STMT, SQL_HANDLE_DESC.
IM001	Driver does not support this function	HandleType argument is SQL_HANDLE_DESC.

Description

SQLAllocHandle is used to allocate the handle for environment, connection, statement, descriptor. When using SQLAllocHandle with *OutputHandlePtr, the driver will overwrite the information on the correspoid ning handle. The driver manager can not verify whether the handle in *OutputHandlePtr is already used, and it can not know the previously overwritten information.

Allocating Environment Handle

The environment handle provides the global information such as whether the connection handle is valid or active.

For requesting the environment handle, the application calls SQLAllocHandle whose HandleType is SQL_HANDLE_ENV and whose InputHandle is SQL_NULL_HANDLE. The driver allocates memory for the environment information, and passes an allocated handle to *OutputHandle argument. The application passes the value of *OutputHandle to the call requiring the environment handle argument.

After allocating the environment handle, the application should set the attribute of SQL_ATTR_ODBC_VERSIONby calling SQLSetEnvAttr. If the attribute is not set when calling SQLAllocHandle for allocating the connection handle, SQLSTATE HY010 (Function sequence error) is returned.

Allocating Connection Handle

The connection handle provides the information such as whether the statement is valid, the descriptor ha ndle is connected, or the transaction currently is opened.

For requesting the connection handle, the application calls SQLAllocHandle whose HandleType is SQL_HA NDLE_DBC. InputHandler argument is set to the environment handle returned by calling SQLAllocHandle. The driver allocates memory for the connection information, and passes an allocated handle to *OutputH andle argument. The application passes the value of *OutputHandle to the call requiring a connection ha ndle argument.

If the environment attribute of SQL_ATTR_ODBC_VERSION is not set before the calling SQLAllocHandle which allocates a connection handle, SQLSTATE HY010(Function sequence error) is returned.

Allocating Statement Handle

The statement handle provides the information such as the error message, the cursor name and the SQL s tatement processing status.

For requesting the statement handle, the application connects to the data source and then calls SQLAlloc Handle before sending the SQL statement. In this call, HandleType should be set to SQL_HANDLE_STMT a nd InputHandler should be set to the connection handle returned by calling SQLAllocHandle. The driver al locates memory for the statement information, and passes the allocated handle to *OutputHandle argum ent. The application passes the value of *OutputHandle to the call requiring a statement handle argumen t.

If the statement handle is allocated, the driver automatically allocates four descriptor sets, and these desc riptor handles are allocated to the statement attribute of SQL_ATTR_APP_ROW_DESC, SQL_ATTR_APP_P ARAM_DESC, SQL_ATTR_IMP_ROW_DESC and SQL_ATTR_IMP_PARAM_DESC. This is called as implicit d escriptor allocation.

SQLAllocStmt

Conformance

Introduced version: ODBC 1.0

Overview

SQLAllocStmt function is replaced by SQLAllocHandle function in ODBC 3.x.
For more information, refer to **SQLAllocHandle**.

Syntax

```
SQLRETURN SQLAllocStmt(
    SQLHDBC    ConnectionHandle,
    SQLHSTMT * StatementHandlePtr);
```

Arguments

ConnectionHandle

[Input] It is the connection handle.

StatementHandlePtr

[Output] It is the pointer of the statement handle to be newly allocated.

SQLBindCol

Conformance

Introduced version: ODBC 1.0
Standards compliance: ISO 92

Overview

SQLBindCol binds the application data buffer to the columns in the result set.

Syntax

```
SQLRETURN SQLBindCol(
    SQLHSTMT       StatementHandle,
    SQLUSMALLINT   ColumnNumber,
    SQLSMALLINT    TargetType,
    SQLPOINTER     TargetValuePtr,
    SQLLEN         BufferLength,
    SQLLEN *       StrLen_or_Ind);
```

Arguments

StatementHandle

[Input] It is the statement handle

ColumnNumber

[Input] It is the column number in the result set to be bound. The number is in ascending order starting from 1.

TargetType

[Input] It is the identifier of C data type of *TargetValuePtr buffer. When retrieving data using SQLFetch, SQLFetchScroll, SQLSetPos, the driver converts the data into this type.

If TargetType is the interval data type, the default value is the interval leading precision (2), interval seconds precision (6), and it is set in each field of SQL_DESC_DATETIME_INTERVAL_PRECISION and SQL_DESC_PRECISION of ARD. If TargetType argument is SQL_C_NUMERIC, the default value is precision (38), scale (0), and it is set in each field of SQL_DESC_PRECISION and SQL_DESC_SCALE of ARD. If the default precision and scale are not appropriate, the application must explicitly set the descriptor field by calling SQLSetDescField or SQLSetDescRec.

TargetValuePtr

[Delayed Input/Output] It is the data buffer pointer to bind to the column. SQLFetch and SQLFetch Scroll return data to this buffer.

If TargetValuePtr is the null pointer, the driver releases the data buffer binding for the column. The application may release the binding of all the columns by calling SQLFreeStmt as SQL_BIND option. The application can release the data buffer binding for the column by setting TargetValuePtr argu ment as a null pointer and calling SQLBindCol. But if StrLen_or_IndPtr is valid, the length/indicator buffer for the column is still bound.

BufferLength

[Input] It is the length of *TargetValuePtr buffer in bytes.

The driver uses BufferLength in order to avoid writing beyond the end of *TargetValuePtr buffer w hen returning the variable-length data such as text or binary data. It should be noted that the drive r take into account the null terminator when returning the character data in *TargetValuePtr. So, * TargetValuePtr should include the space for a null terminator, otherwise the drive may drop the da ta. The driver assumes that the buffer is large enough to store data and it ignores BufferLength wh en returning a fixed-length data structure such as an integer or date. It is important that the applic ation allocates the large buffer for the fixed-length data, otherwise the driver can write beyond the end of the buffer.

StrLen_or_IndPtr

[Delayed Input/Output] It is the pointer to the length/indicator buffer to be bound to the column. SQLFetch and SQLFetchScroll return the value to this buffer.

SQLFetch and SQLFetchScroll return the length of data for the length/indicator buffer, SQL_NO_TO TAL and SQL_NULL_DATA. If there are two separate buffers for the length and indicator, the lengt h buffer may return all values, and the indicator buffer may only return only SQL_NULL_DATA. If St rLen_or_IndPtr is the null pointer, the value of length/indicator is not used and an error occurs whe n getting the null data.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_INVALID_HANDLE, SQL_ERROR

Diagnosis

SQLSTATE	Error	Description
07006	Restricted data type attribute vi olation	ColumnNumber argumnet is 0, and TargetType argument is neither S QL_C_BOOKMARK nor is SQL_C_VARBOOKMARK.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY003	Invalid application buffer type	The value of TargetType argument is the invalid data type.

SQLSTATE	Error	Description
HY010	Function sequence error	After calling SQLExecute and SQLExecDirect, then SQL_NEED_DATA is returned, and the function is called before sending all data-at-execution variables.
HY090	Invalid string or buffer length	The value of BufferLength argument is smaller than 0.
HYC00	Optional feature not implemented	The driver does not support the SQL data type of the column and the value of TargetType argument and the conversion. The value of ColumnNumber argument is 0, and the driver does not support bookmarks.

Description

SQLBindCol is used to bind columns in the result set to the data buffer and the length/indicator buffers in the application. The application calls SQLFetch and SQLFetchScroll to retrieve the data, and the driver returns the bound column data to the specified buffer.
The application does not bind the column, and the data is retrieved by calling SQLGetData.

Binding Column

The application calls SQLBindCol to bind the column, and passes the column number, the type, the address, the data buffer length and the address of length/indicator buffer.

Though the application binds the buffer by calling SQLBindCol, but the driver accesses them when calling SQLFetch and SQLFetchScroll, so these buffers are delayed when it is used. Therefore, the application should make the pointer set in SQLBindCol to be valid until the data is returned. If the application calls after making the pointer invalid, like when releasing the buffer, the result is not correct.

The binding remains until when it is replaced by a new binding, the column binding is released, or the statement is released.

Releasing Bound Column

To release only one bound column, set the column number to be released as ColumnNumber and call SQLBindCol by setting TargetValue to the null pointer in the application. If ColumnNumber is the column number whose binding is released, SQLBindCol continues returning SQL_SUCCESS.

To release all bound columns, call SQLFreeStmt by setting option to SQL_UNBIND in the application. Or, set SQL_DESC_COUNT field of ARD to 0 to release all bound columns.

Rebinding Column

The application can perform one of two operations to change the binding.

- A new binding is specified for an existing bound column by calling SQLBindCol.
- The offset is added to the specified buffer address by calling SQLBindCol. For more information, refer to **Binding Offsets**.

Binding Offsets

Binding offset is the value added to the address before the data and length/indicator buffers (specified in TargetValuePtr and StrLen_or_IndPtr) are dereferenced.

Using binding offset generally has the same effect as rebinding the column by calling SQLBindCol. However, the new address of the data and length/indicator buffer are specified when newly calling SQLBindCol, but binding offset does not change the address, instead it adds only the offsets to the address. The application can specify a new offset anytime, and the offset is always added to the originally bound address. Especially, if the offset is set to 0 or the statement attribute is set to NULL pointer, the driver uses the originally bound address.

The sum of the originally bound address and the offset should be a valid address, but the offset address to be added does not have to be valid.

Binding Array

If the row set size (the value of SQL_ATTR_ROW_ARRAY_SIZE statement attribute) is bigger than 1, the application binds a buffer array instead of a single buffer.

The application can bind an array in two ways as follows.

- Bind an array to each column. Each data structure (array) contains data for a single column, so it is called as **Column-wise Binding**.
- Define as structure containing the entire row data and bind the array of these structures. Each data structure contains data for a single row. so it is called as **Row-wise Binding**.

Each buffer array should have at least as many elements as the row set size.

Column-wise Binding

The application binds the separate data and length/indicator array to each column in column-wise binding.

The application should set SQL_ATTR_ROW_BIND_TYPE statement attribute to SQL_BIND_BY_COLUMN (default value) to use the column-wise binding. Then, the application performs the following processes for the columns to be bound.

1. It allocates the data buffer array.
2. It allocates the array of length/indicator buffer.

If the application directly records to the descriptors when using the column-wise binding, the separate arrays can be used for length and indicator data.

3. It calls SQLBindCol together with the following arguments.

- TargetType is the type of each element of the data buffer array.
- TargetValuePtr is the address of the data buffer array.
- BufferLength is the size of each element of the data buffer array. BufferLength is ignored when the data is fixed length data.
- StrLen_or_IndPtr is the address of the length/indicator array.

Row-wise Binding

The application defines a structure which contains the data and length/indicator buffer of each column to be bound in row-wise binding.

The application performs the following processes to use row-wise binding.

1. It defines a structure which contains a row(including both of data and the length/indicator buffer) and allocates an array of the structures.

If the application directly records to the descriptors when using the row-wise binding, the separate fields can be used for length and indicator data.

2. SQL_ATTR_ROW_BIND_TYPE statement attribute sets the size of structure including a data row or as the buffer instance size for the result columns to be bound. The length should include the space and structure of all bound columns and the padding of the buffer. It should guarantee to point to the starting position in the same column of the next line when the address of the bound column is increased by the specified length. ANSI C guarantees it by using the sizeof operator.

3. It calls SQLBindCol together with the following arguments for each column to be bound.

- TargetType is the type of the data buffer member to be bound to the column.
- TargetValuePtr is the address of the data buffer member in the first array element.
- BufferLength is the size of the data buffer member.
- StrLen_or_IndPtr the address of the length/indicator member to be bound.

Buffer Address

The buffer address is an actual address of the data or the length/indicator buffer. The driver calculates th e buffer address before writing to the buffer (such as the data collect). It is calculated by the following for mula, which uses the address, binding offset, row number specified in TargetValuePtr and StrLen_or_Ind Ptr.

```
Bound Address + Binding Offset + ((Row Number -1) x Element Size )
```

The following table describes the definitions the formula's variables.

Table 31-8 Formula's variable

Variable	Description
Bound address	The address of data buffer is specified in TargetValuePtr argument of SQLBindCol. The address of length/indicator buffer is specified in StrLen_of_IndPtr argument of SQ LBindCol. If the binding address is 0, the data value is not returned, even though the calculated address is not 0.
Binding offset	If row-wise binding is used, this value is stored in the address specified with the SQL_ ATTR_ROW_BIND_OFFSET_PTR statement attribute. If column-wise binding is used or the SQL_ATTR_ROW_BIND_OFFSET_PTR statement attribute is a NULL pointer, then the binding offset is 0.
Row number	It is 1-based number of the row in the row set. When fetching a single row, generally the row number is 1.
Element size	It is the element size of binding array. If column-wise binding is used, it is sizeof (SQLLEN) for the length/indicator buffer. Th e element size of the data buffer of variable length data type is the value of BufferLen gth argument of SQLBindCol, and the element size of the data buffer of fixed length data type is the size of the data type. If row-wise binding is used, the element size of both the data and the length/indicato r buffer are the value of the SQL_ATTR_ROW_BIND_TYPE statement.

Descriptors and SQLBindCol

This chapter describes how SQLBindCol interacts with descriptors.

Calling SQLBindCol for a single statement may affect other statements. It occurs when ARD rela ted to the statement is explicitly allocated and it is related to other statements. The modification s for the descriptor affects all statements related the descriptor because SQLBindCol modifies th e descriptor. If it is not the required behavior, the application should release the relationship bet ween the descriptor and other statements before calling SQLBindCol.

Argument Mapping

Notionally, SQLBindCol performs the following processes in order.

1. It calls SQLGetStmtAttr to obtain ARD handle.

2. It calls SQLGetDescField to obtain the descriptor of SQL_DESC_COUNT field, and if the value in the ColumnNumber argument exceeds the value of SQL_DESC_COUNT, it calls SQLSetDescField to increase the value of SQL_DESC_COUNT to ColumnNumber.

3. It calls SQLSetDescField multiple times to assign values to the following fields of ARD.

- It sets SQL_DESC_TYPE and SQL_DESC_CONCISE_TYPE to the value of TargetType.
  - Except when TargetType is one of the concise identifiers of a datetime or interval subtype, it respectively sets SQL_DESC_TYPE to SQL_DATETIME or SQL_INTERVAL. It sets SQL_DESC_CONCISE_TYPE to the concise identifier; and sets SQL_DESC_DATETIME_INTERVAL_CODE to the corresponding datetime or interval subcode.
- It appropriately sets one or more of SQL_DESC_LENGTH, SQL_DESC_PRECISION, SQL_DESC_SCALE, and SQL_DESC_DATETIME_INTERVAL_PRECISION, for TargetType.
- It sets the SQL_DESC_OCTET_LENGTH field to the value of BufferLength.
- It sets the SQL_DESC_DATA_PTR field to the value of TargetValue.
- It sets the SQL_DESC_INDICATOR_PTR field to the value of StrLen_or_Ind.
- It sets the SQL_DESC_OCTET_LENGTH_PTR field to the value of StrLen_or_Ind.

The variable referenced by StrLen_or_Ind argument is used for both indicator and length information. If the value of column when fetching is null, it stores SQL_NULL_DATA in this variable. Otherwise, it stores the data length in this variable.
Entering a null pointer, and the value of column when fetching is null, then the fetch fails because it cannot return SQL_NULL_DATA.

If SQLBindCol fails, the contents of the descriptor fields which will be set in ARD are not defined, and the value of SQL_DESC_COUNT field of ARD is not updated.

Implicit Initialization of COUNT Field

SQLBindCol sets SQL_DESC_COUNT to the value of the ColumnNumber only when ColumnNumber increase the value of SQL_DESC_COUNT. If the value in the TargetValuePtr argument is a null pointer and the value in the ColumnNumber argument is equal to SQL_DESC_COUNT (when releasing the highest bound column), then SQL_DESC_COUNT is set to the number of the highest remaining bound column.

Caution for SQL_DEFAULT

The application should determine the correct length and starting point of the data in the application buffer to successfully retrieve column data. When the application explicitly specifies an TargetType, application errors are easily detected.

However, when the application specifies a TargetType of SQL_DEFAULT, SQLBindCol can be applied to a column of a different data type from the one data type intended by the application, either from changes to the metadata or by applying the code to a different column. In this case, the application may not alway s determine the start or length of the fetched column data. This may lead to unreported data errors or m emory violations.

SQLBindParameter

Conformance

Introduced version: ODBC 2.0
Standards compliance: ODBC

Overview

SQLBindParameter binds the buffer to the parameter marker of SQL statement.

Syntax

```
SQLRETURN SQLBindParameter(
    SQLHSTMT        StatementHandle,
    SQLUSMALLINT    ParameterNumber,
    SQLSMALLINT     InputOutputType,
    SQLSMALLINT     ValueType,
    SQLSMALLINT     ParameterType,
    SQLULEN         ColumnSize,
    SQLSMALLINT     DecimalDigits,
    SQLPOINTER      ParameterValuePtr,
    SQLLEN          BufferLength,
    SQLLEN *        StrLen_or_IndPtr);
```

Arguments

StatementHandle

[Input] It is the statement handle.

ParameterNumber

[Input] It is the parameter number which is increased sequentially from 1.

InputOutputType

[Input] It is the parameter type.

ValueType

[Input] It is the C data type of the parameter.

ParameterType

[Input] It is the SQL data type of the parameter.

ColumnSize

[Input] It is the size of column or expression of the parameter marker.

DecimalDigits

[Input] It is the number of decimal point of column or expression of the parameter marker.

ParameterValuePtr

[Delayed Input] It is the data buffer pointer of the parameter.

BufferLength

[Input/Output] It is the byte length of ParameterValuePtr buffer.

StrLen_or_IndPtr

[Delayed Input] It is the pointer to the length/indicator of the parameter.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

Table 31-9 SQLSTATE values

SQLSTATE	Error	Description
07006	Restricted data type attribute violation	ValueType argument data type can not be converted into ParameterType argument data type.
07009	Invalid descriptor index	The value of ParameterNumber argument is smaller than 1.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY003	Invalid application buffer type	The value of ValueType argument is not a valid C data type.
HY004	Invalid SQL data type	The value of ParameterType argument is not a valid SQL data type.
HY009	Invalid argument value	ParameterValuePtr argument and StrLen_or_IndPtr argument are the NULL pointer, and InputOutputType argument is not SQL_PARAM_OUTPUT. InputOutputType argument is SQL_PARAM_OUTPUT, and ParameterValuePtr argument is the NULL pointer, and C type is a character or binary, and BufferLength is greater than 0.
HY010	Function sequence error	SQL_NEED_DATA is returned after calling SQLExecute, SQLExecDirect, and this function is called before sending all data-at-execution variables.
HY021	Inconsistent descriptor i	Descriptor information is inconsistent when the integrity is checked.

SQLSTATE	Error	Description
	nformation	
HY090	Invalid string or buffer l ength	The value of BufferLength is smaller than 0.
HY104	Invalid precision or scal e value	The value specified in ColumnSize and DecimalDigits is beyond the SQL da ta support range of ParameterType argument.
HY105	Invalid parameter type	The value of InputOutType argument is not valid.
HYC00	Optional feature not im plemented	The driver does not support the conversion of values of ValueType argum entand ParameterType argument. The value of ParameterType argument is valid but the driver does not sup port.

Description

The application calls SQLBindParameter to bind each parameter marker in an SQL statement. Bindings re main valid until the application calls SQLBindParameter again, calls SQLFreeStmt with the SQL_RESET_PA RAMS option, or calls SQLSetDescField to set the SQL_DESC_COUNT header field of the APD to 0.

ParameterNumber Argument

If ParameterNumber is bigger than the value of SQL_DESC_COUNT when calling SQLBindParameter, SQL SetDescField is called to increase the value of SQL_DESC_COUNT to ParameterNumber.

InputOutputType Argument

The InputOutputType argument specifies the type of the parameter. This argument sets the SQL_DESC_P ARAMETER_TYPE field of the IPD.

InputOutputType argument is one of the followings.

- SQL_PARAM_INPUT: The parameter marks a parameter in an SQL statement, not in a procedure nor i n SELECT INTO statement. For example, the parameters in INSERT INTO Employee VALUES (?, ?, ?) ar e input parameters.
  - When the statement is executed, the driver sends data for the parameter, and the *ParameterVal uePtr buffer should contain a valid input value, or the *StrLen_or_IndPtr buffer should contain S QL_NULL_DATA, SQL_DATA_AT_EXEC, or the result of the SQL_LEN_DATA_AT_EXEC macro.
- SQL_PARAM_INPUT_OUTPUT: The parameter marks an input/output parameter in a procedure.
  - When the statement is executed, the driver sends data for the parameter and the *ParameterVal uePtr buffer should contain a valid input value, or the *StrLen_or_IndPtr buffer should contain S QL_NULL_DATA, SQL_DATA_AT_EXEC, or the result of the SQL_LEN_DATA_AT_EXEC macro.
  - After the statement is executed, the driver returns data to the parameter of the application. If the data source does not return a value to an input/output parameter, the driver sets SQL_NULL_DAT A in *StrLen_or_IndPtr buffer.

- SQL_PARAM_OUTPUT: The parameter marks the return value of a procedure, or an output paramete r in a procedure or SELECT INTO statement. For example, the parameter in SELECT ID INTO ? FROM E mployee WHERE NAME = 'Paul' is an output parameter which returns the ID.
  - After the statement is executed, the driver returns data to the parameter, if the ParameterValueP tr and StrLen_or_IndPtr arguments of an application are not null pointers. Otherwise, the driver d iscards the output value.
  - If the data source can not return a value to an output parameter, the driver sets SQL_NULL_DAT A in *StrLen_or_IndPtr buffer.

ValueType Argument

ValueType argument specifies C data type of the parameter. It sets the values of SQL_DESC_TYPE, SQL_D ESC_CONCISE_TYPE, SQL_DESC_DATETIME_INTERVAL_CODE fields of APD.

When the ValueType argument is an interval data type,

- SQL_DESC_TYPE field of APD ParameterNumber record is set to SQL INTERVAL.
- SQL_DESC_CONCISE_TYPE field is set to concise interval data type.
- SQL_DESC_DATETIME_INTERVAL_CODE field is set to subcode of a specific interval data.
- The default value of interval leading precision is (2), and the default value of interval seconds precisio n is (6), and they are respectively set in SQL_DESC_DATETIME_INTERVAL_PRECISION and SQL_DESC _PRECISION fields of APD.
- If the default precision or scale is not appropriate, the application should explicitly set the descriptor fi eld by calling SQLSetDescField or SQLSetDescRec.

When the ValueType argument is a datetime data type,

- SQL_DESC_TYPE field of ParameterNumber record in APD is set to SQL_DATETIME.
- SQL_DESC_CONCISE_TYPE field is set to the concise date C data type.
- SQL_DESC_DATETIME_INTERVAL_CODE field is set to the sub code of the specific datetime data.

When the ValueType argument is an SQL_C_NUMERIC data type,

- The default precision is (38), the default scale is (0), and they are respectively set in SQL_DESC_PRECI SION, SQL_DESC_SCALE fields of APD.
- If the default precision or scale is not appropriate, the application should explicitly set the descriptor fi eld by calling SQLSetDescField or SQLSetDescRec.

ParameterType Argument

ParameterType specifies the SQL data type of the parameter. It sets the values of SQL_DESC_TYPE, SQL_ DESC_CONCISE_TYPE, SQL_DESC_DATETIME_INTERVAL_CODE fields of IPD.

When the ParameterType argument is a datetime data type,

- SQL_DESC_TYPE field of IPD is set to SQL_DATETIME.
- SQL_DESC_CONCISE_TYPE field is set to the concise datetime SQL data type.
- SQL_DESC_DATETIME_INTERVAL_CODE field is set to the sub code of the specific datetime data.

When the ParameterType argument is a interval data type,

- SQL_DESC_TYPE field of IPD is set to SQL_INTERVAL.
- SQL_DESC_CONCISE_TYPE field is set to the concise SQL interval data type.
- SQL_DESC_DATETIME_INTERVAL_CODE field is set to the sub code of the specific interval data.
- The interval leading precision is set in SQL_DESC_DATETIME_INTERVAL_PRECISION field of IPD, and t he interval second precision is set in SQL_DESC_PRECISION field of IPD.
- If the defaults of SQL_DESC_DATETIME_INTERVAL_PRECISION and SQL_DESC_PRECISION are not ap propriate, the application sets it by calling SQLSetDescField.

When the ParameterType argument is an SQL_NUMERIC data type,

- The default precision is (38), the default scale is (0), and they are respectively set in SQL_DESC_PRECI SION and SQL_DESC_SCALE fields of IPD.
- If the default precision or scale is not appropriate, the application should explicitly set the descriptors field by calling SQLSetDescField or SQLSetDescRec.

ColumnSize Argument

ColumnSize argument specifies the size of the column or expression corresponding the parameter marker. It sets different fields of the IPD depending on SQL data type of ParameterType.

- If ParameterType is SQL_CHAR, SQL_VARCHAR, SQL_LONGVARCHAR, SQL_BINARY, SQL_VARBINAR Y, SQL_LONGVARBINARY, concise SQL datetime, or interval data type, it sets the value of ColumnSiz e in SQL_DESC_LENGTH field of IPD.
- If ParameterType is SQL_DECIMAL, SQL_NUMERIC, SQL_FLOAT, SQL_REAL, or SQL_DOUBLE, it sets t he value of ColumnSize in SQL_DESC_PRECISION field of IPD.
- For other data types, ColumnSize argument is ignored.

DecimalDigit Argument

- If ParameterType argument is SQL_TYPE_TIME, SQL_TYPE_TIMESTAMP, SQL_INTERVAL_SECOND, S QL_INTERVAL_DAY_TO_SECOND, SQL_INTERVAL_HOUR_TO_SECOND, or SQL_INTERVAL_MINUTE_ TO_SECOND, it sets the value of DecimalDigits in SQL_DESC_PRECISION field of IPD.
- If ParameterType argument is SQL_NUMERIC or SQL_DECIMAL, it sets the value of DecimalDigits in S QL_DESC_SCALE field of IPD.
- For other data types, DecimalDigits argument is ignored.

ParameterValuePtr Argument

When calling SQLExecute and SQLExecDirect, ParameterValuePtr points to the actual data for the param eter. The data type should be in a form specified by ValueType argument. This argument sets the SQL_DE SC_DATA_PTR field of the APD.

If *StrLen_or_IndPtr is the result of the SQL_LEN_DATA_AT_EXEC (length) macro or SQL_DATA_AT_EXE C, then ParameterValuePtr is an application-defined pointer value which is related to the parameter. It is r eturned to the application through SQLParamData.
For example, ParameterValuePtr might be a non-zero token such as a parameter number, a pointer to da ta, or a pointer to a structure that the application used to bind input parameters.

If InputOutputType argument is SQL_PARAM_INPUT_OUTPUT or SQL_PARAM_OUTPUT, ParameterValue Ptr should be a buffer pointer in which the output value is stored.

If the value in the SQL_ATTR_PARAMSET_SIZE statement attribute is bigger than 1, ParameterValuePtr p oints to an array. A single SQL statement processes the complete array of input values for an input or inp ut/output parameter and returns an array of output values for an input/output or output parameter.

BufferLength Argument

For character and binary C data, the BufferLength argument specifies the length of the *ParameterValue Ptr buffer (if the value in the SQL_ATTR_PARAMSET_SIZE statement attribute is 1), or specifies the length of the element in the *ParameterValuePtr array(if the value in the SQL_ATTR_PARAMSET_SIZE statement attribute is bigger than 1).

Both when input and output, BufferLength is used to determine the position in the array of *ParameterV aluePtr for. It sets SQL_DESC_OCTET_LENGTH field of APD.

For an input/output parameter and an output parameter, BufferLength is used to determine whether or not to truncate the output.

- For character C data, if the number of bytes available to return is equal to or bigger than BufferLengt h, the data in *ParameterValuePtr is truncated to BufferLength minus 1 bytes and it is null-terminate d.
- For binary C data, if the number of bytes available to return is equal to or bigger than BufferLength, t he data in *ParameterValuePtr is truncated to BufferLength bytes.
- For other C data types, the BufferLength argument is ignored.

StrLen_or_IndPtr Argument

The StrLen_or_IndPtr argument contains one of the followings when SQLExecute or SQLExecDirect is call ed. (This argument sets the SQL_DESC_OCTET_LENGTH_PTR and SQL_DESC_INDICATOR_PTR of APD.)

- The length of the parameter value stored in *ParameterValuePtr. This is ignored except for character or binary C data.
- SQL_NTS: The parameter value is a null-terminated string.
- SQL_NULL_DATA: The parameter value is NULL.
- The result of the SQL_LEN_DATA_AT_EXEC(length) macro: The data for the parameter will be sent when performing SQLPutData.
    - If the ParameterType argument is SQL_LONGVARBINARY, SQL_LONGVARCHAR, or a long, and SQL_NEED_LONG_DATA_LEN information of SQLGetInfo returns "Y", then the length is the number of bytes of data to be sent for the parameter
    - If SQL_NEED_LONG_DATA_LEN information of SQLGetInfo is "N", length should be a nonnegative value and is ignored.
    - For example, to specify 10,000 bytes of SQL_LONGVARCAHR parameter data to be sent by calling SQLPutData for multiple times, *StrLen_or_IndPtr should be set to SQL_LEN_DATA_AT_EXEC(10000).
- SQL_DATA_AT_EXEC: The data for the parameter will be sent when performing SQLPutData.

If StrLen_or_IndPtr is a null pointer, the driver assumes that all input parameter values are non-NULL and that character and binary data is null-terminated. If InputOutputType is SQL_PARAM_OUTPUT, and ParameterValuePtr and StrLen_or_IndPtr are both null pointers, the driver discards the output value.

If the InputOutputType argument is SQL_PARAM_INPUT_OUTPUT, SQL_PARAM_OUTPUT, then StrLen_or_IndPtr points to SQL_NULL_DATA, the number of bytes available to return in *ParameterValuePtr (excluding the null-termination byte of character data), or SQL_NO_TOTAL (if the number of bytes available to return cannot be determined).

If the value in the SQL_ATTR_PARAMSET_SIZE statement attribute is greater than 1, StrLen_or_IndPtr points to an array of SQLLEN values.

Passing Parameter Values

An application can pass the value for a parameter by calling the *ParameterValuePtr buffer or multiple SQLPutData. Parameters whose data is passed through SQLPutData are known as data-at-execution parameters. These are typically used to send data for SQL_LONGVARBINARY and SQL_LONGVARCHAR parameters, and can be mixed with other parameters.

The application should perform the following process to pass the parameter values.

1. It calls SQLBindParameter for each parameter to bind buffers for the parameter's value (ParameterValuePtr argument) and length/indicator (StrLen_or_IndPtr argument). For data-at-execution parameters, ParameterValuePtr is an application-defined pointer value such as a parameter number or a pointer to data. The value will be returned later and can be used to identify the parameter.

2. It sets values for an input parameter or an input/output parameter in the *ParameterValuePtr and *StrLen_or_IndPtr buffers.

- For normal parameters, the application inputs the parameter value in the *ParameterValuePtr buffer and the length of that value in the *StrLen_or_IndPtr buffer.
- For data-at-execution parameters, the application inputs the result of the SQL_LEN_DATA_AT_EXEC (length) macro (when calling an ODBC 2.0 driver) in the *StrLen_or_IndPtr buffer.

3. It calls SQLExecute or SQLExecDirect to execute the SQL statement.

- If data-at-execution parameters do not exist, the process is complete.
- If any data-at-execution parameters exist, the function returns SQL_NEED_DATA.

4. It calls SQLParamData to retrieve the application-defined value specified in the ParameterValuePtr argument of SQLBindParameter for the first data-at-execution parameter to be processed. SQLParamData returns SQL_NEED_DATA.

Although data-at-execution parameters resemble data-at-execution columns, the value returned by SQLParamData is different for each.

Data-at-execution parameters are parameters in an SQL statement for which data will be sent toSQLPutData when the statement is executed together with SQLExecDirect or SQLExecute. They are bound with SQLBindParameter.

The value returned by SQLParamData is a pointer value passed to the ParameterValuePtr argument of SQLBindParameter. Data-at-execution columns are columns in a rowset for which data will be sent when a row is updated or added with SQLBulkOperations or updated with SQLSetPos. They are bound with SQLBindCol. The value returned by SQLParamData is the address of the row in the TargetValuePtr* buffer (set by a call to **SQLBindCol) which is to be processed.

5. It calls SQLPutData for one or more times to send data for the parameter. One or more calls are required if the data value is bigger than what is specified in the *ParameterValuePtr buffer of SQLPutData. Multiple SQLPutData calls for the same parameter are allowed only when sending character C data to a column with a character, binary, or data source-specific data type or when sending binary C data to a column with a character, binary, or data source-specific data type.

6. It calls SQLParamData again to signal that all data has been sent for the parameter.

- If one or more data-at-execution parameters exist, SQLParamData returns SQL_NEED_DATA and processes the application-defined value for the next data-at-execution parameter. The application repeats steps 4 and 5.
- If data-at-execution parameter does not exist, the process is complete. If the statement was successfully executed, SQLParamData returns SQL_SUCCESS or SQL_SUCCESS_WITH_INFO. If the execution failed, it returns SQL_ERROR. At this point, SQLParamData can be returned by any SQLSTATE. It can also be returned by SQLExecDirect or SQLExecute which is used to execute the statement.
- Output values for any input/output or output parameters are available in the *ParameterValuePtr an

d *StrLen_or_IndPtr buffers after the application retrieves all result sets generated by the statement.

Calling SQLExecute or SQLExecDirect puts the statement in an SQL_NEED_DATA state. At this point, the application can call only SQLCancel, SQLGetDiagField, SQLGetDiagRec, SQLGetFunctions, SQLParamData, or SQLPutData together with the statement or the connection handle related to the statement.

If it calls any other function for the statement or the connection related to the statement, the function returns SQLSTATE HY010(Function sequence error). The statement is released from the SQL_NEED_DATA state when SQLParamData or SQLPutData returns an error, SQLParamData returns SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, or the statement is canceled.

If the application calls SQLCancel while the driver still needs data for data-at-execution parameters, the driver cancels statement execution. Then the application can call SQLExecute or SQLExecDirect again.

Using Parameter Array

An application prepares a statement together with parameter markers and passes it in an array of parameters in the following two ways.

- One method is for the driver to rely on the array-processing capabilities, in this case the entire statement with the array of parameters is treated as one atomic unit. Oracle is an example of a data source which supports array processing capabilities.
- The other method is for the driver to generate a batch of SQL statements. Each set of parameters in the parameter array is allocated per an SQL statement and is made into a batch, then the batch is executed. Arrays of parameters can not be used in an UPDATE WHERE CURRENT OF statement.

When processing the parameter array, the number of each result sets/ rows are available per each parameter set or the number of each result sets/ rows are available in whole. The SQL_PARAM_ARRAY_ROW_COUNTS option in SQLGetInfo indicates whether the number of rows are available for each set of parameters(SQL_PARC_BATCH) or only a single row is available (SQL_PARC_NO_BATCH).

The SQL_PARAM_ARRAY_SELECTS option in SQLGetInfo indicates whether a result set is available for each set of parameters (SQL_PAS_BATCH) or is available only in one result set(SQL_PAS_NO_BATCH). If the driver does not allow a result set-generating statement to be executed together with an array of parameters, SQL_PARAM_ARRAY_SELECTS returns SQL_PAS_NO_SELECT.
For more information, refer to **SQLGetInfo**.

To support the parameter array, the SQL_ATTR_PARAMSET_SIZE statement attribute is set to specify the number of values for each parameter. If the field is bigger than 1, the SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, and SQL_DESC_OCTET_LENGTH_PTR fields of the APD should point to arrays. The number of elements in each array is equal to the value of SQL_ATTR_PARAMSET_SIZE.

The SQL_DESC_ROWS_PROCESSED_PTR field of the APD points to a buffer which contains the number o f sets of parameters which have been processed, including error sets. As like each processed parameters s et, the driver stores a new value in the buffer. Any number will not be returned if this is a null pointer. When arrays of parameters are used, the value pointed to by the SQL_DESC_ROWS_PROCESSED_PTR fiel d of the APD is generated even when SQL_ERROR is returned by the setting function. If SQL_NEED_DAT A is returned, the value pointed to by the SQL_DESC_ROWS_PROCESSED_PTR field of the APD is set to t he set of parameters which is being processed.

Binding Column-wise Parameter

For column-wise binding, the application binds the separate parameter and length/indicator arrays to eac h parameter.

For column-wise binding, the application firstly sets the SQL_ATTR_PARAM_BIND_TYPE statement attrib ute to SQL_PARAM_BIND_BY_COLUMN. (It is the default value.) The application performs the following processes to use column-wise binding.

1. It allocates the parameter buffer array.

2. It allocates the length/indicator buffer array.

If the application directly records to the descriptors when using the column-wise binding, the se parate arrays can be used for length and indicator data.

3. It calls SQLBindParameter together with the following arguments.

- ValueType is the C type of a single element in the parameter buffer array.
- ParameterType is the SQL type of the parameter.
- ParameterValuePtr is the address of the parameter buffer array.
- BufferLength is the size of a single element in the parameter buffer array. The BufferLength argumen t is ignored when the data is fixed-length data.

Binding Row-wise Parameter

For row-wise binding, the application defines a structure which contains parameter and length/indicator buffers for each parameter to be bound.

The application performs the following processes to use row-wise binding.

1. It defines a structure to hold a single set of parameters (including both parameter and length/indicat or buffers) and allocates an array of these structures.


If the application directly records to the descriptors when using the row-wise binding, the separate fields can be used for length and indicator data.

2. SQL_ATTR_PARAM_BIND_TYPE statement attribute sets the size of the structure which contains a single set of parameters or to the size of an instance of a buffer into which the parameters will be bound. The length should include space for all bound parameters. The length should include space for binding parameter and the structure buffer, or should be buffered to ensure the result to point to the beginning of the next parameter when the address of a bound parameter is incremented to the specified length. ANSI C guarantees it by using the sizeof operator.

3. It calls SQLBindParameter together with the following arguments for each parameter to be bound.

- ValueType is the type of the parameter buffer member to be bound to the column.
- ParameterType is the SQL type of the parameter.
- ParameterValuePtr is the address of the parameter buffer member in the first array element.
- BufferLength is the size of the parameter buffer member.
- StrLen_or_IndPtr is the address of the length/indicator member to be bound.

Error Information

If a driver does not perform parameter arrays as same as batches (the SQL_PARAM_ARRAY_ROW_COUNTS option is as same as SQL_PARC_NO_BATCH), error situations are handled as if one statement were executed.

If the driver performs parameter arrays as batches, an application can use the SQL_DESC_ARRAY_STATUS_PTR header field of the IPD to determine if a parameter of an SQL statement or a parameter in an array of parameters caused SQLExecDirect or SQLExecute to return an error.
This field contains status information for each row of parameter values. If the field represents that an error occurs, fields in the diagnostic data structure will represent the row and parameter number of the failed parameter. The number of columns in the array will be defined by the SQL_DESC_ARRAY_SIZE header field in the APD, and it can be set by the SQL_ATTR_PARAMSET_SIZE statement attribute.


The SQL_DESC_ARRAY_STATUS_PTR header field in the APD is used to ignore parameters. For more information about ignoring parameters, refer to **Ignoring Parameter Set**.

When SQLExecute or SQLExecDirect returns SQL_ERROR, the elements in the array pointed to by the SQL_DESC_ARRAY_STATUS_PTR field in the IPD contain SQL_PARAM_ERROR, SQL_PARAM_SUCCESS, SQL_PARAM_SUCCESS_WITH_INFO, SQL_PARAM_UNUSED, or SQL_PARAM_DIAG_UNAVAILABLE.

For each element in this array, the diagnostic data structure contains one or more status records. The SQL_DIAG_ROW_NUMBER field of the structure represents the row number of the parameter values which c

aused the error. If it is possible to determine the particular parameter in a row of parameters which cause d the error, the parameter number will be entered in the SQL_DIAG_COLUMN_NUMBER field.

SQL_PARAM_UNUSED is set when a parameter is not used due to an error because SQLExecute or SQLEx ecDirect forcibly canceled an earlier parameter. For example, if 50 parameters exist and an error occurred while executing the 40th set of parameters which caused the cancellation by SQLExecute or SQLExecDire ct, then SQL_PARAM_UNUSED is set in the status array for parameters from 41 to 50.

SQL_PARAM_DIAG_UNAVAILABLE is set when the driver treats arrays of parameters as a single unit, so it does not generate individual error information of parameter level.

Some errors in the processing of a single set of parameters cause processing of the subsequent sets of pa rameters in the array to stop. Other errors do not affect the processing of subsequent parameters. The dri ver defines which errors will stop processing. If processing does not stop, all parameters in the array are p rocessed, SQL_SUCCESS_WITH_INFO is returned as a result of the error, and the buffer defined by SQL_A TTR_PARAMS_PROCESSED_PTR is set to the total number of sets of parameters processed which include s error sets.

ODBC behavior when an error occurs in the processing of an array of parameters is different bet ween ODBC 3.x and ODBC 2.x.

In ODBC 2.x, the function returns SQL_ERROR and stops the processing. The buffer pointed to by the pirow argument of SQLParamOptions contained the number of the error row.

In ODBC 3.x, the function returns SQL_SUCCESS_WITH_INFO, and it may stop or continue proc essing. If it continues, the buffer specified by SQL_ATTR_PARAMS_PROCESSED_PTR will be set to the value of all parameters processed, including those which resulted in an error. This change in behavior can cause problems for existing applications.

When SQLExecute or SQLExecDirect returns SQL_ERROR or SQL_NEED_DATA before completing the pro cessing of all parameter sets in a parameter array, the status array contains statuses for those parameters which have already been processed.
The location pointed to by the SQL_DESC_ROWS_PROCESSED_PTR field in the IPD contains the row num ber in the parameter array which caused the SQL_ERROR or SQL_NEED_DATA error code. When an array of parameters is sent to a SELECT statement, the availability of status array values is defined by the driver. They are available after the statement is executed or result sets are fetched.

Ignoring Parameter Set

SQL_DESC_ARRAY_STATUS_PTR field of APD can be used to indicate the binding parameter set which sh ould be ignored in SQL statement. The application should perform the following processes in order that t

he driver directly ignores one or more parameter sets during the execution.

1. It calls SQLSetDescField in order that the header field of SQL_DESC_ARRAY_STATUS_PTR of APD poi nts to an array of SQLUSMALLINT values including the status information. The field also can be set th rough SQL_ATTR_PARAM_OPERATION_PTR of Attribute argument in SQLSetStmtAttr, and it allows to set the field without a descriptor handle which is the application.

2. It sets each element of the array defined by SQL_DESC_ARRAY_STATUS_PTR of APD to one of the f ollowing two values.

- SQL_PARAM_IGNORE: The row is excluded from the execution of the statement.
- SQL_PARAM_PROCEED: The row is included in the execution of the statement.

3. It calls SQLExecDirect or SQLExecute, and executes the prepared statement. It applies the following r ules to the array defined by SQL_DESC_ARRAY_STATUS_PTR of APD.

- The pointer is set to NULL by default.
- If the pointer is NULL, all parameter set are used as if all elements are set to SQL_ROW_PROCEED.
- Setting the element to SQL_PARAM_PROCEED does not guarantee that the operation would use the specific parameter set.
- SQL_PARAM_PROCEED is defines as 0 in the header file.

An application can set SQL_DESC_ARRAY_STATUS_PTR field of APD to refer to the same array as SQL_D ESC_ARRAY_STATUS_PTR field of IRD. It is very useful when binding the parameters to the row data. The parameters may be ignored depending on the status of row data.

Along with SQL_PARAM_IGNORE, the following status codes are to ignore the parameter set in the SQL statement.

- SQL_ROW_DELETED
- SQL_ROW_UPDATED
- SQL_ROW_ERROR

Along with SQL_PARAM_PROCEED, the following status codes are to process the parameter set in the S QL statement.

- SQL_ROW_SUCCESS
- SQL_ROW_SUCCESS_WITH_INFO
- SQL_ROW_ADDED

Rebinding Parameter

The application can include many parameters, but if there is a buffer area for calling SQLExecDirect or SQ LExecute which uses only some parameters, rebinding the parameter is particularly useful. The remaining space of buffer area can be used to set the next parameter by modifying the existing binding through off

set.

The SQL_DESC_BIND_OFFSET_PTR header field of APD points to the binding offset. If the field is not NUL L, the driver dereferences to the pointer. If values of SQL_DESC_DATA_PTR, SQL_INDICATOR_PTR do not exist, and SQL_DESC_OCTET_LENGTH_PTR field is the NULL pointer, then the dereferenced value is adde d to the fields in the descriptor records during the execution time.

Offset is valid after rebinding. The application can directly modify the offset without calling SQLSetDescFi eld or SQLSetDescRec to update the descriptor field because SQL_DESC_BIND_OFFSET_PTR field is the po inter to offset rather than the offset itself. The pointer is NULL by default.

SQL_DESC_BIND_OFFSET_PTR field of ARD can be set by calling SQLSetDescField or through SQL_ATTR_ PARAM_BIND_OFFSET_PTR in Attribute argument of SQLSetStmtAttr. The offset binding always adds the value directly to SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR and SQL_DESC_OCTET_LENGTH_PT R. If offset is changed to another value, the new value is continuously added directly to each descriptor fi eld. The new offset is not added to the previous one.

Descriptor

The way which the parameter is bound is determined by APD and the IPD fields. The arguments in SQLBi ndParameter are used to set the descriptor fields. It is more effective to use the SQLBindParameter becau se the application can call SQLBindParameter without obtaining the descriptor handle, but the fields can also be set by the SQLSetDescField function.

> ⚠ Calling SQLBindParameter for a single statement may affect other statements. It occurs when A RD related to the statement is explicitly allocated and it is related to other statements. The modi fications for the field affects all statements related the descriptor because SQLBindParameter m odifies the fields of ARD. If it is not the required behavior, the application should release the rela tionship between the descriptor and other statements before calling SQLBindParameter.

Notionally, SQLBindParameter should perform the following processes.

1. It calls SQLGetStmtAttr, and obtains the APD handle.

2. It calls SQLGetDescField, and obtains SQL_DESC_COUNT field of APD is obtained. If the value of Col umnNumber exceeds the value of SQL_DESC_COUNT, then it calls SQLSetDescField to increase the v alue of SQL_DESC_COUNT to the value of ColumnNumber.

3. It calls SQLSetDescField multiple times, and sets the values of the following fields of APD.

- It sets SQL_DESC_TYPE and SQL_DESC_CONCISE_TYPE to the value of ValueType.
  - It excludes ValueType if ValueType is one of the implied identifiers in datetime or interval sub for mat. It sets SQL_DESC_TYPE to SQL_DATETIME or SQL_INTERVAL, and it sets SQL_DESC_CONCI

SE_TYPE to the implied identifier, and sets SQL_DESC_DATETIME_INTERVAL_CODE to the corres ponding datetime or interval subcode.

- It sets SQL_DESC_OCTET_LENGTH field to the value of BufferLength.
- It sets SQL_DESC_DATA_PTR field to the value of ParameterValue.
- It sets SQL_DESC_OCTET_LENGTH_PTR field to the value of StrLen_or_Ind.
- It also sets SQL_DESC_INDICATOR_PTR field to the value of StrLen_or_Ind.
- StrLen_or_Ind specifies both the indicator information and length of parameter value.

4. It calls SQLGetStmtAttr, and obtains the IPD handle.

5. It calls SQLGetDescField, and obtains SQL_DESC_COUNT field of IPD. If the value of ColumnNumber exceeds the value of SQL_DESC_COUNT, then it calls SQLSetDescField to increase the value of SQL_ DESC_COUNT to the value of ColumnNumber.

6. It calls SQLSetDescField multiple times, and sets the values of following fields of IPD.

- It sets SQL_DESC_TYPE and SQL_DESC_CONCISE_TYPE to the value of ParameterType.
  - It excludes ParameterType, if ParameterType is one of the implied identifiers with datetime or int erval sub format. It sets SQL_DESC_TYPE to SQL_DATETIME or SQL_INTERVAL, and it sets SQL_D ESC_CONCISE_TYPE to the implied identifier, and sets SQL_DESC_DATETIME_INTERVAL_CODE t o the corresponding datetime or interval subcode.
- It sets one or more SQL_DESC_LENGTH, SQL_DESC_PRECISION and SQL_DESC_DATETIME_INTERVA L_PRECISION for ParameterType, properly.
- It sets SQL_DESC_SCALE to the value of DecimalDigits.

If it failed to call SQLBindParameter, the contents of the descriptor fields to be set in APD are not defined and SQL_DESC_COUNT field of APD is not changed. Additionally, SQL_DESC_LENGTH, SQL_DESC_PRECI SION, SQL_DESC_SCALE, and SQL_DESC_TYPE fields of the proper record in IPD are not defined and SQL _DESC_COUNT field of IPD is not changed.

SQLBrowseConnect

It is not supported.

Conformance

Introduced version: ODBC 1.0
Standards compliance: ODBC

Overview

SQLBrowseConnect finds the attribute and its value which are required to connect to the data source, an d supports a method for the iterative list.

Syntax

```
SQLRETURN SQLBrowseConnect(
    SQLHDBC         ConnectionHandle,
    SQLCHAR *       InConnectionString,
    SQLSMALLINT     StringLength1,
    SQLCHAR *       OutConnectionString,
    SQLSMALLINT     BufferLength,
    SQLSMALLINT *   StringLength2Ptr);
```

SQLBulkOperations

It is not supported.

Conformance

Introduced version: ODBC 3.0
Standards compliance: ODBC

Overview

SQLBulkOperations performs the massive bookmark operations such as the bulk inserts, and the updates, deletes, fetches through the bookmark.

Syntax

```
SQLRETURN SQLBulkOperations(
    SQLHSTMT       StatementHandle,
    SQLUSMALLINT   Operation);
```

SQLCancel

Conformance

Introduced version: ODBC 1.0
Standards compliance: ISO 92

Overview

SQLCancel cancels the statement in progress.

Syntax

```
SQLRETURN SQLCancel(
    SQLHSTMT     StatementHandle);
```

Arguments

StatementHandle

[Input] It is the statement handle.

Returns

SQL_SUCCESS, SQL_ERROR or SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY010	Function sequence error	The asynchronously executing function is called for the connection handle related to StatementHandle,. This asynchronous function is still being executed even when SQLCancel is called. It failed because the asynchronous operation was being executed in the connection handle related to StatementHandle.
HYT00	Connection timeout expired	The login time limit is expired before connecting to the data source. The time limit can be set via SQL_ATTR_LOGIN_TIMEOUT of SQLSetConnectAttr.

Description

SQLCancel can cancel the following processing in the statement.

- The asynchronously executing function for statement
- The function for the statement requiring the data
- The function whose statement is operated in another thread.

When calling SQLCancel to cancel the asynchronously executing function for statement or the function fo r the statement requiring the data, the diagnosis record is deleted and SQLCancel posts self diagnosis rec ord. However, when calling SQLCancel to cancel the statement operated in another thread, it neither del etes the diagnosis record nor posts self diagnosis record.

When cancelling the statement operating in another thread, it is allowed to make temporary new connec tion to cancel the statement.

SQLCancelHandle

Conformance

Introduced version: ODBC 3.8
Standards compliance: It is not available.

Overview

SQLCancelHandle cancels the statement processing.
For more information, refer to **SQLCancel**.

Syntax

```
SQLRETURN SQLCancelHandle(
    SQLSMALLINT  HandleType,
    SQLHANDLE    Handle);
```

Arguments

HandleType

[Input] It is the handle type. It supports SQL_HANDLE_STMT only.

Handle

[Input] It should be the statement handle.

Returns

SQL_SUCCESS, SQL_ERROR or SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY010	Function sequenc	The asynchronously executing statement-related function is called for the statement handles associated with the handle. This asynchronous function is still being executed even when SQLCancelHandle is called.

SQLSTATE	Error	Description
	e error	It failed because the asynchronous operation was being executed in the related connection handle when the handle is a statement.
HYT00	Connection timeout expired	The login time limit is expired before connecting to the data source. The time limit can be set via SQL_ATTR_LOGIN_TIMEOUT of SQLSetConnectAttr.
IM001	Driver does not support function	It does not support HandleType except for SQL_HANDLE_STMT.

SQLCloseCursor

Conformance

Introduced version: ODBC 3.0
Standards compliance: ISO 92

Overview

SQLCloseCursor closes an open cursor on the statement and discards the remaining results.

Syntax

```
SQLRETURN SQLCloseCursor(
    SQLHSTMT     StatementHandle);
```

Arguments

StatementHandle

[Input] It is the statement handle.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
24000	Invalid cursor state	Open cursor does not exist on the statement.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY010	Function sequence error	SQL_NEED_DATA is returned after calling SQLExecute, SQLExecDirect, the function is called before sending all data-at-execution parameters.

Description

If open cursor does not exist, SQLCloseCursor returns SQLSTATE 24000(Invalid cursor state). Calling SQLCloseCursor is as same as calling SQLFreeStmt with SQL_CLOSE option. However, when open cursor does not exist, SQLCloseCursor returns SQLSTATE 24000 (Invalid cursor state), but calling SQLFreeStmt does not affect the application.

SQLColAttribute

Conformance

Introduced version: ODBC 3.0
Standards compliance: ISO 92

Overview

SQLColAttribute returns the descriptor information for the result set column. The descriptor information i s returned as a string or integer value.

Syntax

```
SQLRETURN SQLColAttribute (
    SQLHSTMT        StatementHandle,
    SQLUSMALLINT    ColumnNumber,
    SQLUSMALLINT    FieldIdentifier,
    SQLPOINTER      CharacterAttributePtr,
    SQLSMALLINT     BufferLength,
    SQLSMALLINT *   StringLengthPtr,
    SQLLEN *        NumericAttributePtr);
```

Arguments

StatementHandle

[Input] It is the statement handle.

ColumnNumber

[Input] It is the record number to retrieve for a field value in IRD. It starts from 1 and corresponds t o the column number of result data which sequentially increases. The column can be described in r andom order.

The number 0 column can be specified to ColumnNumber, but the undefined value is returned exc ept for SQL_DESC_TYPE and SQL_DESC_OCTET_LENGTH.

FieldIdentifier

[Input] It is the descriptor handle. It defines the field retrieved in IRD. (e.g. SQL_COLUMN_TABLE_ NAME)

CharacterAttributePtr

[Output] It is the buffer pointer returning the field value when the value of FieldIdentifier field of th e ColumnNumber column of IRD is a string. If the field value is not a string, it is not used.

If CharacterAttributePtr is NULL, StringLengthPtr returns the total number of bytes which can be re turned. (Except for null-termination character)

BufferLength

[Input] It is the length of *CharacterAttributePtr when FieldIdentifier is defined in ODBC and Chara cterAttributePtr points to a string or binary buffer. If FieldIdentifier is defined in ODBC and *Charac terAttributePtr is an integer, it is ignored.

StringLengthPtr

[Output] It is the pointer returning the total number of bytes which can be returned in *Character AttributePtr. (Except for null-termination character for the character data)

For the character data, if the number of bytes which can be returned is equal to or bigger than Buf ferLength, the description information of *CharacterAttributePtr is truncated to the length of Buff erLength minus 1, and it is null terminated by the driver.

For other data types, the value of BufferLength is ignored.

NumericAttributePtr

[Output] It is the buffer pointer which returns the field value when the value of FieldIdentifier field of the ColumnNumber column of IRD is a number such as SQL_DESC_COLUMN_LENGTH. If the fie ld value is not a number, it is not used.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01004	String data, right truncated	*CharacterAttributePtr buffer is not large enough to return the e ntire string, so the string is truncated. The length of string not tru ncated is returned to *StringLengthPtr. (The function returns SQL _SUCCESS_WITH_INFO.)
07005	Prepared statement not a cursor-specification	The statement does not return the result set, and FieldIdentifier is not SQL_DESC_COUNT. The column to explain does not exist.
07009	Invalid descriptor index	The value of ColumnNumber argument is bigger than the number of columns in the result set.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.

SQLSTATE	Error	Description
HY010	Function sequence error	This function is called before SQLPrepre, SQLExecDirect, the catalog function. After calling SQLExecute, SQLExecDirect, then SQL_NEED_DATA is returned, and this function is called before sending all data-at-execution variables.
HY090	Invalid string or buffer length	*CharacterAttributePtr is a string, BufferLength is smaller than 0 but it is not SQL_NTS.
HY091	Invalid descriptor field identifier	The value of FieldIdentifier argument is not defined.

Description

SQLColAttribute returns information to *NumericAttributePtr or *CharacterAttributePtr. Integer information is returned as SQLLEN value to *NumericAttributePtr. All other data type information is returned to *CharacterAttributePtr. When information is returned to *NumericAttributePtr, the driver ignores CharacterAttributePtr, BufferLength, StringLengthPtr. When the information is returned to *CharacterAttributePtr, the driver ignores NumericAttributePtr.

SQLColAttribute returns the value in the descriptor field of IRD. The value of FieldIdentifier returned to SQLColAttribute can be obtained by calling SQLGetDescField with appropriate IRD handle.

The following table is the descriptor types returned to SQLColAttribute. The type of NumericAttributePtr is SQLLEN*.

FieldIdentifier	Information return	Description
SQL_DESC_AUTO_UNIQUE_VALUE (ODBC 1.0)	NumericAttributePtr	• SQL_TRUE: It is an auto increment column. • SQL_FALSE: It is neither an auto increment column nor a numeric type.
SQL_DESC_BASE_COLUMN_NAME (ODBC 3.0)	CharacterAttributePtr	It is the default column name for the result set columns. If the default column name does not exist(in case of an expression column), the variable will contain an empty string. This information is returned in the record field of SQL_DESC_BASE_COLUMN_NAME which is the read-only field of IRD.
SQL_DESC_BASE_TABLE_NAME (ODBC 3.0)	CharacterAttributePtr	It is the base table name which contains the column. If the base table name can not be defined or is not applicable the variable contains an empty string. This information is returned in the record field of SQL_DESC_BASE_TABLE_NAME which is the read-only field of IRD.
SQL_DESC_CASE_SENSITIVE (ODBC 1.0)	NumericAttributePtr	• SQL_TRUE: The column is case-sensitive for sorting or comparisons. • SQL_FALSE: The column is neither case-sensitive for sorting or comparison nor is a character.

FieldIdentifier	Information return	Description
SQL_DESC_CATALOG_NAME (ODBC 2.0)	CharacterAttributePtr	It is the catalog of the table containing the column.
SQL_DESC_CONCISE_TYPE (ODBC 1.0)	NumericAttributePtr	It is concise data type. For datetime and interval data, the concise data such as SQL_TYPE_TIME, SQL_INTERVAL_YEAR is returned. This information is returned in the record field of SQL_DESC_CONCISE_TYPE of IRD.
SQL_DESC_COUNT (ODBC 1.0)	NumericAttributePtr	It is the number of columns which can be used in the result set. If column does not exist in the result set, 0 is returned. ColumnNumber argument is ignored. This information is returned in the header field of SQL_DESC_COUNT of IRD.
SQL_DESC_DISPLAY_SIZE (ODBC 1.0)	NumericAttributePtr	It is the maximum number of characters required to display the column.
SQL_DESC_FIXED_PREC_SCALE (ODBC 1.0)	NumericAttributePtr	• SQL_TRUE: The column has the fixed precision and non-zero scale. • SQL_FALSE: The column does not have the fixed precision but it has the non-zero scale.
SQL_DESC_LABEL (ODBC 2.0)	CharacterAttributePtr	It is the column label or title. For example, the column name, EmpName, can be displayed as employee name or alias. If the label doe not exist, the column name is returned. If a label or name is not specified to the column, an empty string is returned.
SQL_DESC_LENGTH (ODBC 3.0)	NumericAttributePtr	It is the maximum or actual length of data of the string or binary data type. A fixed length data type is the maximum character length, and a variable-length data type is the actual character length. This value always excludes the null-termination byte of the string at the end. This information is returned in the record field of SQL_DESC_LENGTH of IRD.
SQL_DESC_LITERAL_PREFIX (ODBC 3.0)	CharacterAttributePtr	This VARCHAR (128) record field contains a character or string which the driver recognizes the prefix of the data type. The data type to which a prefix is not applied contains an empty string.
SQL_DESC_LITERAL_SUFFIX (ODBC 3.0)	CharacterAttributePtr	This VARCHAR (128) record field contains a character or string which the driver recognizes the suffix of the data type. The data type to which a suffix is not applied contains an empty string.
SQL_DESC_LOCAL_TYPE_NAME (ODBC 3.0)	CharacterAttributePtr	The VARCHAR (128) record field contains the localized(native language) name of data type which is different from the regular name of data type. If the localized name does not exist, an empty string is returned. This field is only for display purposes. The character set of string depends on the locale, and the default is usually the character set of the server.
		It is the column alias of when the column alias is applied. If the column alias

FieldIdentifier	Information return	Description
SQL_DESC_NAME (ODBC 3.0)	CharacterAttributePtr	is not applied, the column name is returned. In both cases, SQL_DESC_UNNAMED is set to SQL_NAMED. If column name or alias does not exist, an empty string is returned, and SQL_DESC_UNNAMED is set to SQL_UNNAMED. This information is returned in the record field of SQL_DESC_NAME of IRD.
SQL_DESC_NULLABLE (ODBC 3.0)	NumericAttributePtr	• SQL_NULLABLE: The column can have NULL value. • SQL_NO_NULLS: The column can not have NULL value. • SQL_NULLABLE_UNKNOWN: It is unknown whether NULL value is allowed for the column. This information is returned in the record field of SQL_DESC_NULLABLE of IRD.
SQL_DESC_NUM_PREC_RADIX (ODBC 3.0)	NumericAttributePtr	If the data type of SQL_DESC_TYPE field is the approximate numeric data type, this field contains 2 because SQL_DESC_PRECISION field contains the number of bits. If the data type of SQL_DESC_TYPE field is the exact numeric data type, this field contains 10 because SQL_DESC_PRECISION field contains the number of decimal digits. This field is set to 0 for all non-numeric data types.
SQL_DESC_OCTET_LENGTH (ODBC 3.0)	NumericAttributePtr	It is byte length of string or binary data type. For a fixed-length character or binary data type, it is the actual byte length. For a variable-length character or binary data type, it is the maximum length in bytes. This value does not include null termination. This information is returned in the record field of SQL_DESC_OCTET_LENGTH of IRD.
SQL_DESC_PRECISION (ODBC 3.0)	NumericAttributePtr	It is the precision applicable to the numeric data type. For SQL_TYPE_TIME, SQL_TYPE_TIMESTAMP, all interval data types representing time interval, the value is the applicable fractional seconds precision. This information is returned in the record field of SQL_DESC__PRECISION of IRD.
SQL_DESC_SCALE (ODBC 3.0)	NumericAttributePtr	It is the scale applicable to the numeric data type. For DECIMAL or NUMERIC data type, the scale is defined, and for all other data types, it is not defined. This information is returned in the record field of SQL_DESC_SCALE of IRD.
SQL_DESC_SCHEMA_NAME (ODBC 2.0)	CharacterAttributePtr	It is the schema of a table containing a column.
SQL_DESC_SEAR		• SQL_PRED_NONE: The column can not used in WHERE clause. (It is as same as SQL_UNSEARCHABLE of ODBC 2.x.) • SQL_PRED_CHAR: The column can be used in WHERE clause but only

FieldIdentifier	Information return	Description
CHABLE (ODBC 1.0)	NumericAttributePtr	with the LIKE predicate. (It is as same as SQL_LIKE_ONLY of ODBC 2.x.). • SQL_PRED_BASIC: Any comparison operator except LIKE can be used in WHERE clause. (It is as same as SQL_EXCEPT_LIKE of ODBC 2.x.) • SQL_PRED_SEARCHABLE: The column can be used in WHERE clause together with any comparison operator.
SQL_DESC_TABLE_NAME (ODBC 2.0)	CharacterAttributePtr	It is the name of table including the column. If the tabe name is unknown, an empty string is returned.
SQL_DESC_TYPE (ODBC 3.0)	NumericAttributePtr	It is the numeric value to specify the SQL data type. For datetime or interval data type, the verbose data type such as SQL_DATETIME or SQL_INTERVAL is returned. This information is returned in the record field of SQL_DESC_TYPE of IRD.
SQL_DESC_TYPE_NAME (ODBC 1.0)	CharacterAttributePtr	It is the data type name which is dependent on the data source. (e.g. "CHARACTER", "CHARACTER VARYING", "CHARACTER LONG VARYING")
SQL_DESC_UNNAMED (ODBC 3.0)	NumericAttributePtr	It is SQL_NAMED or SQL_UNNAMED. If the column alias or column name is included in the field of SQL_DESC_NAME of IRD, SQL_NAMED is returned, and if there is not a column name or alias, SQL_UNNAME is returned. This information is returned in the record field of SQL_DESC_UNNAMED of IRD.
SQL_DESC_UNSIGNED (ODBC 1.0)	NumericAttributePtr	• SQL_TRUE: The column is neither an unsigned nor is a number. • SQL_FALSE: The column is a signed.
SQL_DESC_UPDATABLE (ODBC 1.0)	NumericAttributePtr	The column can have the value of SQL_ATTR_READONLY, SQL_ATTR_WRITE, SQL_ATTR_READWRITE_UNKNOWN.

SQLColAttributes

Conformance

Introduced version: ODBC 1.0
Standards compliance: It is not available.

Overview

SQLColAttributes function in ODBC 2.0 was replaced with SQLColAttribute function in ODBC 3.x. For more information, refer to **SQLColAttribute**.

SQLColumnPrivileges

Conformance

Introduced version: ODBC 1.0
Standards compliance: ODBC

Overview

SQLColumnPrivileges returns a list of columns and related privileges for the specified table as a result set.

Syntax

```
SQLRETURN SQLColumnPrivileges(
    SQLHSTMT      StatementHandle,
    SQLCHAR *     CatalogName,
    SQLSMALLINT   NameLength1,
    SQLCHAR *     SchemaName,
    SQLSMALLINT   NameLength2,
    SQLCHAR *     TableName,
    SQLSMALLINT   NameLength3,
    SQLCHAR *     ColumnName,
    SQLSMALLINT   NameLength4);
```

Arguments

StatementHandle

[Input] It is the statement handle.

CatalogName

[Input] It is the catalog name. CatalogName can not include the string search pattern.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, CatalogName is treated a s a case-insensitive identifier. If it is set to SQL_FALSE, CatalogName is case-sensitive, and it is an or dinary argument literally processed.

NameLength1

[Input] It is the length of *CatalogName.

SchemaName

[Input] It is the schema name. SchemaName can not include the string search pattern.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, SchemaName is treated a s a case-insensitive identifier. If it is set to SQL_FALSE, SchemaName is case-sensitive, and it is an or dinary argument literally processed.

NameLength2

[Input] It is the length of *SchemaName.

TableName

[Input] It is the table name. This argument can not be a null pointer. TableName can not include th e string search pattern.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, TableName is treated as a case-insensitive identifier. If it is set to SQL_FALSE, TableName is case-sensitive, and it is an ordinar y argument literally processed.

NameLength3

[Input] It is the length of *TableName.

ColumnName

[Input] It is the string search pattern for the column name.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, ColumnName is treated a s a case-insensitive identifier. If it is set to SQL_FALSE, ColumnName is case-sensitive, and it is a pat ternvalue literally processed.

NameLength4

[Input] It is the length of *ColumnName.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
08S01	Communication link failure	Before the function processing is completed, the connection betwee n the driver and the data source is failed.
24000	Invalid cursor state	SQLFetch, SQLFetchScroll are called and a cursor is open.
HY000	General error	It is an error without specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY009	Invalid use of null pointer	TableName argument is the null pointer. The attribute value of SQL_ATTR_METADATA_ID is SQL_TRUE state

SQLSTATE	Error	Description
		ment, and SchemaName or ColumnName is the null pointer.
HY010	Function sequence error	After calling SQLExecute, SQLExecDirect, then SQL_NEED_DATA is returned, and the function is called before sending all data-at-execution variables.
HY090	Invalid string or buffer length	A name length argument value is smaller than 0 but it is not SQL_NTS.
HYT00	Timeout expired	Before downloading the entire result set from the data source, the query timeout expired. The timeout can be set through SQL_ATTR_QUERY_TIMEOUT of SQLSetStmtAttr.

Description

SQLColumnPrivileges returns the standard result set aligned as TABLE_CAT, TABLE_SCHEM, TABLE_NAME, COLUMN_NAME, PRIVILEGE.

The following table describes the columns in the result set.

Column name	Column number	Data type	Description
TABLE_CAT (ODBC 1.0)	1	VARCHAR	It is the catalog identifier.
TABLE_SCHEM (ODBC 1.0)	2	VARCHAR	It is the schema identifier.
TABLE_NAME (ODBC 1.0)	3	VARCHAR not NULL	It is the table identifier.
COLUMN_NAME (ODBC 1.0)	4	VARCHAR not NULL	It is the column name. It returns an empty string for a column which does not have a name.
GRANTOR (ODBC 1.0)	5	VARCHAR	It is the grantor name.
GRANTEE (ODBC 1.0)	6	VARCHAR not NULL	It is the grantee name.
PRIVILEGE (ODBC 1.0)	7	VARCHAR not NULL	It is the column privilege identifier. It can be one of the followings. • SELECT: The grantee is allowed to retrieve the column data. • INSERT: The grantee is allowed to insert the data to the column of the associated table. • UPDATE: The grantee is allowed to update the column data. • REFERENCES: The grantee is allowed to reference the column in the constrains. (e.g. unique, referential, table check constraint)

Column name	Column number	Data type	Description
IS_GRANTABLE (ODBC 1.0)	8	VARCHAR	It checks whether the grantee can grant the privilege to other user, and it is specified as "YES", "NO".

SQLColumns

Conformance

Introduced version: ODBC 1.0
Standards compliance: Open Group

Overview

SQLColumns returns a list of column names in the specified table as a result set.

Syntax

```
SQLRETURN SQLColumns(
    SQLHSTMT       StatementHandle,
    SQLCHAR *      CatalogName,
    SQLSMALLINT    NameLength1,
    SQLCHAR *      SchemaName,
    SQLSMALLINT    NameLength2,
    SQLCHAR *      TableName,
    SQLSMALLINT    NameLength3,
    SQLCHAR *      ColumnName,
    SQLSMALLINT    NameLength4);
```

Arguments

StatementHandle

[Input] It is the statement handle.

CatalogName

[Input] It is the catalog name. CatalogName can not include the string search pattern.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, CatalogName is treated a s a case-insensitive identifier. If it is set to SQL_FALSE, CatalogName is case-sensitive, and it is an or dinary argument literally processed.

NameLength1

[Input] It is the length of *CatalogName

SchemaName

[Input] It is the string search pattern for the schema name.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, SchemaName is treated a s a case-insensitive identifier. If it is set to SQL_FALSE, SchemaName is case-sensitive, and it is an or dinary argument literally processed.

NameLength2

[Input] It is the length of *SchemaName.

TableName

[Input] It is the string search pattern for the table name.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, TableName is treated as a case-insensitive identifier. If it is set to SQL_FALSE, TableName is case-sensitive, and it is an ordinar y argument literally processed.

NameLength3

[Input] It is the length of *TableName.

ColumnName

[Input] It is the string search pattern for the column name.

If the SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, ColumnName is treated a s a case-insensitive identifier. If it is set to SQL_FALSE, ColumnName is case-sensitive, and it is a pat tern value literally processed.

NameLength4

[Input] It is the length of *ColumnName.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
08S01	Communication l ink failure	Before the function processing is completed, the connection between the dr iver and the data source is failed.
24000	Invalid cursor stat e	SQLFetch, SQLFetchScroll are called and a cursor is open.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocatio n error	It is a memory allocation error.
HY009	Invalid use of null pointer	TableName argument is a null pointer. The attribute value of SQL_ATTR_METADATA_ID statement is SQL_TRUE, S chemaName or ColumnName is a null pointer.
HY010	Function sequenc e error	After calling SQLExecute, SQLExecDirect, then SQL_NEED_DATA is returned, and the function is called before sending all data-at-execution variables.

SQLSTATE	Error	Description
HY090	Invalid string or buffer length	A name length argument value is smaller than 0 but it is not SQL_NTS.
HYT00	Timeout expired	Before downloading the entire result set from the data source, the query timeout expired. The timeout can be set through SQL_ATTR_QUERY_TIMEOUT of SQLSetStmtAttr.

Description

This function is generally used prior to the execution of statement which retrieves information about the columns of the table or tables in the catalog of the data source. SQLColumns can be used to retrieve all data types returned by SQLTables. By contrast, SQLColAttribute and SQLDescribeCol describe the columns in the result set, and SQLNumResultCols returns the number of columns in the result set.

SQLColumns returns the standard result set sorted as TABLE_CAT, TABLE_SCHEM, TABLE_NAME, ORDINAL_POSITION.

The following table describes the columns in the result set.

Column name	Column number	Data type	Description
TABLE_CAT (ODBC 1.0)	1	VARCHAR	It is the catalog name.
TABLE_SCHEM (ODBC 1.0)	2	VARCHAR	It is the schema name.
TABLE_NAME (ODBC 1.0)	3	VARCHAR not NULL	It is the table name.
COLUMN_NAME (ODBC 1.0)	4	VARCHAR not NULL	It is the column name. It returns an empty string for a column which does not have a name.
DATA_TYPE (ODBC 1.0)	5	SMALLINT not NULL	It is SQL data type. For datetime and interval data types, the column returns the concise data type like SQL_TYPE_DATE, SQL_INTERVAL_YEAR_TO_MONTH.
TYPE_NAME (ODBC 1.0)	6	VARCHAR not NULL	It is the data type name dependent on the data source. (e.g. "CHARACTER", "CHARACTER VARYING", "CHARACTER LONG VARYING")
COLUMN_SIZE (ODBC 1.	7	INTEGER	If DATA_TYPE is SQL_CHAR or SQL_VARCHAR, the column includes the maximum number of characters up to length of the column. If it is datetime data column, it is the number of characters needed for converting the value to the character. If it is a numeric data type, it is the total

Column name	Column number	Data type	Description
0)			number of digits of the column NUM_PREC_RADIX or the number of b its allowed for the column. If it is an interval data type, it is the number of characters needed to be represented by the interval leading precisio n.
BUFFER_LEN GTH (ODBC 1.0)	8	INTEGER	If SQL_C_DEFAULT is specified, it is the byte length of the data to be tr ansmitted to SQLGetData, SQLFetch, SQLFetchScroll.
DECIMAL_D IGITS (ODBC 1.0)	9	SMALLINT	For a positive number, it is the number of significant digits on the right of the decimal point. For a negative number, it is the number of signific ant digits on the left of the decimal point. For SQL_TYPE_TIME and SQ L_TYPE_TIMESTAMP, this column is the number of digits of fractional s econds. For interval data type that contains the second, it is the numbe r of digits right of the decimal point(fractional seconds). The data type that DECIMAL DIGITS can not be applied, returns NULL.
NUM_PREC _RADIX (OD BC 1.0)	10	SMALLINT	For the numeric data type, it is 2 or 10. For 2, COLUMN_SIZE and DECIMAL_DIGITS are the number of bits allo wed in the column. For 10, COLUMN_SIZE and DECIMAL_DIGITS are th e number of digits allowed in the column. The data type that NUM_PREC_RADIX can not be applied, returns NUL L.
NULLABLE (ODBC 1.0)	11	SMALLINT n ot NULL	• SQL_NO_NULLS: The column can not have NULL value. • SQL_NULLABLE: The column can have NULL value. • SQL_NULLABLE_UNKNOWN: It is unknown whether the column is allowed to have NULL value.
REMARKS (ODBC 1.0)	12	VARCHAR	It is the description on the column.
COLUMN_D EF (ODBC 3. 0)	13	VARCHAR	It is the default value of column. If the value is enclosed by the quote, t he column should be interpreted as a string.
SQL_DATA_ TYPE (ODBC 3.0)	14	SMALLINT n ot NULL	It is the SQL data type of the record field of SQL_DESC_TYPE of IRD. Th e column is the same as DATA TYPE except the datetime and interval d ata types. For datetime and interval data types, the column returns the nonconcise data type like SQL_DATE, SQL_INTERVAL, and the specific data type is determined using SQL_DATETIME_SUB column.
SQL_DATETI ME_SUB (O DBC 3.0)	15	SMALLINT	It is the sub type code of datetime and interval data types. Other data t ypes return NULL.
CHAR_OCTE T_LENGTH (ODBC 3.0)	16	INTEGER	It is the maximum length in bytes of characters or binary data type colu mn. Other data types return NULL.
ORDINAL_P			

Column name	Column number	Data type	Description
OSITION (ODBC 3.0)	17	INTEGER not NULL	It is the column position in the table.
IS_NULLABLE (ODBC 3.0)	18	VARCHAR	• "NO": The column can not contain NULL. • "YES": The column can contain NULL. If it is unknown whether NULL is allowed, It returns a zero-length string.

SQLConnect

Conformance

Introduced version: ODBC 1.0
Standards compliance: ISO 92

Overview

SQLConnect sets a connection between the driver and the data source. The connection handle refers to a ll information about the connection including status, transaction status and error information.

Syntax

```
SQLRETURN SQLConnect(
    SQLHDBC        ConnectionHandle,
    SQLCHAR *      ServerName,
    SQLSMALLINT    NameLength1,
    SQLCHAR *      UserName,
    SQLSMALLINT    NameLength2,
    SQLCHAR *      Authentication,
    SQLSMALLINT    NameLength3);
```

Arguments

ConnectionHandle

[Input] It is the connection handle.

ServerName

[Input] It is the data source name.

NameLength1

[Input] It is the length of *ServerName.

UserName

[Input] It is the user identifier.

NameLength2

[Input] It is the length of *UserName.

Authentication

[Input] It is the authentication string (typically password).

NameLength3

[Input] It is the length of *Authentication.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE, SQL_STILL_EXECUTING

Diagnosis

SQLSTATE	Error	Description
08001	Client unable to establish connection	The driver can not set a connection with a data source.
08002	Connection name in use	The specified ConnectionHandle is already connected with the data source.
08004	Server rejected the connection	In the state of the setting limits, the data source rejects to establish the connection.
08S01	Communication link failure	Before the function processing is completed, the connection between the driver and the data source is failed.
28000	Invalid authorization specification	The argument value of UserName or Authentication is not correct.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY090	Invalid string or buffer length	The value of NameLength1, NameLength2 or NameLength3 is smaller than 0, but it is not SQL_NTS.
HYT00	Timeout expired	Before connecting to the data source, Login timeout expired. The timeout can be set through SQL_ATTR_LOGIN_TIMEOUT of SQLSetConnectAttr.

Description

The driver searches for the user DSN information in an order of $HOME/.odbc.ini file and /home/.odbc.ini file which are files set in $ODBCINI environment variable. If the DSN which was input in the user DSN does not exist, the driver searches for the DSN information in an order of $ODBCSYSINI/odbc.ini file, /etc/odbc.ini file which is system DSN.

SQLCopyDesc

It is not supported.

Conformance

Introduced version: ODBC 3.0
Standards compliance: ISO 92

Overview

SQLCopyDesc copies the descriptor information from one descriptor handle to another descriptor handle.

Syntax

```
SQLRETURN SQLCopyDesc(
    SQLHDESC     SourceDescHandle,
    SQLHDESC     TargetDescHandle);
```

SQLDescribeCol

Conformance

Introduced version: ODBC 1.0
Standards compliance: ISO 92

Overview

SQLDescribeCol returns a column name, type, column size, decimal place, information about null permit i n the result set column. The information can be used in the fields of IRD.

Syntax

```
SQLRETURN SQLDescribeCol(
    SQLHSTMT       StatementHandle,
    SQLUSMALLINT   ColumnNumber,
    SQLCHAR *      ColumnName,
    SQLSMALLINT    BufferLength,
    SQLSMALLINT *  NameLengthPtr,
    SQLSMALLINT *  DataTypePtr,
    SQLULEN *      ColumnSizePtr,
    SQLSMALLINT *  DecimalDigitsPtr,
    SQLSMALLINT *  NullablePtr);
```

Arguments

StatementHandle

[Input] It is the statement handle.

ColumnNumber

[Input] It is the column number in the result set which is started from 1 and increased sequentially.

ColumnName

[Output] It is the buffer pointer which is terminated by a null and returns the column name. This va lue can be read in SQL_DESC_NAME field of IRD. If the column name does not exist or the column name is unknown, the driver returns an empty string.

If ColumnName is null, NameLengthPtr returns the total number of bytes returnable. (excluding nu ll-termination character)

BufferLength

[Input] It is the length of *ColumnName.

NameLengthPtr

[Output] It is the buffer pointer which returns the total number of bytes returnable to *ColumnName (excluding the null-termination character). If the returnable length is equal to or bigger than BufferLength, *ColumnName is truncated to the length of BufferLength minus null.

DataTypePtr

[Output] It is the buffer pointer which returns the SQL type of a column. The value can be read in SQL_DESC_CONCISE_TYPE of IRD.

ColumnSizePtr

[Output] It is the buffer pointer which returns the column size of the data source.

DecimalDigitsPtr

[Output] It is the buffer pointer which returns the decimal place of the data source.

NullablePtr

[Output] It is the buffer pointer which returns whether the column allows for null. The value can be read in SQL_DESC_NULLABLE field of IRD. It is one of the followings.

- SQL_NO_NULLS: The column does not allow NULL.
- SQL_NULLABLE: The column allows NULL.
- SQL_NULLABLE_UNKNOWN: The driver can not determine whether the column allows NULL.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01004	String data, right truncated	*ColumnName is not large enough to return the entire column name, so the column name is truncated. The column length not truncated is returned in *NameLengthPtr. (The function returns SQL_SUCCESS_WITH_INFO.)
07005	Prepared statement not a cursor-specification	The statement does not return a result set, so there is not a column to be explained.
07009	Invalid descriptor index	The value of ColumnNumber argument is bigger than the number of columns in the result set.
08S01	Communication link failure	Before the function processing is completed, the connection between the driver and the data source is failed.

SQLSTATE	Error	Description
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation failure	It is a memory allocation error.
HY010	Function sequence error	After calling SQLExecute, SQLExecDirect, then SQL_NEED_DATA is returned, the function is called before sending all data-at-execution variables.
HY090	Invalid string or buffer length	The value of BufferLength argument is smaller than 0.

Description

Generally, the application calls SQLDescibeCol before or after calling SQLExecute related after calling SQL Prepare. Also, the application may call SQLDescribeCol after calling SQLExecDirect.

SQLDescribeParam

Conformance

Introduced version: ODBC 1.0
Standards compliance: ODBC

Overview

SQLDescribeParam returns the description of a parameter marker related to a prepared SQL statement. T his information can be used in the fields of the IPD.

Syntax

```
SQLRETURN SQLDescribeParam(
    SQLHSTMT        StatementHandle,
    SQLUSMALLINT    ParameterNumber,
    SQLSMALLINT *   DataTypePtr,
    SQLULEN *       ParameterSizePtr,
    SQLSMALLINT *   DecimalDigitsPtr,
    SQLSMALLINT *   NullablePtr);
```

Arguments

StatementHandle

[Input] It is the statement handle.

ParameterNumber

[Input] It is the parameter marker number which is started from 1 and increased sequentially.

DataTypePtr

[Output] It is the buffer pointer which returns the SQL type of parameter. The value can be read in the record field of SQL_DESC_CONCISE_TYPE of IPD.

ParameterSizePtr

[Output] It is the buffer pointer which returns the size of column or expression for the parameter marker.

DecimalDigitsPtr

[Output] It is the buffer pointer which returns the decimal places of column or expression for the p arameter marker.

NullablePtr

[Output] It is the buffer pointer which returns whether the parameter allows for null. The value can be read in SQL_DESC_NULLABLE of IPD. The value is one of the followings.

- SQL_NO_NULLS: The parameter does not allow NULL.
- SQL_NULLABLE: The parameter allows NULL.
- SQL_NULLABLE_UNKNOWN: The driver can not determine whether the parameter allows NULL.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
07009	Invalid descriptor index	The value of ParameterNumber argument is smaller than 1. The value of ParameterNumber argument is bigger than the number of parameters of the related SQL statement.
08S01	Communication link failure	Before the function processing is completed, the connection between the driver and the data source is failed.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY010	Function sequence error	The function is called before SQLPrepare or SQLExecDirect. After calling SQLExecute, SQLExecDirect, then SQL_NEED_DATA is returned, and the function is called before sending all data-at-execution variables.

Description

SQL_VARCHAR should be returned to *DataTypePtr, 4000 should be returned to *ParameterSizePtr, 0 should be returned to *DecimalDigitsPtr, and SQL_NULLABLE should be returned to *NullablePtr because the driver can not provide the exact information of the parameter using the prepared SQL.

SQLDisconnect

Conformance

Introduced version: ODBC 1.0
Standards compliance: ISO 92

Overview

SQLDisconnect closes the connection related to the specific connection handle.

Syntax

```
SQLRETURN SQLDisconnect(
    SQLHDBC     ConnectionHandle);
```

Arguments

ConnectionHandle

[Input] It is the connection handle.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE, SQL_STILL_EXECUTING

Diagnosis

SQLSTATE	Error	Description
08003	Connection not open	The connection of ConnectionHandle argument is not open.
25000	Invalid transaction state	The transaction is in progress on the connection of ConnectionHandle argument. The transaction remains active.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.

Description

If the application calls SQLDisconnect to the connection handle with an incomplete transaction, the driver returns SQLSTATE 25000 (Invalid transaction state), the transaction is not changed, and the connection is open. The incomplete transaction is a transaction which is not committed or rolled back by using SQLEnd Tran.

If the application calls SQLDisconnect before disconnecting all statements, the driver is disconnected from the data source, then deletes all statements and the descriptor explicitly assigned to the connection handl e.

SQLDriverConnect

Conformance

Introduced version: ODBC 1.0
Standards compliance: ODBC

Overview

SQLDriverConnect can replace SQLConnect, and it supports the data source which requires more information of connection than three arguments of SQLConnect.

SQLDriverConnect establishes the connection by using the connection string which includes other information required by the data source name, one or more users, one or more passwords and the data sources.

When the connection is established, SQLDriverConnect returns the completed connection string. The application can use this string when requesting the next connection.

Syntax

```
SQLRETURN SQLDriverConnect(
    SQLHDBC         ConnectionHandle,
    SQLHWND         WindowHandle,
    SQLCHAR *       InConnectionString,
    SQLSMALLINT     StringLength1,
    SQLCHAR *       OutConnectionString,
    SQLSMALLINT     BufferLength,
    SQLSMALLINT *   StringLength2Ptr,
    SQLUSMALLINT    DriverCompletion);
```

Arguments

ConnectionHandle

[Input] It is the connection handle.

WindowHandle

[Input] It is the window handle. The application passes the superordinate window handle or a null pointer, and for the null pointer, SQLDriverConnect does not display the dialog box.

InConnectionString

[Input] It is the full connection string, partial connection string or empty string.

StringLength1

[Input] It is the length of *InConnectionString.

OutConnectionString

[Output] It is the buffer pointer of the completed connection string. If it is successfully connected to the target data source, the buffer contains the completed connection string. The application should allocate a buffer with at least 1,024 characters.

If OutConnectionString is NULL, the total number of returnable characters are returned in StringLength2Ptr (excluding null termination character).

BufferLength

[Input] It is the length of *OutConnectionString

StringLength2Ptr

[Output] It is the buffer pointer which returns the total number of returnable characters to *OutConnectionString (excluding null termination character). If the returnable length is equal to or bigger than BufferLength, *OutConnectionString is truncated to the length of BufferLength minus a null-termination character.

DriverCompletion

[Input] It is a flag that indicates whether the driver has to request more information. the value is one of SQL_DRIVER_PROMPT, SQL_DRIVER_COMPLETE, SQL_DRIVER_COMPLETE_REQUIRED, SQL_DRIVER_NOPROMPT.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_ERROR, SQL_INVALID_HANDLE, SQL_STILL_EXECUTING

Diagnosis

SQLSTATE	Error	Description
01004	String data, right truncated	*OutConnectionString buffer is not large enough to return the entire connection string, so the connection string is truncated. The connection string length not truncated is returned in *StringLength2Ptr. (The function returns SQL_SUCCESS_WITH_INFO.)
08001	Client unable to establish connection	The driver can not establish a connection with the data source.
08002	Connection name in use	The specified ConnectionHandle is already connected with a data source.

SQLSTATE	Error	Description
08004	Server rejected the connection	In the state of the limits of the setting value, the data source rejects to establish the connection.
08S01	Communication link failure	Before the function processing is completed, the connection between the driver and the data source is failed.
28000	Invalid authorization specification	The user identifier and authentication string of the connection string are not correct.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY090	Invalid string or buffer length	The value of StringLength1 argument is smaller than 0, but it is not SQL_NTS. The value of BufferLength argument is smaller than 0.
HY110	Invalid driver completion	The value of DriverCompletion argument is not one of SQL_DRIVER_PROMPT, SQL_DRIVER_COMPLETE, SQL_DRIVER_COMPLETE_REQUIRED, SQL_DRIVER_NOPROMPT.
HYC00	Optional feature not implemented	The driver does not support the ODBC operation which is required by the application.
HYT00	Timeout expired	Before connecting to the data source, login timeout is expired. The timeout can be set through SQL_ATTR_LOGIN_TIMEOUT of SQLSetConnectAttr.

Description

The syntax of connection string is as follows.

```
connection-string ::= empty-string[;] ¦ attribute[;] ¦ attribute; connection-string
empty-string ::=attribute ::= attribute-keyword=attribute-value ¦ DRIVER=[{]attribute-value[}]
attribute-keyword ::= DSN ¦ PROTOCOL ¦ CS_MODE ¦ HOST ¦ PORT ¦ UID ¦ PWD ¦ ALTERNATE_SERVERS ¦
FAILOVER_TYPE ¦ FAILOVER_GRANULARITY ¦ DATE_FORMAT ¦ TIME_FORMAT ¦ TIME_WITH_TIME_ZONE_FORMAT
¦ TIMESTAMP_FORMAT ¦ TIMESTAMP_WITH_TIME_ZONE_FORMAT ¦ CHAR_LENGTH_UNITS ¦ CONN_NAME
attribute-value ::= character-string
```

character-string is zero or more characters. attribute-keyword is case-insensitive, and attribute-value may be case-sensitive. The value of DSN keyword does not consist only of white spaces.

The following table describes attribute-keywords.

Keyword	Description
DSN	It is the data source name.
PROTOCOL	It is the connection type (DA, TCP).
CS_MODE	It sets whether to connect with dedicated mode or shared mode. If the setting is not used, the mode is determined depending on the configuration (DE

Keyword	Description
	FAULT_CS_MODE) of listener.
HOST	It is the host name or the IP address.
PORT	It is the connection port number.
TCP_NODELAY	It is a socket TCP_NODELAY option.
UID	It is the user ID.
PWD	It is the password for user ID. If password does not exist, it is an empty string (PWD=;).
ALTERNATE_SERVERS	It is a server list which attempts a connection when the failover occurs, and each server is separated with comma (,). If failover is not used, ALTERNATE_SERVERS is not set.
FAILOVER_TYPE	• CONNECTION: When the connection fails, it is connected to ALTERNATE_SERVERS. • SESSION: When the connection fails or the connection is disconnected during operating the statement, it is connected to ALTERNATE_SERVERS and the statement is restored. The statement is executed after the failover if the connection is disconnected when a transaction is not in progress.
FAILOVER_GRANULARITY	• 0: Failover proceeds even when an error occurs during the failover. • 1: Failover fails when an error occurs except for SQLExeceute(), SQLExecDirect() during the failover. • 2: Failover fails when an error occurs during the failover.
DATE_FORMAT	It is the DATE type format string.
TIME_FORMAT	It is the TIME type format string.
TIME_WITH_TIME_ZONE_FORMAT	It is the TIME WITH TIME ZONE type format string.
TIMESTAMP_FORMAT	It is the TIMESTAMP type format string.
TIMESTAMP_WITH_TIME_ZONE_FORMAT	It is the TIMESTAMP WITH TIME ZONE type format string.
CHAR_LENGTH_UNITS	If ParameterType is one of SQL_CHAR or SQL_VARCHAR in SQLBindParameter(), it is a unit of ColumnSize. • BYTE, OCTETS: Bytes unit • CHAR, CHARACTERS: Characters unit
CONN_NAME	It is the connection name used in XA. The specified name is valid only in embedded SQL program, and is ignored in other programs.

SQLEndTran

Conformance

Introduced version: ODBC 3.0
Standards compliance: ISO 92

Overview

SQLEndTran requests a commit or rollback for active transactions of all statements related to the connecti on.

Syntax

```
SQLRETURN SQLEndTran(
    SQLSMALLINT   HandleType,
    SQLHANDLE     Handle,
    SQLSMALLINT   CompletionType);
```

Arguments

HandleType

[Input] It is the handle identifier. If it is an environment handle it is SQL_HANDLE_ENV, and if it is a connection handle it is SQL_HANDLE_DBC.

Handle

[Input] It is the handle of HandleType which indicates the transaction range.

CompletionType

[Input] It is SQL_COMMIT or SQL_ROLLBACK.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE, SQL_STILL_EXECUTING

Diagnosis

SQLSTATE	Error	Description
08003	Connection not o	HandleType is SQL_HANDLE_DBC, and a handle is not connected.

SQLSTATE	Error	Description
	pen	
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY010	Function sequence error	After calling SQLExecute, SQLExecDirect, then SQL_NEED_DATA is returned, and the function is called before sending all data-at-execution variables.
HY012	Invalid transaction operation code	The value of CompletionType argument is not one of SQL_COMMIT or SQL_ROLLBACK.
HY092	Invalid attribute/ option identifier	The value of HandleType argument is not one of SQL_HANDLE_ENV or SQL_HANDLE_DBC.

Description

If CompletionType is SQL_COMMIT, then SQLEndTran requests the commit for all active transactions of statements related to the connection. If CompletionType is SQL_ROLLBACK, then SQLEndTran requests the rollback for all active transactions of statements related to the connection. If active transaction does not exist, SQLEndTran returns SQL_SUCCESS without affecting the data source.

If the driver is a manual commit mode(The SQL_ATTR_AUTOCOMMIT attribute is set to SQL_AUTOCOMMIT_OFF by calling SQLSetConnectAttr.) and the SQL statement is executed for the current data source, a new transaction implicitly starts.

SQLEndTran does not affect the open cursor related to the connection when committing. The cursor remains on the row pointed before calling SQLEndTran.

SQLEndTran closes all open cursors on all statements when rolling back. SQLEndTran sets the statement to the ready state, and the application does not call SQLPrepare, but it calls SQLExecute.

If an active transaction does not exist, SQLEndTran returns SQL_SUCCESS.

If the driver is an auto commit mode, SQLEndTran always returns SQL_SUCCESS regardless of CompletionType.

SQLError

Conformance

Introduced version: ODBC 1.0
Standards compliance: It is not available.

Overview

SQLError returns an error or status information.

SQLExecDirect

Conformance

Introduced version: ODBC 1.0
Standards compliance: ISO 92

Overview

If SQLExecDirect has the parameter in the statement, it executes the statement by using the current value of parameter marker. SQLExecDirect is the fastest way of when the statement is executed only once.

Syntax

```
SQLRETURN SQLExecDirect(
    SQLHSTMT     StatementHandle,
    SQLCHAR *    StatementText,
    SQLINTEGER   TextLength);
```

Arguments

StatementHandle

[Input] It is the statement handle.

StatementText

[Input] It is the executed SQL statement.

TextLength

[Input] It is the length of *StatementText.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NEED_DATA, SQL_STILL_EXECUTING, SQL_ERROR, SQL_NO_DATA, SQL_INVALID_HANDLE, SQL_PARAM_DATA_AVAILABLE

Diagnosis

SQLSTATE	Error	Description
01004	String data, right truncated	The string or binary data returned to the input/output or output parameters are truncated. The right part of the string is truncated. (The function returns SQL_SUCCESS_WITH_INFO.)
01S02	Option value changed	It is temporarily replaced with a similar value because the attribute value of t he specified statement is not suitable for executing the operation. (SQLGetS tmtAttr can be called to see which value is temporarily changed.) The replac ed value is valid until the cursor is closed, and it is changed to the previous v alue when the cursor is closed. The statement attributes which can be changed are as follows. SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ ATTR_KEYSET _SIZE, SQL_ATTR_MAX_LENGTH, SQL_ATTR_MAX_ROWS, SQL_ ATTR_QUE RY_TIMEOUT, SQL_ATTR_SIMULATE_CURSOR. (The function returns SQL_SUCCESS_WITH_INFO.)
07006	Restricted data type attribute violation	The data value identified by ValueType argument of SQLBindParameter can not be converted to the data type identified by ParameterType argument of SQLBindParameter. The data value returned to the parameter of SQL_PARAM_INPUT_OUTPUT o r SQL_PARAM_OUTPUT can not be converted to the data type identified by ValueType argument of SQLBindParameter. (If one or more rows are successfully returned, the function returns SQL_SU CCESS_WITH_INFO.)
07007	Restricted parameter value violation	The parameter type is SQL_PARAM_INPUT_OUTPUT, and *StrLen_or_IndPtr of SQLBindParameter is not one of SQL_NULL_DATA, SQL_DEFAULT_PARA M, SQL_LEN_DATA_AT_EXEC(len), or SQL_DATA_AT_EXEC.
08S01	Communication link failure	Before the function processing is completed, the connection between the dr iver and the data source is failed.
22001	String data, right truncation	The string, or binary data is truncated.
22002	Indicator variable req uired but not supplied	NULL data is bound to the output parameter whose of StrLen_or_IndPtr of S QLBindParameter is the null pointer.
24000	Invalid cursor state	The cursor is positioned in StatementHandle through SQLFetch, SQLFetchScr oll. The cursor is open but it is not positioned in StatementHandle. *StatementText is the positioned update or delete statement, and the cursor

SQLSTATE	Error	Description
		is positioned before the start or after the end of the result set.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY009	Invalid use of null pointer	*StatementText is the null pointer.
HY010	Function sequence error	After calling SQLExecute, SQLExecDirect, then SQL_NEED_DATA is returned and the function is called before sending all data-at-execution variables.
HY090	Invalid string or buffer length	The argument value of TextLength is smaller than 0, but it is not SQL_NTS. The parameter value set by SQLBindParameter is the null pointer, and the parameter length is not 0, SQL_NULL_DATA, SQL_DATA_AT_EXEC, SQL_DEFAULT_PARAM, or less than SQL_LEN_DATA_AT_EXEC_OFFSET. The parameter value set by SQLBindParameter is not the null pointer, and C data type is SQL_C_BINARY or SQL_C_CHAR, and the parameter length is smaller than 0, but it is not less than SQL_NTS, SQL_NULL_DATA, SQL_DATA_AT_EXEC, SQL_DEFAULT_PARAM, or equal to or less than SQL_LEN_DATA_AT_EXEC_OFFSET.
HYT00	Timeout expired	Before returning the result set from the data source, the query timeout is expired. The timeout can be set through SQL_ATTR_QUERY_TIMEOUT of SQLSetStmtAttr.

Description

The application sends the SQL statement to the data source by calling SQLExecDirect.

The application can include one or more parameter markers in an SQL statement. The application should include a question mark (?) on the appropriate position of SQL statement to include a parameter marker.

If the SQL statement is the SELECT statement and the application connects the cursor with SQLSetCursorName, the driver uses the specified cursor. If the application does not connect the statement and the cursor, then the driver creates a cursor name.

If the data source is the manual commit mode, and the transaction has not yet been started, the driver starts the transaction before sending the SQL statement.

If SQLExecDirect finds the parameter of data-at-execution, it returns SQL_NEED_DATA. The application transmits data by using SQLParamData and SQLPutData.

If SQLExecDirect executes the statement such as searched update, insert, or delete, but rows have not been changed on the data source, then calling SQLExecDirect returns SQL_NO_DATA.

If the attribute value of SQL_ATTR_PARAMSET_SIZE statement is bigger than 1, and the SQL statement in cludes at least one parameter marker character, then SQLExecDirect executes the SQL statement once pe r a parameter set in the array pointed by ParameterValuePtr argument of SQLBindParameter.

SQLExecute

Conformance

Introduced version: ODBC 1.0
Standards compliance: ISO 92

Overview

If a statement includes a parameter, SQLExecute performs the prepared statement by using the current v alue of the parameter marker.

Syntax

```
SQLRETURN SQLExecute(
    SQLHSTMT     StatementHandle);
```

Arguments

StatementHandle

[Input] It is the statement handle.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NEED_DATA, SQL_STILL_EXECUTING, SQL_ERROR, SQL _NO_DATA, SQL_INVALID_HANDLE, SQL_PARAM_DATA_AVAILABLE

Diagnosis

SQLSTATE	Error	Description
01004	String data, right truncated	The string or binary data returned to the input/output or output parameters are truncated. The right part of the string is truncated. (The function returns SQL_SUCCESS_WITH_INFO.)
01S02	Option value	It is temporarily replaced with a similar value because the attribute value of t he specified statement is not suitable for executing the operation. (SQLGetS tmtAttr can be called to see which value is temporarily changed.) The replac ed value is valid until the cursor is closed, and it is changed to the previous v alue when the cursor is closed.

SQLSTATE	Error	Description
	changed	The statement attributes which can be changed are as follows. SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ ATTR_KEYSET _SIZE, SQL_ATTR_MAX_LENGTH, SQL_ATTR_MAX_ROWS, SQL_ ATTR_QUE RY_TIMEOUT, SQL_ATTR_SIMULATE_CURSOR. (The function returns SQL_SUCCESS_WITH_INFO.)
07006	Restricted data type attribute violation	The data value identified by ValueType argument of SQLBindParameter can not be converted to the data type identified by ParameterType argument of SQLBindParameter. The data value returned to the parameter of SQL_PARAM_INPUT_OUTPUT o r SQL_PARAM_OUTPUT can not be converted to the data type identified by ValueType argument of SQLBindParameter. (If one or more rows are successfully returned, the function returns SQL_SU CCESS_WITH_INFO.)
07007	Restricted parameter value violation	The parameter type is SQL_PARAM_INPUT_OUTPUT, and *StrLen_or_IndPtr of SQLBindParameter is not one of SQL_NULL_DATA, SQL_DEFAULT_PARA M, SQL_LEN_DATA_AT_EXEC(len), or SQL_DATA_AT_EXEC.
08S01	Communication link failure	Before the function processing is completed, the connection between the dr iver and the data source is failed.
22001	String data, right truncation	The string, or binary data is truncated.
22002	Indicator variable req uired but not supplied	NULL data is bound to the output parameter whose of StrLen_or_IndPtr of S QLBindParameter is the null pointer.
24000	Invalid cursor state	The cursor is positioned in StatementHandle through SQLFetch, SQLFetchScr oll. The cursor is open but it is not positioned in StatementHandle. *StatementText is the positioned update or delete statement, and the cursor is positioned before the start or after the end of the result set.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY009	Invalid use of null pointer	*StatementText is the null pointer.
HY010	Function sequence er ror	After calling SQLExecute, SQLExecDirect, then SQL_NEED_DATA is returned and the function is called before sending all data-at-execution variables.
		The argument value of TextLength is smaller than 0, but it is not SQL_NTS.

SQLSTATE	Error	Description
HY090	Invalid string or buffer length	The parameter value set by SQLBindParameter is the null pointer, and the parameter length is not 0, SQL_NULL_DATA, SQL_DATA_AT_EXEC, SQL_DEFAULT_PARAM, or less than SQL_LEN_DATA_AT_EXEC_OFFSET. The parameter value set by SQLBindParameter is not the null pointer, and C data type is SQL_C_BINARY or SQL_C_CHAR, and the parameter length is smaller than 0, but it is not less than SQL_NTS, SQL_NULL_DATA, SQL_DATA_AT_EXEC, SQL_DEFAULT_PARAM, or equal to or less than SQL_LEN_DATA_AT_EXEC_OFFSET.
HYT00	Timeout expired	Before returning the result set from the data source, the query timeout is expired. The timeout can be set through SQL_ATTR_QUERY_TIMEOUT of SQLSetStmtAttr.

Description

SQLExecute executes the statement prepared by SQLPrepare. The application can call SQLExecute again with the new parameter value after it discards the result of calling SQLExecute.

To execute SELECT statement once or more, the application should call SQLCloseCursor before executing SELECT statement again.

If the data source is a manual commit mode, and the transaction has not yet been started, the driver starts the transaction before sending the SQL statement.

If SQLExecute finds a data-at-execution parameter, it returns SQL_NEED_DATA. The application sends data by using SQLParamData and SQLPutData.

If SQLExecute executes the statement such as searched update, insert, or delete, but rows have not been changed on the data source, calling SQLExecute returns SQL_NO_DATA.

If the attribute value of SQL_ATTR_PARAMSET_SIZE statement is bigger than 1, and the SQL statement includes at least one parameter marker character, then SQLExecute executes the SQL statement once per a parameter set in the array pointed by ParameterValuePtr argument of SQLBindParameter.

SQLExtendedFetch

Conformance

Introduced version: ODBC 1.0.
Compliance: It is not available.

Overview

SQLExtendedFetch fetches the specified data set from the result set, and returns it to all bound columns.

Syntax

```
SQLRETURN SQLExtendedFetch(
    SQLHSTMT         StatementHandle,
    SQLUSMALLINT     FetchOrientation,
    SQLLEN           FetchOffset,
    SQLULEN *        RowCountPtr,
    SQLUSMALLINT *   RowStatusArray);
```

SQLFetch

Conformance

Introduced version: ODBC 1.0
Standards compliance: ISO 92

Overview

SQLFetch fetches the next row set from the result set, and returns it to all columns bound.

Syntax

```
SQLRETURN SQLFetch(
    SQLHSTMT     StatementHandle);
```

Arguments

StatementHandle

[Input] It is the statement handle.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01004	String data, right truncated	The string, binary data returned for the columns are truncated. The right part of the string is truncated.
01S07	Fractional truncation	The data returned for the column is truncated. For numeric data types, the decimal place is truncated. For time, timestamp, interval data types which contain the period component, the decimal place of time is truncated. (The function returns SQL_SUCCESS_WITH_INFO.)
07006	Restricted data type attribute violation	The column data value in the result set can not be converted to the data type specified by TargetType of SQLBindCol.

SQLSTATE	Error	Description
08S01	Communication link failure	Before the function processing is completed, the connection between the driver and the data source is failed.
22002	Indicator variable required but not supplied	StrLen_or_IndPtr (or SQL_DESC_INDICATOR_PTR set by SQLSetDescField or SQLSetDescRec) of SQLBindCol fetches NULL data to the column which is the null pointer.
22003	Numeric value out of range	The integer part (not the decimal place) of the numerical value returned from one or more columns is truncated.
22007	Invalid datetime format	The string in the result set is not the valid date, time, timestamp format.
22012	Division by zero	The result of the arithmetic expression divided by 0 is returned.
22015	Interval field overflow	When the interval C type is specified in the exact numeric or interval SQL data type, the significant figures in the leading field is lost. The value of SQL type can not be expressed to C interval type.
22018	Invalid character value for cast specification	The character not represented as the character set of C buffer is included in the character column of the result set. The C type is the exact or approximate numeric, datetime, interval data type, and if the SQL type is the character data type, the value of the column bound to the C type is not valid.
24000	Invalid cursor state	StatementHandle is executed but the result set related to StatementHandle does not exist.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY010	Function sequence error	The StatementHandle specified is not at the state of running. The function is called without calling SQLExecDirect, SQLExecute, the catalog function. After calling SQLExecute, SQLExecDirect, then SQL_NEED_DATA is returned and the function is called before sending all data-at-execution variables.
HYT00	Timeout expired	Before returning the result set from the data source, the query timeout is expired. The timeout can be set through SQL_ATTR_QUERY_TIMEOUT of SQLSetStmtAttr.

Description

SQLFetch returns the next data set from the result set. SQLFetch can be called while the result set exists, which is after the result set is created and before the cursor is closed. If the column is bound, the data is returned to the column. If the application specifies the pointer to a row status array or specifies the buffer which returns the number of fetched rows, SQLFetch also returns this information. SQLFetch and SQLFetchScroll can be called being mixed together.

Cursor Position

If the result set is generated, the cursor is positioned before the start of result set. SQLFetch fetches the next row set. It is as same as calling SQLFetchScroll which FetchOrientation is set to SQL_FETCH_NEXT.

The attribute of SQL_ATTR_ROW_ARRAY_SIZE statement specifies the number of rows in the row set. If the row set fetched by SQLFetch overlaps with the end of result set, SQLFetch returns a partial row set. If S is the start row of fetched row set, R is the size of row set, L is the last row of result set, S + R - 1 is bigger than L, then only the first L - S + 1 rows of row set is valid. All remaining rows are empty, and it becomes the SQL_ROW_NOROW state.

After returning SQLFetch, the current row is the first row of row set.

The rules in the following table describes the cursor position according to the conditions in the second table of the session after calling SQLFetch.

Status	The first row of new row set
Before start	1
CurrRowsetStart <= LastResultRow - RowsetSize[1]	CurrRowsetStart + RowsetSize[2]
CurrRowsetStart > LastResultRow - RowsetSize[1]	After end
After end	After end

[1] If the row set size is changed during fetching, the row set size is the row set size used before fetching.
[2] If the row set size is changed during fetching, the row set size is the row set size used in new fetch.

Notation	Description
Before start	The block cursor is positioned before the start of result set. If the first row of new row set is before the start of result set, SQLFetch returns SQL_NO_DATA.
After end	The block cursor is positioned after the end of result set. If the first row of new row set is after the end of result set, SQLFetch returns SQL_NO_DATA.
CurrRowsetStart	It is the first row number of the current row set.
LastResultRow	It is the last row number of result set.
RowsetSize	It is the row set size.

The following is an example of when 100 rows exist in the result set and the row set size is 5, then the following table describes the row set and the return code returned by SQLFetch for the different start position.

Current row set	Return code	New row set	The number of fetched rows
Before start	SQL_SUCCESS	1 to 5	5
1 to 5	SQL_SUCCESS	6 to 10	5
52 to 56	SQL_SUCCESS	57 to 61	5

Current row set	Return code	New row set	The number of fetched rows
91 to 95	SQL_SUCCESS	96 to 100	5
93 to 97	SQL_SUCCESS	98 to 100 4,5 rows of the row status array are set to SQL_ROW_NOROW.	3
96 to 100	SQL_NO_DATA	None	0
99 to 100	SQL_NO_DATA	None	0
After end	SQL_NO_DATA	None	0

Returing the Data in the Bound Column

Like as SQLFetch returns each row, it inserts the data into each bound column in the buffer bound to the column. If the bound column does not exist, SQLFetch does not return any data, but it does not move the cursor forward. Data can be continuously fetched through SQLGetData. If the cursor is a multiple row cur sor (SQL_ATTR_ROW_ARRAY_SIZE is bigger than 1.), SQLGetData can be called when after setting InfoT ype of SQLGetInfo to SQL_FETDATA_EXTENSIONS, or when returning SQL_GD_BLOCK.
For more information, refer to **SQLGetData**.

SQLFetch executes the followings for each bound columns in the row.

1. If data is NULL, the length/indicator buffer is set to SQL_NULL_DATA and the next row is processed. I f the data is NULL and the length/indicator buffer is not bound, SQLFetch returns SQLSTATE 22002 (Indicator variable required but not supplied) for the row and the next row to be processed. For mor e information about how to determine the address of the length/indicator, refer to **Buffer Address** o f SQLBindCol. If the column data is not NULL, SQLFetch executes the process number 2.

2. If the attribute of SQL_ATTR_MAX_LENGTH statement is set to non-zero and the column includes th e character or binary data, the data is truncated to the length of SQL_ATTR_MAX_LENGTH in bytes.

The attribute of SQL_ATTR_MAX_LENGTH statement is intended to reduce network traffic. It is usually implemented by the data source, and the data is truncated before the data is returned fr om the network. Drivers and data sources are not required to support it. Therefore, the applicati on should specify the size to cbValueMax argument of SQLBindCol and creates the buffer to gu arantee that the data is truncated to the specific size.

3. The data is converted to the format specified in TargetType of SQLBindCol.

4. If the data is converted to the variable length data type such as a string or binary, SQLFetch confirms if the data length exceeds the data buffer length. If the character data (including NULL termination c haracter) exceeds the data buffer length, SQLFetch truncates the data to the data buffer length whic h is smaller than NULL termination character length. In this case, the data is terminated by NULL. If t he binary data length exceeds the data buffer length, SQLFetch truncates the data to the data buffer

length. The data buffer length is specified in BufferLength of SQLBindCol. SQLFetch never truncate the data converted to the fixed length data format because the data buffer length is always equal to the data type length.

5. The converted data (truncated data if possible) is put in the data buffer. For more information about how to determine the data buffer address, refer to **Buffer Address** of SQLBindCol.

6. The data length is put in the length/indicator buffer. If both the length pointer and indicator pointer are set in the same buffer (by calling SQLBindCol), the valid data length is recorded in the buffer, and SQL_NULL_DATA is recorded in the buffer if the data is NULL. If the length/indicator buffer is not bound, SQLFetch does not return the length.

- For the character or binary data, it is the data length before truncated after the data is converted because of too small buffer size. If the driver can not determine the length of too long data after conversion, the length is set to SQL_NO_TOTAL. If the data is truncated due to the attribute of SQL_ATTR_MAX_LENGTH statement, this attribute value is put in the length/indicator buffer instead of the actual length. It is because the attribute is designed to truncate the data in the server before conversion. So, the driver can not calculate the actual length.
- For all other data types, it is the data length after conversion.

7. If the data is truncated without any lost of significant digits during conversion(for example, the real number 1.234 is converted into 1 by truncation.), SQLFetch returns SQLSTATE 01S07(Fractional truncation) and SQL_SUCCESS_WITH_INFO. If the data is truncated because the data buffer length is too small (For example the string "abcdef" is put in the buffer of four bytes.), SQLFetch returns SQLSTATE 01004 (Data truncated) and SQL_SUCCESS_WITH_INFO. If the data is truncated because of the attribute of SQL_ATTR_MAX_LENGTH statement, SQLFetch returns SQL_SUCCESS and it does not return SQLSTATE 01S07(Fractional truncation) nor SQLSTATE 01004(Data truncated). If the significant digits of data is truncated while the data is converted (for example, the value of SQL_INTEGER which is bigger than 100,000 is converted into SQL_C_TINYINT.), SQLFetch returns SQLSTATE 22003(Numeric value out of range), SQL_ERROR (If the row set size is 1.) or SQL_SUCCESS_WITH_INFO (If the row set size is bigger than 1.).

If SQLFetch, SQL_SUCCESS of SQLFetchScroll, or SQL_SUCCESS_WITH_INFO is not returned, then the contents of the bound data buffer and the length/indicator buffer are not defined.

Row Status Array

The row status array is used to return the status of each row set. The array address is specified in the attribute of SQL_ATTR_ROW_STATUS_PTR statement. The array should assign the elements as many as specified by the attribute of SQL_ATTR_ROW_ARRAY_SIZE statement in the application. The value is set by SQLFetch, SQLFetchScroll, SQLBulkOperations or SQLSetPos. If the attribute value of SQL_ATTR_ROW_STATUS_PTR statement is the null pointer, the function does not return the row status.

The content of row status buffer is not defined if SQL_SUCCESS or SQL_SUCCESS_WITH_INFO is not returned by SQLFetch, SQLFetchScroll.

The following values are returned to the row status array.

The value of row status array	Description
SQL_ROW_SUCCESS	The row is successfully fetched, and it is not changed after the last fetch from the result set.
SQL_ROW_SUCCESS_WITH_INFO	The row is successfully fetched, and it is not changed after the last fetch from the result set. But the warning about the row is returned.
SQL_ROW_ERROR	An error occurs while the row is fetched.
SQL_ROW_UPDATED	The row is successfully fetched, and it is changed after the last fetch from the result set. If the row is fetched again or it is refreshed by SQLSetPos. Status is changed to the new row status.
SQL_ROW_DELETED	It is deleted after the row is fetched last from the result set.
SQL_ROW_NOROW	The row set is overlapped with the end of result set, and returns that there is not a row.

Row Fetch Buffer

The row fetch buffer is used to return the number of fetched rows. When the data is fetched, the row without data due to an error is also included. It is the number of rows which is not SQL_ROW_NOROW value in the row status array. This buffer address is specified in the arribute of SQL_ATTR_ROWS_FETCHED_PTR statement. The buffer is allocated by the application, and it is set by SQLFetch, SQLFetchScroll. If the attribute value of SQL_ATTR_ROWS_FETCHED_PTR statement is the null pointer, the function does not return the number of fetched rows. The application calls SQLGetStmtAttr as the attribute of SQL_ATTR_ROW_NUMBER to determine the number of current rows in the result set.

The content of row fetch buffer is not defined if SQLFetch, SQLFetchScroll does not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO. If SQL_NO_DATA is returned, the value of row fetch buffer is set to 0.

Error Processing

Errors and warnings can be applied to the individual row or the entire function.

Error and Warning for Entire Function

If the error is applied to the entire function, as like SQLSTATE HYT00 (Timeout expired) or SQLSTATE 24000 (Invalid cursor state), SQLFetch returns SQL_ERROR and the corresponding SQLSTATE. The content of row buffer is not defined and the cursor position is not changed.

If the warning is applied to the entire function, SQLFetch returns SQL_SUCCESS_WITH_INFO and the corresponding SQLSTATE. The warning applied to the entire function is returned before the state is recorded to each row.

Error and Warning for Individual Row

The error such as SQLSTATE 22012 (division by zero) or the warning such as SQLSTATE 01004 (data trun cated) is applied to the individual row.

SQLFetch executes the followings.

- It sets the element of row status array to SQL_ROW_ERROR for an error or SQL_ROW_SUCCESS_WIT H_INFO for a warning.
- It adds one or more records which includes SQLSTATE for the error or warnig.
- It sets the row and column number fields in the status record. If SQLFetch can not determine the row or column number, it sets each number to SQL_ROW_NUMBER_UNKNOWN or SQL_COLUMN_NUM BER_UNKNOWN. If the status record is not applied to the specified column, SQLFetch sets the colum n number to SQL_NO_COLUMN_NUMBER.

If an error occurs in all rows of the row set (excluding the row in SQL_ROW_NOROW status), SQLFetch re turns SQL_ERROR. If an error occurs in some rows, it returns SQL_SUCCESS_WITH_INFO. If the row set siz e is 1 and an error occurs in the row, SQLFetch returns SQL_ERROR.

Descriptor and SQLFetch

SQLFetch uses the following descriptor fields.

Descriptor field	Descriptor	Field location	Setting
SQL_DESC_ARRAY_SIZE	ARD	header	SQL_ATTR_ROW_ARRAY_SIZE statement att ribute
SQL_DESC_ARRAY_STATUS_PTR	IRD	header	SQL_ATTR_ROW_STATUS_PTR statement att ribute
SQL_DESC_BIND_OFFSET_PTR	ARD	header	SQL_ATTR_ROW_BIND_OFFSET_PTR statem ent attribute
SQL_DESC_BIND_TYPE	ARD	header	SQL_ATTR_ROW_BIND_TYPE statement attri bute
SQL_DESC_COUNT	ARD	header	ColumnNumber argument of SQLBindCol
SQL_DESC_DATA_PTR	ARD	record	TargetValuePtr argument of SQLBindCol
SQL_DESC_INDICATOR_PTR	ARD	record	StrLen_or_IndPtr argument of SQLBindCol
SQL_DESC_OCTET_LENGTH	ARD	record	BufferLength argument of SQLBindCol
SQL_DESC_OCTET_LENGTH_PTR	ARD	record	StrLen_or_IndPtr argument of SQLBindCol
SQL_DESC_ROWS_PROCESSED_ PTR	IRD	record	SQL_ATTR_ROWS_FETCHED_PTR statement attribute
SQL_DESC_TYPE	ARD	record	TargetType argument of SQLBindCol

All descriptor fields can be set through SQLSetDescField.

Separating Length and Indicator Buffer

The application can bind one or two buffers to store the length and indicator value. If the application call s SQLBindCOl, SQL_DESC_OCTET_LENGTH_PTR and SQL_DESC_INDICATOR_PTR field of ARD is set in th e address passed to StrLen_or_IndPtr argument. The application can set the two fields to another address es by calling SQLSetDescField or SQLSetDescRec.

SQLFetch determines whether the application specifies a separate length and indicator buffer. If the data is not NULL, SQLFetch sets the indicator buffer to 0, and returns the length to the length buffer. If the dat a is NULL, SQLFetch sets the indicator buffer to SQL_NULL_DATA, and it does not modify the length buff er.

SQLFetchScroll

Conformance

Introduced version: ODBC 3.0
Standards compliance: ISO 92

Overview

SQLFetchScroll fetches the specified data set from the result set, and returns it to all bound columns.

Syntax

```
SQLRETURN SQLFetchScroll(
    SQLHSTMT      StatementHandle,
    SQLSMALLINT   FetchOrientation,
    SQLLEN        FetchOffset);
```

Arguments

StatementHandle

[Input] It is the statement handle.

FetchOrientation

[Input] It is the fetch type: SQL_FETCH_NEXT, SQL_FETCH_PRIOR, SQL_FETCH_FIRST, SQL_FETCH_LAST, SQL_FETCH_ABSOLUTE, SQL_FETCH_RELATIVE, SQL_FETCH_BOOKMARK

FetchOffset

[Input] It is the number of rows to fetch. The interpretation of this argument depends on the value of FetchOrientation argument.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01004	String data, right truncated	The string, binary data returned for the columns are truncated. The right part of the string is truncated.
01S07	Fractional truncation	The data returned for the column is truncated. For numeric data types, the decimal place is truncated. For time, timestamp, interval data types which contain the period component, the decimal place of time is truncated. (The function returns SQL_SUCCESS_WITH_INFO.)
07006	Restricted data type attribute violation	The column data value in the result set can not be converted to the data type specified by TargetType of SQLBindCol.
08S01	Communication link failure	Before the function processing is completed, the connection between the driver and the data source is failed.
22002	Indicator variable required but not supplied	StrLen_or_IndPtr (or SQL_DESC_INDICATOR_PTR set by SQLSetDescField or SQLSetDescRec) of SQLBindCol fetches NULL data to the column which is the null pointer.
22003	Numeric value out of range	The integer part (not the decimal place) of the numerical value returned from one or more columns is truncated.
22007	Invalid datetime format	The string in the result set is not the valid date, time, timestamp format.
22012	Division by zero	The result of the arithmetic expression divided by 0 is returned.
22015	Interval field overflow	When the interval C type is specified in the exact numeric or interval SQL data type, the significant figures in the leading field is lost. The value of SQL type can not be expressed to C interval type.
22018	Invalid character value for cast specification	The character not represented as the character set of C buffer is included in the character column of the result set. The C type is the exact or approximate numeric, datetime, interval data type, and if the SQL type is the character data type, the value of the column bound to the C type is not valid.
24000	Invalid cursor state	StatementHandle is executed but the result set related to StatementHandle does not exist.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY010	Function sequence error	The StatementHandle specified is not at the state of running. The function is called without calling SQLExecDirect, SQLExecute, the catalog function. After calling SQLExecute, SQLExecDirect, then SQL_NEED_DATA is returned and the function is called before sending all data-at-execution variables.
		The value specified to FetchOrientation argument is not valid.

SQLSTATE	Error	Description
HY106	Fetch type out of range	The attribute value of SQL_ATTR_CURSOR_TYPE statement is SQL_CURSOR_FORWARD_ONLY, and the value of FetchOrientation argument is not SQL_FETCH_NEXT. The attribute value of SQL_ATTR_CURSOR_SCROLLABLE statement is SQL_NONSCROLLABLE, and the value of FetchOrientation argument is not SQL_FETCH_NEXT.
HYT00	Timeout expired	Before returning the result set from the data source, the query timeout is expired. The timeout can be set through SQL_ATTR_QUERY_TIMEOUT of SQLSetStmtAttr.

Description

SQLFetchScroll returns the specified row set in the result set. The row set is set as the absolute or relative position or the bookmark position. SQLFetchScroll can be called while the result set exists, which is after the result set is created and before the cursor is closed. If the column is bound, the data is returned to the column. If the application specifies the pointer to a row status array or specifies the buffer which returns rows fetched, SQLFetchScroll also returns this information. SQLFetch and SQLFetchScroll can be called being mixed together.

Cursor Position

If the result set is created, the cursor is positioned before the start of result set. SQLFetchScroll positions the block cursor according to FetchOrientation argument and FetchOffset argument as follows. The following rules are used to determine the beginning of new row set.

FetchOrientation	Description
SQL_FETCH_NEXT	It returns the next row set. it is as same as calling SQLFetch. SQLFetchScroll ignores the value of FetchOffset.
SQL_FETCH_PRIOR	It returns the previous row set. SQLFetchScroll ignores the value of FetchOffset.
SQL_FETCH_RELATIVE	It returns the row set of FetchOffset at the beginning of current row set.
SQL_FETCH_ABSOLUTE	It returns the row set starting at FetchOffset.
SQL_FETCH_FIRST	It returns the first row set in the result set. SQLFetchScroll ignores the value of FetchOffset.
SQL_FETCH_LAST	It returns the last complete row set in the result set. SQLFetchScroll ignores the value of FetchOffset.
SQL_FETCH_BOOKMARK	It returns the row set of FetchOffset on the bookmark specified by the attribute of SQL_ATTR_FETCH_BOOKMARK_PTR statement.

The attribute of SQL_ATTR_ROW_ARRAY_SIZE statement specifies the number of rows of the row set. If

the row set fetched by SQLFetchScroll overlaps with the end of result set, SQLFetchScroll returns a partial row set. If S is the start row of fetched row set, R is the row set size, L is the last row of result set, S + R - 1 is bigger than L, and the first L - S + 1 rows of row set is valid. All remaining rows are empty, and it bec omes SQL_ROW_NOROW state.

After returning SQLFetchScroll, the current row is the first row of row set.

Cursor Position Rules

The following chapters describe rules for each FetchOrientation. The following notations are used for the rules.

Notation	Meaning
Before start	The block cursor is positioned before the start of result set. If the first row of the new row set is before the start of the result set, SQLFetchScroll returns SQL_NO_DATA.
After end	The block cursor is positioned after the end of result set. If the first row of the new ro w set is after the end of the result set, SQLFetchScroll returns SQL_NO_DATA.
CurrRowsetStart	It is the first row number of current row set.
LastResultRow	It is the last row number of result set.
RowsetSize	It is the row set size.
FetchOffset	It is the value of FetchOffset argument.
BookmarkRow	It is the row which corresponds to the bookmark specified by the attribute of SQL_AT TR_FETCH_BOOKMARK_PTR statement.

SQL_FETCH_NEXT

The following rule is applied.

State	The first row of the new row set
Before start	1
CurrRowsetStart + RowsetSize[1] <= LastResultRow	CurrRowsetStart + RowsetSize[1]
CurrRowsetStart + RowsetSize[1] > LastResultRow	After end
After end	After end

[1] If the row set size is changed during fetching the row set size is the row set size used before fetching.

SQL_FETCH_PRIOR

The following rule is applied.

State	The first row of the new row set
Before start	Before start
CurrRowsetStart = 1	Before start
1 < CurrRowsetStart <= RowsetSize[1]	1

State	The first row of the new row set
CurrRowsetStart > RowsetSize[1]	CurrRowsetStart - RowsetSize[1]
After end AND LastResultRow < RowsetSize[1]	1
After end AND LastResultRow >= RowsetSize[1]	LastResultRow - RowsetSize + 1[1]

[1] If the row set size is changed during fetching, the row set size is the row set size used in new fetch.

SQL_FETCH_RELATIVE

The following rule is applied.

State	The first row of the new row set
(Before start AND FetchOffset > 0) OR (After end AND FetchOffset < 0)	__ [1]
BeforeStart AND FetchOffset <= 0	Before start
CurrRowsetStart = 1 AND FetchOffset < 0	Before start
CurrRowsetStart > 1 AND CurrRowsetStart + FetchOffset < 1 AND | FetchOffset | > RowsetSize[2]	Before start
CurrRowsetStart > 1 AND CurrRowsetStart + FetchOffset < 1 AND | FetchOffset | <= RowsetSize[2]	1
1 <= CurrRowsetStart + FetchOffset <= LastResultRow	CurrRowsetStart + FetchOffset
CurrRowsetStart + FetchOffset > LastResultRow	After end
After end AND FetchOffset >= 0	After end

[1] SQLFetchScroll returns the row set as same as when FetchOrientation is set to SQL_FETCH_ABSOLUTE and it is called.
[2] If the row set size is changed during fetching, the row set size is the row set size used in new fetch.

SQL_FETCH_ABSOLUTE

The following rule is applied.

State	The first row of the new row set
FetchOffset < 0 AND | FetchOffset | <= LastResultRow	LastResultRow + FetchOffset + 1
FetchOffset < 0 AND | FetchOffset | > LastResultRow AND | FetchOffset | > RowsetSize[1]	Before start
FetchOffset < 0 AND | FetchOffset | > LastResultRow AND | FetchOffset | <= RowsetSize[1]	1
FetchOffset = 0	Before start
1 <= FetchOffset <= LastResultRow	FetchOffset
FetchOffset > LastResultRow	After end

[1] If the row set size is changed during fetching, the row set size is the row set size used in new fetch.

SQL_FETCH_FIRST

The following rule is applied.

State	The first row of the new row set
Any	1

SQL_FETCH_LAST

The following rule is applied.

State	The first row of the new row set
RowsetSize[1]<= LastResultRow	LastResultRow - RowsetSize + 1[1]
RowsetSize[1]> LastResultRow	1

[1]If the row set size is changed during fetching, the row set size is the row set size used in new fetch.

SQL_FETCH_BOOKMARK

The following rule is applied.

State	The first row of the new row set
BookmarkRow + FetchOffset < 1	Before start
1 <= BookmarkRow + FetchOffset <= LastResultRow	BookmarkRow + FetchOffset
BookmarkRow + FetchOffset > LastResultRow	After end

Buffer Address

SQLFetchScroll determines the data address and length/indicator buffer address in the same way as SQLFetch. For more information, refer to **Buffer Address** of SQLBindCol.

Row Status Array

SQLFetchScroll sets the row status array in the same way as SQLFetch. For more information, refer to **Row Status Array** of SQLFetch.

Row Fetch Buffer

SQLFetchScroll returns the number of rows fetched in the same way as SQLFetch. For more information, refer to **Row Fetch Buffer** of SQLFetch.

Error Processing

SQLFetchScroll returns the errors and warning in the same way as SQLFetch. For more information, refer t o **Error Processing** of SQLFetch.

SQLForeignKeys

Conformance

Introduced version: ODBC 1.0
Standards compliance: ODBC

Overview

SQLForeignKeys returns the followings.

- The foreign key list of the specified table (the column in the specified table which refers to the the pri mary key of another table)
- The foreign key list of another table which refers to the primary key of the specified table

The driver returns each list as the result set.

Syntax

```
SQLRETURN SQLForeignKeys(
    SQLHSTMT       StatementHandle,
    SQLCHAR *      PKCatalogName,
    SQLSMALLINT    NameLength1,
    SQLCHAR *      PKSchemaName,
    SQLSMALLINT    NameLength2,
    SQLCHAR *      PKTableName,
    SQLSMALLINT    NameLength3,
    SQLCHAR *      FKCatalogName,
    SQLSMALLINT    NameLength4,
    SQLCHAR *      FKSchemaName,
    SQLSMALLINT    NameLength5,
    SQLCHAR *      FKTableName,
    SQLSMALLINT    NameLength6);
```

Arguments

StatementHandle

[Input] It is the statement handle.

PKCatalogName

[Input] It is the primary key table catalog name. PKCatalogName can not include the string search pattern.

If the attribute of SQL_ATTR_METADATA_ID statement is set to SQL_TRUE, then PKCatalogName i s treated as a case-insensitive identifier. If it is set to SQL_FALSE, then PKCatalogName is case-sensi tive, and it is an ordinary argument literally processed.

NameLength1

[Input] It is the length of *PKCatalogName

PKSchemaName

[Input] It is the primary key table schema name. PKSchemaName can not include the string search pattern.

If the attribute of SQL_ATTR_METADATA_ID statement is set to SQL_TRUE, then PKSchemaName i s treated as a case-insensitive identifier. If it is set to SQL_FALSE, then PKSchemaName is case-sensi tive, and it is an ordinary argument literally processed.

NameLength2

[Input] It is the length of *PKSchemaName

PKTableName

[Input] It is the primary key table name. PKTableName can not include the string search pattern.

If the attribute of SQL_ATTR_METADATA_ID statement is set to SQL_TRUE, then PKTableName is t reated as a case-insensitive identifier. If it is set to SQL_FALSE, then PKTableName is case-sensitive, and it is an ordinary argument literally processed.

NameLength3

[Input] It is the length of *PKTableName

FKCatalogName

[Input] It is the foreign key table catalog name. FKCatalogName can not include the string search p attern.

If the attribute of SQL_ATTR_METADATA_ID statement is set to SQL_TRUE, then FKCatalogName i s treated as a case-insensitive identifier. If it is set to SQL_FALSE, then FKCatalogName is case-sensi tive, and it is an ordinary argument literally processed.

NameLength4

[Input] It is the length of *FKCatalogName

FKSchemaName

[Input] It is the foreign key table schema name. FKSchemaName can not include the string search p attern.

If the attribute of SQL_ATTR_METADATA_ID statement is set to SQL_TRUE, FKSchemaName is trea ted as a case-insensitive identifier. If it is set to SQL_FALSE, FKSchemaName is case-sensitive, and it is an ordinary argument literally processed.

NameLength5

[Input] It is the length of *FKSchemaName

FKTableName

[Input] It is the foreign key table name. FKTableName can not include the string search pattern.

If the attribute of SQL_ATTR_METADATA_ID statement is set to SQL_TRUE, then FKTableName is t reated as a case-insensitive identifier. If it is set to SQL_FALSE, FKTableName is case-sensitive, and i t is an ordinary argument literally processed.

NameLength6

[Input] It is the length of *FKTableName

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
08S01	Communication l ink failure	Before the function processing is completed, the connection between the dr iver and the data source is failed.
24000	Invalid cursor stat e	SQLFetch, SQLFetchScroll are called and a cursor is open.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocatio n error	It is a memory allocation error.
HY009	Invalid use of null pointer	The PKTableName argument and FKTableName argument are the null point er. The attribute value of SQL_ATTR_METADATA_ID statement is SQL_TRUE, a nd the arguments of PKSchemaName, FKSchemaName, PKTableame or FKT ableame are the null pointer.
HY010	Function sequenc e error	After calling SQLExecute, SQLExecDirect, then SQL_NEED_DATA is returned and the function is called before sending all data-at-execution variables.
HY090	Invalid string or b uffer length	A name length argument value is smaller than 0, but it is not SQL_NTS.
HYT00	Timeout expired	Before returning the result set from the data source, the query timeout is ex pired. The timeout can be set through SQL_ATTR_QUERY_TIMEOUT of SQL SetStmtAttr.

Description

If *PKTableName includes the table name, SQLForeignKeys returns a result set which includes the primary key of the specified table and all foreign keys referring it. The foreign key list of another table does not in clude the foreign key pointing to the unique constraint on the specified table.

If *FKTableName includes the table name, SQLForeignKeys returns the result set which contains the forei gn key in the specified table pointing to the primary key of another table, and returns its primary key of a nother table which is referenced by them.

If both *PKTableName and *FKTableName include the table name, SQLForeignKeys returns the foreign k ey of the table specified in *FKTableName which refers to the primary key of the table specified in *PKTa bleName. The key should be one.

SQLForeignKeys returns the standard result set. If the foreign key related to the primary key is requested, the result set is sorted as FKTABLE_CAT, FKTABLE_SCHEM, FKTABLE_NAME, KEY_SEQ. If the primary key related to the foreign key is requested, the result set is sorted as PKTABLE_CAT, PKTABLE_SCHEM, PKTA BLE_NAME, KEY_SEQ.

The following table describes the columns of result set.

Column na me	Column number	Data type	Description
PKTABLE_C AT (ODBC 1 .0)	1	VARCHAR	It is the primary key table catalog name.
PKTABLE_S CHEM (ODB C 1.0)	2	VARCHAR	It is the primary key table schema name.
PKTABLE_N AME (ODBC 1.0)	3	VARCHAR n ot NULL	It is the primary key table name.
PKCOLUMN _NAME (OD BC 1.0)	4	VARCHAR n ot NULL	It is the primary key column name. The driver returns an empty string f or a column which does not have a name.
FKTABLE_C AT (ODBC 1 .0)	5	VARCHAR	It is the foreign key table catalog name.
FKTABLE_SC HEM (ODBC 1.0)	6	VARCHAR	It is the foreign key table schema name.
FKTABLE_N AME (ODBC 1.0)	7	VARCHAR n ot NULL	It is the foreign key table name.
FKCOLUMN		VARCHAR n	It is the foreign key column name. The driver returns an empty string fo

Column name	Column number	Data type	Description
_NAME (ODBC 1.0)	8	ot NULL	r a column which does not have a name.
KEY_SEQ (ODBC 1.0)	9	SMALLINT not NULL	It is the column sequential number of the key starting from 1.
UPDATE_RULE (ODBC 1.0)	10	SMALLINT	It is the operation applied to the foreign key when the SQL operation is UPDATE. (The referenced table has the primary key, and the referring table has the foreign key.) • SQL_CASCADE: If the primary key of referencing table is updated, the foreign key of the referring table is also updated. • SQL_NO_ACTION: If a row in the referring table does not corresponds to the referenced table when updating the primary key of the referenced table, the update is rejected. If the foreign key update of the referring table does not exist as the value of the primary key of the referenced table, the update is rejected. • SQL_SET_NULL: If one or more rows in the referenced table are updated in a way that one or more components of the primary key are changed, the components of the foreign key in the referring table which corresponds to the changed components of the primary key are set to NULL in all matching rows of the referring table. • SQL_SET_DEFAULT: If one or more rows in the referenced table are updated in a way that one or more components of the primary key are changed, the components of the foreign key in the referring table which corresponds to the changed components of the primary key are set to the default value in all matching rows of the referring table.
DELETE_RULE (ODBC 1.0)	11	SMALLINT	It is the operation applied to the foreign key when the SQL operation is DELETE. (The referenced table has the primary key, and the referring table has the foreign key.) • SQL_CASCADE: If the primary key of referenced table is deleted, the foreign key of the referring table is also deleted. • SQL_NO_ACTION: If a row in the referring table does not corresponds to the referenced table when deleting the primary key of the referenced table, the update is rejected. • SQL_SET_NULL: If one or more rows of the referenced table are deleted, each components of the foreign key of the referring table is set to NULL in all matching rows of the referring table. • SQL_SET_DEFAULT: If one or more rows of the referenced table a

Column name	Column number	Data type	Description
			re deleted, each component of the foreign key of the referencing table is set to the applicable default in all matching rows of the referencing table.
FK_NAME (ODBC 2.0)	12	VARCHAR	It is the foreign key name.
PK_NAME (ODBC 2.0)	13	VARCHAR	It is the primary key name.
DEFERRABILITY (ODBC 3.0)	14	SMALLINT	SQL_INITIALLY_DEFERRED, SQL_INITIALLY_IMMEDIATE, SQL_NOT_DEFERRABLE.

SQLFreeConnect

Conformance

Introduced version: ODBC 1.0
Standards compliance: It is not available.

Overview

SQLFreeConnect function is replaced with SQLFreeHandle function in ODBC 3.x.
For more information, refer to **SQLFreeHandle**.

SQLFreeEnv

Conformance

Introduced version: ODBC 1.0
Standards compliance: It is not available.

Overview

SQLFreeEnv function is replaced with SQLFreeHandle function in ODBC 3.x.
For more information, refer to **SQLFreeHandle**.

SQLFreeHandle

Conformance

Introduced version: ODBC 3.0
Standards compliance: ISO 92

Overview

SQLFreeHandle releases the resources related to the specified environment, connection, statement, and d escriptor handles.

Syntax

```
SQLRETURN SQLFreeHandle(
    SQLSMALLINT   HandleType,
    SQLHANDLE     Handle);
```

Arguments

HandleType

[Input] It is the handle type to be released by using SQLFreeHandle. It should be one of SQL_HAND LE_DBC, SQL_HANDLE_DESC, SQL_HANDLE_ENV, SQL_HANDLE_STMT.

If HandleType is not one of the values above, SQLFreeHandle returns SQL_INVALID_HANDLE.

Handle

[Input] It is the handle to be released.

Returns

SQL_SUCCESS, SQL_ERROR, SQL_INVALID_HANDLE.
If SQLFreeHandle returns SQL_ERROR, the handle is still valid.

Diagnosis

SQLSTATE	Error	Description
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocatio	It is a memory allocation error.

SQLSTATE	Error	Description
	n error	
HY010	Function sequence error	HandleType argument is SQL_HANDLE_ENV, and at least one connection is assigned or connected. Before calling SQLFreeHandle whose HandleType argument is SQL_HANDLE_ENV, SQLDisconnect and SQLFreeHandle argument whose HandleType argument is SQL_HANDLE_DBC should be called. HandleType argument is SQL_HANDLE_DBC, the function is called before SQLDisconnect is called. HandleType argument is SQL_HANDLE_STMT, after calling SQLExecute, SQLExecDirect, then SQL_NEED_DATA is returned and the function is called before sending all data-at-execution variables.

Description

SQLFreeHandle is used to release the environment, connection, statement, descriptor handles.
After the handle is released, the application can not use the released handle.

Releasing the Environment Handle

Before calling SQLFreeHandle whose HandleType is SQL_HANDLE_ENV, the application should call SQLFreeHandle whose HandleType is SQL_HANDLE_DBC for all connection allocated from the environment. Otherwise, SQLFreeHandle returns SQL_ERROR, and the environment and active connection remain valid.

Releasing the Connection Handle

Before calling SQLFreeHandle whose HandleType is SQL_HANDLE_DBC, if the handle is connected, the application should call SQLDisconnect. Otherwise, SQLFreeHandle returns SQL_ERROR, and the connection remains valid.

Releasing the Statement Handle

SQLFreeHandle whose HandleType is SQL_HANDLE_STMT, releases all resources allocated by calling SQLAllocHandle whose HandleType is SQL_HANDLE_STMT. If the application calls SQLFreeHandle with the remaining statement, the remaining result is deleted. When the application releases the statement handle, the driver releases the four auto allocation descriptors related to the statement.

SQLDisconnect automatically deletes all statements and descriptors which are opened for the connection.

SQLFreeStmt

Conformance

Introduced version: ODBC 1.0
Standards compliance: ISO 92

Overview

SQLFreeStmt stops the processing related to the specified statement, closes an open cursor, removes the remaining result or selectively releases all resources connected to the statement handle.

Syntax

```
SQLRETURN SQLFreeStmt(
    SQLHSTMT       StatementHandle,
    SQLUSMALLINT   Option);
```

Arguments

StatementHandle

[Input] It is the statement handle.

Option

[Input] It is one of the following options.

• SQL_ CLOSE: It closes the cursor related to StatementHandle, and deletes the remaining result. T he applications executes SELECT statement with the same or different parameter values, and it can open cursor again later. If the cursor is not open, this option does not affect the application. SQLCl oseCursor also closes the cursor.

• SQL_DROP: This option is not used any more.

• SQL_UNBIND: It releases all column buffers bound with SQLBindCol for the specified Statement Handle, and sets SQL_DESC_COUNT field of ARD to 0.

• SQL_RESET_PARAMS: It releases all parameter buffers set to SQLBindParameter for the specified StatementHandle, and sets SQL_DESC_COUNT field of APD to 0.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY010	Function sequence error	After calling SQLExecute, SQLExecDirect, then SQL_NEED_DATA is returned and the function is called before sending all data-at-execution variables.
HY092	Option type out of range	The value of option argument is not SQL_CLOSE, SQL_DROP, SQL_UNBIND, SQL_RESET_PARAMS.

Description

Calling SQLFreeStmt with SQL_CLOSE option is as same as calling of SQLCloseCursor. However, if an open cursor does not exist, calling SQLFreeStmt with SQL_CLOSE option does not affect the application. SQLCloseCursor returns SQLSTATE 24000 (Invalid cursor state).

SQLGetConnectAttr

Conformance

Introduced version: ODBC 3.0
Standards compliance: ISO 92

Overview

SQLGetConnectAttr returns the current setting of the connection attribute.

Syntax

```
SQLRETURN SQLGetConnectAttr(
    SQLHDBC        ConnectionHandle,
    SQLINTEGER     Attribute,
    SQLPOINTER     ValuePtr,
    SQLINTEGER     BufferLength,
    SQLINTEGER *   StringLengthPtr);
```

Arguments

ConnectionHandle

[Input] It is the connection handle.

Attribute

[Input] It is the attribute to be searched

ValuePtr

[Output] It is a memory pointer which returns the current setting value of the attribute specified by Attribute.

If ValuePtr is null, StringLengthPtr returns the total number of bytes returnable (excluding the null-termination character).

BufferLength

[Input] If Attribute is a defined field in ODBC and ValuePtr points to the string or binary buffer, this argument should be the length of *ValuePtr. If Attribute is a defined field in ODBC and *ValuePtr i s an integer, this argument is ignored.

StringLengthPtr

[Output] It is the pointer which returns the total number of bytes returnable in *Value_Ptr (excluding null-termination byte for character data)

For the character data, if the number of returnable bytes is equal to or bigger than BufferLength, *ValuePtr is truncated to length of BufferLength minus 1, and it is null terminated by the driver.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01004	String data, right truncated	The data truncated to the length of BufferLength minus null-termination character is returned to *ValuePtr. The length of string not truncated is returned to *StringLengthPtr. (The function returns SQL_SUCCESS_WITH_INFO.)
08003	Connection not open	The Attribute value required on the connection status is specified.
08S01	Communication link failure	Before the function processing is completed, the connection between the driver and the data source is failed.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY090	Invalid string or buffer length	*ValuePtr is a string, and BufferLength is smaller than 0 but it is not SQL_NTS.
HY092	Invalid attribute/option identifier	The Attribute argument value is not valid.
HYC00	Optional feature not implemented	The Attribute argument value is valid but it is not supported by the driver.

Description

If an attribute specifies the attribute which returns the string, ValuePtr should have the pointer to the string buffer. The maximum length of the returned string including null-termination character is BufferLength in bytes.

Attribute	The description of ValuePtr
SQL_ATTR_ACCESS_MODE (ODBC 1.0)	It is the SQLUINTEGER value. SQL_MODE_READ_ONLY is used as the indicator of the connection which does not request the update. This mode is used for the transaction management, the optimization, and the lock plan of the driver or data source.

Attribute	The description of ValuePtr
	The default value is SQL_MODE_READ_WRITE.
SQL_ATTR_AUTOCOMMIT (ODBC 1.0)	It is SQLUINTEGER value which specifies whether to use auto commit or manual commit. • SQL_AUTOCOMMIT_ON: It is the default value. The driver uses the auto commit mode. Each statement is immediately committed after execution. When SQL_ATTR_AUTOCOMMIT is set to SQL_AUTOCOMMIT_ON, the open transaction is committed to the connection to change from the manual commit mode to the auto commit mode. • SQL_AUTOCOMMIT_OFF: The driver uses the manual commit mode, and the application should explicitly commit or rollback with SQLEndTrans.
SQL_ATTR_CHARACTER_SET	It is the character set string of the driver.
SQL_ATTR_DATABASE_CHARACTER_SET	It is the character set string of the data source.
SQL_ATTR_DATE_FORMAT	It is the DATE format string of the driver.
SQL_ATTR_LOGIN_TIMEOUT (ODBC 1.0)	It is the waiting time (in seconds) for a login request with SQLUINTEGER value. If ValuePtr is 0, timeout is not used, and the connection attempt indefinitely wait.
SQL_ATTR_METADATA_ID (ODBC 3.0)	It is SQLUINTEGER value which determines the string argument of the catalog function. If it is SQL_TRUE, the string argument of catalog function is treated as an identifier, and it is case-insensitive. If the string is not separated by a delimiter, the driver removes all leading or trailing spaces and the string is capitalized. If the string is separated by a delimiter, the driver removes all leading or trailing spaces and the string between delimiters remains literally. If one of the arguments is set to the null pointer, the function returns SQL_ERROR and SQLSTATE HY009 (Invalid use of null pointer). If it is SQL_FALSE, the string argument of catalog function is not treated as an identifier. It may or may not include a search string pattern depending on the string argument. The default value is SQL_FALSE. SQL_ATTR_METADATA_ID can also be set in the statement level.
SQL_ATTR_TIMESTAMP_FORMAT	It is the TIMESTAMP format string of the driver.
SQL_ATTR_TIMESTAMP_WITH_TIMEZONE_FORMAT	It is the TIMESTAMP WITH TIME ZONE format string of the driver.
SQL_ATTR_TIMEZONE	It is the timezone string of the driver.
SQL_ATTR_TIME_FORMAT	It is the TIME format string of the driver.
SQL_ATTR_TIME_WITH_TI	It is the TIME WITH TIME ZONE format string of the driver.

Attribute	The description of ValuePtr
MEZONE_FORMAT	
SQL_ATTR_TXN_ISOLATIO N (ODBC 1.0)	It is 32-bit mask which sets the isolation level for the current connection.

SQLGetConnectOption

Conformance

Introduced version: ODBC 1.0
Standards compliance: It is not available.

Overview

SQLGetConnectOption function is replaced with SQLGetConnectAttr function in ODBC 3.x.
For more information, refer to **SQLGetConnectAttr**.

SQLGetCursorName

Conformance

Introduced version: ODBC 1.0
Standards compliance: ISO 92

Overview

SQLGetCursorName returns the specified statement and related cursor name.

Syntax

```
SQLRETURN SQLGetCursorName(
    SQLHSTMT        StatementHandle,
    SQLCHAR *       CursorName,
    SQLSMALLINT     BufferLength,
    SQLSMALLINT *   NameLengthPtr);
```

Arguments

StatementHandle

[Input] It is the statement handle.

CursorName

[Output] It is the buffer pointer which returns the cursor name

If CursorName is NULL, NameLengthPtr returns the returnable byte length (excluding the null-termi nation character)

BufferLength

[Input] It is the length of *CursorName.

NameLengthPtr

[Output] It is the memory pointer for the length of bytes returnable to *CursorName (excluding th e null-termination character). If the returnable byte length is equal to or bigger than it, *sCursorNa me is truncated to the length of BufferLength minus 1.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01004	String data, right truncated	*CursorName buffer is not large enough to return the entire cursor name, so the cursor name is truncated. The length of cursor name not truncated is returned to *NameLengthPtr. (The function returns SQL_SUCCESS_WITH_INFO.)
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY010	Function sequence error	After calling SQLExecute, SQLExecDirect, then SQL_NEED_DATA is returned and the function is called before sending all data-at-execution variables.
HY090	Invalid string or buffer length	The value of BufferLength argument is smaller than 0.

Description

The cursor name is used only in positioned update and positioned delete statements. (e.g. UPDATE table-name ...WHERE CURRENT OF cursor-name) If the cursor name is not set with SQLSetCursorName in the application, the driver generates the cursor name starting with SQL_CUR.

SQLGetCursorName can re-set the cursor name, if the statement is the assigned or ready state.

The cursor name which is explicitly or implicitly set is valid until the related statement is deleted by calling SQLFreeHandle whose HandleType is SQL_HANDLE_STMT.

SQLGetData

Conformance

Introduced version: ODBC 1.0
Standards compliance: ISO 92

Overview

SQLGetData retrieves a column data in the result set, and it can be called multiple times to retrieve the va riable length data.

Syntax

```
SQLRETURN SQLGetData(
    SQLHSTMT       StatementHandle,
    SQLUSMALLINT   Col_or_Param_Num,
    SQLSMALLINT    TargetType,
    SQLPOINTER     TargetValuePtr,
    SQLLEN         BufferLength,
    SQLLEN *       StrLen_or_IndPtr);
```

Arguments

StatementHandle

[Input] It is the statement handle.

Col_or_Param_Num

[Input] It is the column number for retrieving the column data from the returned data. Result set c olumns are numbered in increasing order, starting from 1.

TargetType

[Input] It is the C data type identifier of *TargetValuePtr buffer.

If TargetType is SQL_ARD_TYPE, the driver uses the type identifier specified in SQL_DESC_CONCISE _TYPE field of ARD.

TargetValuePtr

[Output] It is the buffer pointer to which the data is returned

TargetValuePtr can not be NULL.

BufferLength

[Input] It is the byte length of *TargetValuePtr buffer.

When the driver returns the variable length data such as string or binary data, it uses BufferLength to avoid writing beyond the end of *TargetValuePtr buffer. Be cautious when returning character data to *TargetValuePtr because the driver calculates the null-termination character. Therefore, *TargetValuePtr should include the space for the null-termination character. Otherwise, the driver can truncate the data.

When the driver returns the fixed length data such as integer or date structure, the driver assumes that the buffer is large enough to store the data, and it ignores BufferLength. Therefore, the application should allocate the buffer which is large enough to the fixed length data. If not, the driver can write beyond the end of buffer.

SQLGetData returns SQLSTATE HY090(Invalid string or buffer length), if BufferLength is smaller than 0.

StrLen_or_IndPtr

[Output] It is the buffer pointer which returns the length or indicator value. If the argument is the null pointer, the length and indicator values are not returned, and an error occurs when fetching NULL data.

SQLGetData can return SQL_NO_TOTAL, SQL_NULL_DATA, and the data length which can be returned to the length/indicator buffer.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01004	String data, right truncated	All data for the column specified as Col_or_Param_Num can not be retrieved by a single function call. The length of the remaining data in the specified column is returned to *StrLen_or_IndPtr before calling SQL_NO_TOTAL or SQLGetData. (The function returns SQL_SUCCESS_WITH_INFO.)
01S07	Fractional truncation	The data returned for one or more column is truncated. For numeric data types, the decimal place is truncated. For time, timestamp, interval data types which contain the period component, the decimal place of time is truncated. (The function returns SQL_SUCCESS_WITH_INFO.)
07006	Restricted data type attribute violation	The column data value in the result set can not be converted to the C data type specified by TargetType.
	Invalid descriptor	Col_or_Param_Num argument value is bigger than the number of columns i

SQLSTATE	Error	Description
07009	index	n the result set.
08S01	Communication l ink failure	Before the function processing is completed, the connection between the dr iver and the data source is failed.
22002	Indicator variable required but not supplied	StrLen_or_IndPtr is the null pointer, and the returned data is NULL.
22003	Numeric value ou t of range	The integer part (not the decimal place) of the numerical value returned fro m one or more columns is truncated.
22007	Invalid datetime f ormat	The string in the result set is not the valid date, time, timestamp format.
22012	Division by zero	The result of the arithmetic expression divided by 0 is returned.
22015	Interval field over flow	When the interval C type is specified in the exact numeric or interval SQL da ta type, the significant figures in the leading field is lost. The value of SQL type can not be expressed to C interval type.
22018	Invalid character value for cast spe cification	The character not represented as the character set of C buffer is included in the character column of the result set. The C type is the exact or approximate numeric, datetime, interval data type, and if the SQL type is the character data type, the value of the column boun d to the C type is not valid.
24000	Invalid cursor stat e	The function is called without calling SQLFetch or SQLFetchScroll. StatementHandle is executed but result set related to StatementHandle doe s not exist. The cursor is open by calling SQLFetch or SQLFetchScroll, but the cursor is p ointing to before the start or after the end of result set.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocatio n error	It is a memory allocation error.
HY003	Program type out of range	TargetType argument value is not valid.
HY009	Invalid use of null pointer	TargetValuePtr argument is the null pointer.
HY010	Function sequenc e error	The specified StatementHandle is not in executed state. The function is calle d without calling SQLExecDirect, SQLExecute, the catalog function. After calling SQLExecute, SQLExecDirect, then SQL_NEED_DATA is returned and the function is called before sending all data-at-execution variables.
HY090	Invalid string or b uffer length	BufferLength argument value is smaller than 0.

Description

SQLGetData returns the data of the specified column, and it can be called only after one or more rows ar e fetched from the result set of SQLFetch or SQLFetchScroll. When the variable length data is too large to be returned with a single call SQLGetData (because of application restrictions), SQLGetData can partially r etrieve it.

Using SQLGetData

If TargetType argument is the interval data type, the default value is interval leading precision(2), interval seconds precision(6), and it is set to SQL_DESC_DATETIME_INTERVAL_PRECISION field and SQL_DESC_P RECISION field of ARD each. If TargetType argument is the SQL_C_NUMERIC data type, the default value is precision(38), scale(0), and it is set to SQL_DESC_PRECISION field and SQL_DESC_SCALE field of ARD e ach. If the default precision and scale are not appropriate, the application should explicitly set the descrip tor field by calling SQLSetDescField or SQLSetDescRec.

Partial Retrieving of Variable Length Data

SQLGetData can partially retrieve the variable length data whose SQL data type is SQL_CHAR, SQL_VARC HAR, SQL_LONGVARCHAR, SQL_WCHAR, SQL_WVARCHAR, SQL_WLONGVARCHAR, SQL_BINARY, SQL _VARBINARY, SQL_LONGVARBINARY.

The application continuously calls SQLGetData multiple times for the same column to partially retrieve the data of the column. SQLGetData returns the next part of the data at each call. The applications should re assemble the parts, being careful to remove the null-termination character in the middle of the character data. If more data should be returned, SQLGetData returns SQL_SUCCESS_WITH_INFO, SQLSTATE 01004 (Data truncated). If the last part of data is returned, SQL_SUCCESS is returned.

SQLGetData can not be used for returning the part of fixed length data. If SQLGetData is called once or more for a column which contains the fixed-length data, SQL_NO_DATA is returned after the first call.

Retrieving Data Using SQLGetData

SQLGetData performs the following processes to return the data for the specified column.

1. If all data is returned to the column, SQL_NO_DATA is returned.

2. If the data is NULL, *StrLen_or_IndPtr is set to SQL_NULL_DATA. If the data is NULL and *StrLen_or_ IndPtr is the null pointer, SQLGetData returns SQLSTATE 22002 (Indicator variable required but not s upplied).
   If the column data is not NULL, SQLGetData performs the third process.

3. If the attribute of SQL_ATTR_MAX_LENGTH statement is set to non-zero value or the column contai ns the string or binary data or SQLGetData is not called for the column before, then the data is trunc ated to length of SQL_ATTR MAXLENGTH bytes.

The attribute of SQL_ATTR_MAX_LENGTH statement is used to reduce the network traffic. It is generally implemented by the data source, and it truncates the data before returning the data via the network. The driver and the data source are not required to support it. Therefore, the application should allocate a buffer of the proper size, and specify the size of BufferLength argument to guarantee that the data is truncated to the specific size.

4. The data is converted to the data type specified in TargetType. The default precision and scale values for the data type are given to the data. If TargetType is SQL_ARD_TYPE, the data type in SQL_DESC_CONCISE_TYPE field of ARD is used. The data provides the precision and scale of SQL_DESC_DATETIME_INTERVAL_PRECISION field, SQL_DESC_PRECISION field, SQL_DESC_SCALE field of ARD depending on the data type of SQL_DESC_CONCISE_TYPE field. If all default precisions and scales are not appropriate, the application should explicitly set the descriptor field by calling SQLSetDescField or SQLSetDescRec.

5. If the data is converted to the variable length data type such as the string or binary, SQLGetData checks whether the data length exceeds BufferLength (including the null-termination character). If the data length exceeds BufferLength, SQLGetData truncates the data to the length of BufferLength minus the null-termination character. If the binary data length exceeds the data buffer length, SQLGetData truncates it to the length of BufferLength bytes.

   If the null-termination character is not stored in the provided data buffer, SQLGetData returns SQL_SUCCESS_WITH_INFO and SQLSTATE 01004.

   SQLGetData does not truncate the data converted to the fixed length data type. In this case, the length of *TargetValuePtr is always assumed as the size of the data type.

6. It stores the converted data in *TargetValuePtr. Be cautious that SQLGetData can not return data out of line.

7. The data length is stored in *StrLen_or_IndPtr. If StrLen_or_IndPtr is the null pointer, SQLGetData does not return the length.
   - For the string and binary data, it is the length after conversion and before truncated to BufferLength. If the driver can not check the data length after conversion (it can happen when it is a long data type), it returns SQL_SUCCESS_WITH_INFO, and it sets the length to SQL_NO_TOTAL.(The last call of SQLGetData should return the value of data length which is neither 0 nor is SQL_NO_TOTAL) If the data is truncated by the attribute of SQL_ATTR_MAX_LENGTH statement, the attribute value is stored in *StrLen_or_IndPtr. It is because the data is designed to be passed on the server before the attribute is converted, and the driver can not know the actual length. When SQLGetData is continuously called multiple times for the same column, this is the data length available at the beginning of the current call. In other words, the length is reduced due to each subsequent calls.
   - For all other data types, this is the data length after conversion. In other words, it is the size of the type to which data is converted.

8. If the data is truncated without loss of the default value(for example, the real number 1.234 is truncated to an integer 1 when converted), or if it is truncated because BufferLength is too small (e.g. the

string "abcdef" is stored in the buffer of 4-byte length) during data conversion, then SQLGetData returns SQLSTATE 01004 (Data truncated) and SQL_SUCCESS_WITH_INFO. If the data is truncated without loss of the default value due to the attribute of SQL_ATTR_MAX_LENGTH statement, SQLGetData returns SQL_SUCCESS but it does not return SQLSTATE 01004 (Data truncated).

If SQLGetData does not return SQL_SUCCESS nor SQL_SUCCESS_WITH_INFO (When SQLGetData is called for the bound column) the contents of bound data buffer, and the length/indicator buffer are not defined.

If SQLGetData is continuously called, it retrieves the data from the last column requested. The previous offset is not valid.

The following is an example.

```
SQLGetData(icol=n), SQLGetData(icol=m), SQLGetData(icol=n)
```

The second call SQLGetData(icol=n) retrieves the data starting from the column n. All offsets of the data are not valid any more due to the previous SQLGetData call.

SQLGetData and Descriptor

SQLGetData does not directly interact with any descriptor field.

If TargetType is SQL_ARD_TYPE, the data type of SQL_DESC_CONCISE_TYPE field of ARD is used. If TargetType is SQL_ARD_TYPE or SQL_C_DEFAULT, the precisions and scales of SQL_DESC_DATETIME_INTERVAL_PRECISION field, SQL_DESC_PRECISION field, and SQL_DESC_SCALE field of ARD are given according to the data type of SQL_DESC_CONCISE_TYPE field.

SQLGetDescField

Conformance

Introduced version: ODBC 3.0
Standards compliance: ISO 92

Overview

SQLGetDescField returns the current setting or value of a single field of descriptor record.

Syntax

```
SQLRETURN SQLGetDescField(
    SQLHDESC        DescriptorHandle,
    SQLSMALLINT     RecNumber,
    SQLSMALLINT     FieldIdentifier,
    SQLPOINTER      ValuePtr,
    SQLINTEGER      BufferLength,
    SQLINTEGER *    StringLengthPtr);
```

Arguments

DescriptorHandle

[Input] It is the descriptor handle.

RecNumber

[Input] It is the descriptor record of information which the application tries to find. If FieldIdentifier argument is the header field, RecNumber is ignored. If RecNumber is equal to or less than SQL_DESC_COUNT, and the row does not include the data for the column or parameter, SQLGetDescField returns the default value of field.

FieldIdentifier

[Input] It is the descriptor field to which the value is returned.

ValuePtr

[Output] It is the buffer pointer which returns the descriptor information. The data type is dependent on the value of FieldIdentifier.

If ValuePtr is the integer type, the application should use SQLULEN buffer initialized to 0.

If ValuePtr is null, StringLengthPtr returns the number of the total returnable bytes (excluding null-

termination character).

BufferLength

[Input] If FieldIdentifier field is defined in ODBC and ValuePtr points to the string or binary buffer, the argument should be the length of *ValuePtr. If FieldIdentifier field is defined in ODBC and *ValuePtr is an integer, the argument is ignored.

StringLengthPtr

[Output] It is the pointer which returns the number of the total returnable bytes from *ValuePtr (excluding null-termination byte for the character data).

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_NO_DATA, SQL_INVALID_HANDLE

If RecNumber is bigger than the number of current descriptor records, SQL_NO_DATA is returned.
If DescriptorHandle is IRD handle and the statement is in the ready state or executing state but associated cursor does not exist, SQL_NO_DATA is returned.

Diagnosis

SQLSTATE	Error	Description
01000	General Warning	It is the driver-specific informational message. (The function returns SQL_SUCCESS_WITH_INFO.)
01004	String data, right truncated	The buffer length of *ValuePtr is shorter than the value length of descriptor field, so it is truncated. The remaining length of descriptor field is returned in *StringLengthPtr. (The function returns SQL_SUCCESS_WITH_INFO.)
07009	Invalid descriptor index	RecNumber argument is set to 0, and the status attribute of SQL_ATTR_USE_BOOKMARKS is set to SQL_UB_OFF, and DescriptorHandle argument is a IRD handle. FieldIdentifier argument is the record field, RecNumber argument is 0, and DescriptorHandle argument is a IPD handle. RecNumber argument is smaller than 0.
08S01	Communication link failure	Before the function processing is completed, the connection between the driver and the data source is failed.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY007	Associated statement is not prepared	DescriptorHandle is related to a IRD handle, and the related handle is not in the preparation state or executing state.
		It is the DescriptorHandle related to StatementHandle for the asynchronous performance function which is still being executed while it is called.

SQLSTATE	Error	Description
HY010	Function sequence error	It is the DescriptorHandle related to StatementHandle which SQLExecute, SQLExecDirect, SQLBulkOperations, SQLSetPos are called and SQL_NEED_DATA is returned. The asynchronously executing function is called for the connection handle related to DescriptorHandle, and it is still being executed even when SQLGetDescField is called.
HY013	Memory management error	The size of buffer used as an argument is smaller than 0, or it can not access the memory.
HY021	Inconsistent descriptor information	SQL_DESC_TYPE and SQL_DESC_DATETIME_INTERVAL_CODE field are not the valid format for ODBC SQL type, SQL type of the specific driver (for IPD) or ODBC C type (for APD or ARD).
HY090	Invalid string or buffer length	*Valueptr is the string, and BufferLength is smaller than 0.
HY091	Invalid descriptor field Identifier	FieldIdentifier is not the field defined in ODBC, and it is not the implemented value. FieldIdentifier is not defined for DescriptorHandle.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	For more information about the suspended state, refer to **SQLEndTran**.
HYT01	Connection timeout expired	The connection is expired before the data source responds to the request. The time limit can be set via SQL_ATTR_CONNECTION_TIMEOUT of SQLSetStmtAttr.
IM001	Driver does not support this function	The driver related to DescriptorHandle does not support the function.

Description

The application can call SQLGetDescField to return a single field value of the descriptor record. Calling SQLGetDescField returns the settings for all fields of all descriptor types including the header field, record fields, and bookmarks field. The application can obtain the multiple field settings in an arbitrary order in the same or another descriptor by repeatedly calling SQLGetDescField. SQLGetDescField can be called to return the driver definition descriptor field.

For the performance reason, the application should not call SQLGetDescField for IRD before executing the statement.

Multiple field settings for the size of name, data type, column or parameter data can be retrieved by calling SQLGetDescRec once. SQLGetStmtAttr can be called to return a single field settings in the descriptor he

ader which is the statement attribute. SQLColAttribute, SQLDescribeCol, SQLDescribeParam return the re cords or book mark fields.

When the application calls SQLGetDescField to retrieve the undefined field value for a particular descripto r type, the function returns SQL_SUCCESS but the returned field value is not defined. For example, calling SQLGetDescField for SQL_DESC_NAME field of APD/ ARD or calling SQLGetDescField for SQL_DESC_NUL LABLE field returns SQL_SUCCESS, but the field value is not defined.

When the application calls SQLGetDescField to retrieve the field value whose particular descriptor is defin ed but any settings such as default value are not defined, then the function returns SQL_SUCCESS but th e returned field value is not defined.

Header Field

Each descriptor consists of the following fields.

SQL_DESC_ALLOC_TYPE[All]

It is the header field which is read-only SQLSMALLINT. It specifies whether the descriptor is automa tically assigned by the driver or it is explicitly assigned by the application. The application can read t he field but can not modify it. When the descriptor is automatically allocated by the driver, the field is set to SQL_DESC_ALLOC_AUTO by the driver.

SQL_DESC_ARRAY_SIZE[Application descriptors]

It is the header field which is SQLULEN in ARD. It specifies the number of rows in the row set. This i s the number of rows returned by the operation caused by calling SQLFetch, SQLFetchScroll, SQLB ulkOperations or SQLSetPos.

It is the header field which is SQLULEN in APD. It specifies the number of values of each parameter.

The default value of the field is 1. If SQL_DESC_ARRAY_SIZE is bigger than 1, SQL_DESC_DATA_PT R, SQL_DESC_INDICATOR_PTR, SQL_DESC_OCTET_LENGTH_PTR of APD or ARD points to the arra y. The constant of each array is equal to the field value.

This field in ARD can be set by calling SQLSetStmtAttr together with SQL_ATTR_ROW_ARRAY_SIZE attribute. The field in APD can also be set by calling SQLSetStmtAttr together with SQL_ATTR_PAR AMSET_SIZE.

SQL_DESC_ARRAY_STATUS_PTR[All]

For each descriptor type, the header field which is SQLUSMALLINT* points to the array of SQLUSM ALLINT value. The arrays are named as row status array (IRD), parameter status array (IPD), row op eration array (ARD), parameter operation array (APD).

This header field in IRD points to the row status array which contains the status value after calling S QLBulkOperations, SQLFetch, SQLFetchScroll, or SQLSetPos. The application allocates SQLUSMALLI NT array, and this field should point to the array. The field is generally the NULL pointer. The driver will create an array unless SQL_DESC_ARRAY_STATUS_PTR is set to the NULL pointer.

If the application sets the elements of row status array pointed by SQL_DESC_ARRAY_STATUS_PTR field of IRD, then the operation of driver is not defined.

The array is initialized and created by calling SQLBulkOperations, SQLFetch, SQLFetchScroll or SQLSetPos. If this call does not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, then the content of array pointed by the field is not defined.

Elements in the array can contain the following values.

- SQL_ROW_SUCCESS: The row is successfully retrieved, and it is not updated since the last retrieving.
- SQL_ROW_SUCCESS_WITH_INFO: The row is successfully retrieved, and it is not updated since the last return. However, the warning for the row is returned.
- SQL_ROW_ERROR: An error occurs during retrieving the row.
- SQL_ROW_UPDATED: The row is successfully retrieved, and it is updated since the last retrieving. If the row is retrieved again, the state is SQL_ROW_SUCCESS.
- SQL_ROW_DELETED: The row is deleted since the last retrieving.
- SQL_ROW_ADDED: The row is entered by SQLBulkOperations. If the row is retrieved again, the state is SQL_ROW_SUCCESS.
- SQL_ROW_NOROW: The row set and the last of result set are overlapped. And a row is not returned corresponding to the elements of row status array.

This field of IRD can be set by calling SQLSetStmtAttr together with SQL_ATTR_ROW_STATUS_PTR attribute.

SQL_DESC_ARRAY_STATUS_PTR field of IRD is valid only after SQL_SUCCESS or SQL_SUCCESS_WITH_INFO is returned. If the return code is not one of these, the thing pointed by SQL_DESC_ROWS_PROCESSED_PTR is not defined.

This header field in IPD calls SQLExecute or SQLExecDirect and then points to the parameter status array which includes each parameter status information. If SQLExecute or SQLExecDirect is called and SQL_SUCCESS or SQL_SUCCESS_WITH_INFO is not returned, then the content of array pointed by this field is not defined. The application allocates SQLUSMALLINT array, and this field should point the array. The driver will create an array if SQL_DESC_ARRAY_STATUS_PTR field is not set to the NULL pointer.

Elements in the array can contain the following values.

- SQL_PARAM_SUCCESS: SQL statement was successfully executed for this parameter set.
- SQL_PARAM_SUCCESS_WITH_INFO: SQL statement was successfully executed for this parameter set, but there is a warning information available to the diagnostic data structure.
- SQL_PARAM_ERROR: An error occurred when processing the parameter set. Additional error information is in the diagnostic data structure.
- SQL_PARAM_UNUSED: An error occurred while processing some previous parameters, or it is not used because SQL_PARAM_IGNORE is set in the parameter set of the array pointed by SQL_DESC_ARRA

Y_STATUS_PTR field of APD.
- SQL_PARAM_DIAG_UNAVAILABLE: The diagnostic information can not be used. For example, the dri ver treats the array of the parameter as one and does not generate the level of the error information.

This field in IPD can be set by calling SQLSetStmtAttr together with SQL_ATTR_PARAM_STATUS_PTR attri bute.

In ARD, the corresponding field points to the row operation array for the values set by the application to determine whether or not to ignore the row in SQLSetPos operation.

Elements in the array can contain the following values.

- SQL_ROW_PROCEED: The row is included in the bulk operation which uses SQLSetPos. (This setting does not guarantee that the operation occurs on the row. If the row has SQL_ROW_ERROR state of t he row status array of IRD, the driver can not implement the operation for the row.)
- SQL_ROW_IGNORE: The row is excluded from the bulk operation which uses SQLSetPos.

If elements of the array is not set, all rows are included in the bulk operation. If the value of SQL_DESC_A RRAY_STATUS_PTR field of ARD is the NULL pointer, all rows are included in the bulk operation. It is tran slated as same as when the pointer points to a valid array and all elements of the array are SQL_ROW_PR OCEED. When all elements in the array are set to SQL_ROW_IGNORE, the values in the row status array f or the ignored rows are not updated.

This field in ARD can also be set by calling SQLSetStmtAttr together with SQL_ATTR_ROW_OPERATION_P TR attribute.

When SQLExecute or SQLExecDirect is called, this header field in ARD points to the parameter operation array of the values set by the application to indicate whether or not to ignore the parameter set.

Elements in the array can contain the following values.

- SQL_PARAM_PROCEED: The parameter set is included in calling SQLExecute or SQLExecDirect.
- SQL_PARAM_IGNORE: The parameter set is not included in calling SQLExecute or SQLExecDirect.

If elements of the array are not set, all parameter sets of the array are used to call SQLExecute or SQLExec Direct. If the value of SQL_DESC_ARRAY_STATUS_PTR field of APD is the NULL pointer, all parameter set are used. It is read as same as when the pointer points to the valid array, or the array whose all elements are SQL_PARAM_PROCEED.

This field in APD can also be set by calling SQLSetStmtAttr together with SQL_ATTR_PARAM_OPERATION _PTR attribute.

SQL_DESC_BIND_OFFSET_PTR[Application descriptors]

This header field is SQLLEN* and it points to the binding offset. It is set to the NULL pointer by defa ult. If this field is not the NULL pointer, the driver dereferences the pointer and each value of the d

eferred field which has non-NULL value of the descriptor record (SQL_DESC_DATA_PTR_, SQL_DES C_INDICATOR_PTR and SQL_DESC_OCTET_LENGTH_PTR) at fetching time, and uses a new pointer value when binding.

The binding offset is always directly added to SQL_DESC_DATA_PTR field, SQL_DESC_INDICATOR_ PTR field and SQL_DESC_OCTET_LENGTH_PTR field. If the offset value is changed to another value, the new value is directly and continuously added as each descriptor field value. The new offset is n ot added to the previous offset value of the field.

It is the deferred field. This field is not used at the time of setting, but it is used by the driver when the data buffer address is checked.

This field in ARD can be set by calling SQLSsetStmtAttr together with SQL_ATTR_ROW_BIND_OFFS ET_PTR attribute.

For more information, refer to **SQLFetchScroll** or **SQLBindParameter**.

SQL_DESC_BIND_TYPE[Application descriptors]

This header field is SQLUINTEGER and is used to set the binding direction.

This field in ARD specifies the binding direction when SQLFetchScroll or SQLFetch is called on the r elated statement handle.

This field can be set to SQL_BIND_BY_COLUMN(default) to select the column-wise binding for the columns.

The field in ARD can be set by calling SQLSetStmtAttr together with SQL_ATTR_ROW_BIND_TYPE attribute.

The field specifies the binding direction used in the dynamic parameter.

This field may be set to SQL_BIND_BY_COLUMN(default) to select the column-wise binding for the parameter.

This field can be set by calling SQLSetStmtAttr together with SQL_ATTR_PARAM_BIND_TYPE attrib ute.

SQL_DESC_COUNT[All]

This header field is SQLSMALLINT and specifies the 1-based index of the top-level record which incl udes the data. When the driver sets the data structure to the descriptor, SQL_DESC_COUNT should be set to display the number of important records. It is not required to specify how many space to r eserve for the record when the application allocates instances of data structure. Like as the applicat ion specifies the contents of the records, the driver performs the requested operation for the descri ptor handle to guarantee displaying the sufficient size of the data structure.

It is not the number of all data column or all parameters which SQL_DESC_COUNT is bound to, but it is the number of top-level records. If the binding of the top-level column or parameter is released, SQL_DESC_COUNT is changed to the number of next top-level columns or parameters. If the bindi ng of the top-level column, the column which is smaller than the parameter, or parameters is relea sed (if TargetValuePtr argument is set to the NULL pointer, and SQLBindCol is called, or if Paramete

rValuePtr argument is set to the NULL pointer and SQLBindParameter is called), SQL_DESC_COUNT is not changed. If the added column or parameter is bound with bigger number than the number of top level records which includes the data, the driver automatically increases the value of SQL_DESC_COUNT field. If the binding for all columns is released by calling SQLFreeStmt with the SQL_UNBIND option, SQL_DESC_COUNT in ARD and IRD is set to 0. If SQLFreeStmt is called with the SQL_RESET_PARAMS option, SQL_DESC_COUNT fields in APD and IPD is set to 0.

The value of SQL_DESC_COUNT is explicitly set by calling SQLSetDescField in the application. If the value of SQL_DESC_COUNT is explicitly decreased, the records which have the value bigger than the new value of SQL_DESC_COUNT are effectively removed. If the value of SQL_DESC_COUNT field of ARD is set to 0, all buffers are released except for the bound bookmark column. The number of records of the field Of ARD does not include the bound bookmark column. The only way to release the binding of book mark column is to set SQL_DESC_DATA_PTR to NULL pointer.

SQL_DESC_ROWS_PROCESSED_PTR[Implementation descriptors]

This header field is SQLULEN* of IRD, and points to the buffer. The buffer includes the number of rows returned after calling SQLFetch or SQLFetchScroll, and the number of rows or number of errors affected by calling SQLBulkOperations or SQLSetPos and executing the bulk operation.

This header field is SQLUINTEGER* of IPD and points to the buffer which includes the processed parameter set or the number of errors. If it is NULL pointer, the number is not returned.

SQL_DESC_ROWS_PROCESSED_PTR is valid only after calling SQLFetch or SQLFetchScroll(IRD) or after SQL_SUCCESS or SQL_SUCCESS_WITH_INFO is returned by calling SQLExecute, SQLExecDirect or SQLParamData(IPD). If the functions above do not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, then the buffer contents are not defined and the buffer value is set to 0 until SQL_NO_DATA is returned.

The field in ARD can be set by calling SQLSetStmtAttr together with SQL_ATTR_ROWS_FETCHED_PTR attribute. ARD can be set by calling SQL_ATTR_PARAMS_PROCESSED_PTR.

The buffer pointed by this field is allocated by the application. It is the deferred output buffer defined by the driver. NULL pointer is set by default.

Record Field

Each descriptor includes one or more records consisting of the fields which define the column data or dynamic parameter depending on the descriptor type. Each record is a complete specification of a single column or parameter.

SQL_DESC_AUTO_UNIQUE_VALUE[IRDs]

This record field is read-only SQLINTEGER. It is SQL_TRUE if the column is automatically increased. Otherwise, it is SQL_FALSE. The auto-increment column does not need to be read-only.

SQL_DESC_BASE_COLUMN_NAME[IRDs]

This record field is read-only SQLCHAR*. It includs the default column name of the result set column. If the default column name does not exist, the field should include an empty string.

SQL_DESC_TABLE_NAME[IRDs]

This record field is read-only SQLCHAR*. It includes the base table name of the result set column. If the base table name is not defined or used, the field should include an empty string.

SQL_DESC_CASE_SENSITIVE[Implementation descriptors]

This record field is read-only SQLINTEGER. It includes SQL_TRUE if the column or parameter is case-sensitive when sorting or comparing columns or parameters, but it includes SQL_FALSE if the colu mn or parameter is case-insensitive or non-character column.

SQL_DESC_CATALOG_NAME[IRDs]

This record field that is read-only SQLCHAR*. It includes the catalog of the base table which includ es the column. If the column is an expression or a part of a view, the return value is dependent on the driver. If the data source does not support the catalog nor does it check the catalog, the field s hould include an empty string.

SQL_DESC_CONCISE_TYPE[All]

This header field is SQLSMALLINT. It speifies the simple format for the data types which includes da tetime and interval data types.

The value of SQL_DESC_CONCISE_TYPE field, SQL_DESC_TYPE field and SQL_DESC_DATETIME_IN TERVAL_CODE field are interdependent. If time is set in one of the fields, it should be set on other fields as well. SQL_DESC_CONCISE_TYPE can be set by calling SQLBindCol, SQLBindParameter or S QLSetDescField. SQL_DESC_TYPE is set by calling SQLSetDescField or SQLSetDescRec.

If SQL_DESC_CONCISE_TYPE is set as the simple data types except for the interval or datetime data type, then SQL_DESC_TYPE field is set to the same value, and SQL_DESC_DATETIME_INTERVAL_C ODE field is set to 0.

If SQL_DESC_CONCISE_TYPE is set as the simple datetime or interval data type, then SQL_DESC_T YPE field is set to the detailed type(SQL_DATETIME or SQL_INTERVAL), and SQL_DESC_DATETIME _INTERVAL_CODE field is set to the proper subcode.

SQL_DESC_DATETIME_INTERVAL_CODE[All]

This record field is SQLSMALLINT. It includes the subcode which specifies the datetime or interval d ata type when SQL_DESC_TYPE field is SQL_DATETIME or SQL_INTERVAL. It is the same for both S QL and C support it, the code includes the data type name such as "TYPE" or "C_TYPE" for datetim e type, "CODE" replaced from "INTERVAL" or "C_INTERVAL" for interval types.

If SQL_DESC_TYPE and SQL_DESC_CONCISE_TYPE of the application descriptor are set to SQL_C_ DEFAULT and the descriptor is not related to the statement handle, then the content of SQL_DESC _DATETIME_INTERVAL_CODE is not defined.

This field can set the datetime data types listed in the following table.

Datetime type	DATETIME_INTERVAL_CODE
SQL_TYPE_DATE SQL_C_TYPE_DATE	SQL_CODE_DATE

Datetime type	DATETIME_INTERVAL_CODE
SQL_TYPE_TIME SQL_C_TYPE_TIME	SQL_CODE_TIME
SQL_TYPE_TIMESTAMP SQL_C_TYPE_TIMESTAMP	SQL_CODE_TIMESTAMP

This field can set the interval data types listed in the following table.

Interval type	DATETIME_INTERVAL_CODE
SQL_INTERVAL_DAY SQL_C_INTERVAL_DAY	SQL_CODE_DAY
SQL_INTERVAL_DAY_TO_HOUR SQL_C_INTERVAL_DAY_TO_HOUR	SQL_CODE_DAY_TO_HOUR
SQL_INTERVAL_DAY_TO_MINUTE SQL_C_INTERVAL_DAY_TO_MINUTE	SQL_CODE_DAY_TO_MINUTE
SQL_INTERVAL_DAY_TO_SECOND SQL_C_INTERVAL_DAY_TO_SECOND	SQL_CODE_DAY_TO_SECOND
SQL_INTERVAL_HOUR SQL_C_INTERVAL_HOUR	SQL_CODE_HOUR
SQL_INTERVAL_HOUR_TO_MINUTE SQL_C_INTERVAL_HOUR_TO_MINUTE	SQL_CODE_HOUR_TO_MINUTE
SQL_INTERVAL_HOUR_TO_SECOND SQL_C_INTERVAL_HOUR_TO_SECOND	SQL_CODE_HOUR_TO_SECOND
SQL_INTERVAL_MINUTE SQL_C_INTERVAL_MINUTE	SQL_CODE_MINUTE
SQL_INTERVAL_MINUTE_TO_SECOND SQL_C_INTERVAL_MINUTE_TO_SECOND	SQL_CODE_MONUTE_TO_SECOND
SQL_INTERVAL_MONTH SQL_C_INTERVAL_MONTH	SQL_CODE_MONTH
SQL_INTERVAL_SECOND SQL_C_INTERVAL_SECOND	SQL_CODE_SECOND
SQL_INTERVAL_YEAR SQL_C_INTERVAL_YEAR	SQL_CODE_YEAR
SQL_INTERVAL_YEAR_TO_MONTH SQL_C_INTERVAL_YEAR_TO_MONTH	SQL_CODE_YEAR_TO_MONTH

SQL_DESC_DATETIME_INTERVAL_PRECISION[All]

This SQLINTEGER record field includes the interval leading precision if SQL_DESC_TYPE field is SQL_INTERVAL. When SQL_DESC_DATETIME_INTERVAL_CODE field is set to the interval data type, the field sets the default interval leading precision.

SQL_DESC_DISPLAY_SIZE[IRDs]

This read-only SQLINTEGER record field contains the maximum number of characters required to s how the data from the column.

SQL_DESC_FIXED_PREC_SCALE[Implementation descriptors]

This read-only SQLSMALLINT record field sets SQL_TRUE if the column is the exact numeric column and it has non-zero scale and the fixed precision. Otherwise, it sets SQL_FALSE.

SQL_DESC_INDICATOR_PTR[Application descriptors]

This SQLLEN* record field in ARD represents the indicator variable. The variable includes SQL_NULL _DATE if the column value is NULL. For APD, the indicator variable is set to SQL_NULL_DATA to sp ecify a dynamic factor as NULL, otherwise the variable is zero.

If SQL_DESC_INDICATOR_PTR field of ARD is NULL pointer, the driver prevents the information ret urned for whether the column is NULL. If the column is NULL and SQL_DESC_INDICATOR_PTR is N ULL pointer, SQLSTATE 22002(Indicator variable required but not supplied) is returned when the d river creates the buffer after calling SQLFetch or SQLFetchScroll. If calling SQLFetch or SQLFetchScr oll does not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, the buffer content is not defined.

SQL_DESC_INDICATOR_PTR field determines whether to set the field represented by SQL_DESC_O CTET_LENGTH_PTR. If the column value is NULL, the driver sets the indicator variable to SQL_NULL _DATA. The field represented by SQL_DESC_OCTET_LENGTH_PTR is not set at that moment. If NU LL is not returned while retrieving the data, the buffer represented by SQL_DESC_INDICATOR_PTR is set to 0, and the buffer represented by SQL_DESC_OCTET_LENGTH_PTR sets the data length.

If SQL_DESC_INDICATOR_PTR field of APD is NULL pointer, the application can not use the descrip tor record to specify the argument as NULL.

This field is the deferred field. It is not used when the driver is set, but it is used when the driver dis plays NULL possibility (ARD) or determines the NULL possibility (APD).

SQL_DESC_LABEL[IRDs]

This read-only SQLCHAR* record field includes the label or mark of a column. If the column does n ot have label, then the variable contains the column name. If the column is not named or it can not use the label, then the variable includes an empty string.

SQL_DESC_LENGTH[All]

This SQLULEN record field is the maximum length or actual length of the string or binary data in by tes. It is the maximum length of fixed length data type data or the actual length of variable length data type data. The value always excludes NULL termination character at the end of string. If its dat a type is SQL_TYPE_DATE, SQL_TYPE_TIME, SQL_TYPE_TIMESTAMP or SQL interval data type, the field has the string length of when datetime or interval value is rewritten to string.

The field value is different from the value of "length" defined in ODBC 2.x.

SQL_DESC_LITERAL_PREFIX[IRDs]

This read-only SQLCHAR* record field includes the characters recognized as a suffix by the charact er or the driver. The variable contains the empty string for the data type whose character suffix can

not be converted.

SQL_DESC_LOCAL_TYPE_NAME[Implementation descriptors]

This read-only SQLCHAR* record field includes the localized name in the data type, and it could be different from the regular name in the data type. If the localized name does not exist, an empty stri ng is returned. The field is only for the purpose of displaying.

SQL_DESC_NAME[Implementation descriptor]

This SQLCHAR* record field includes the field alias in the row descriptor. If the column alias is not a pplied, the column name is returned. In some cases, the driver sets SQL DESC_UNNAMED field to S QL_NAMED when setting SQL_DESC_NAME field. If neither column name nor column alias existm the driver returns the empty string of SQL_DESC_NAME field, and sets SQL_DESC_UNNAME field t o SQL_UNNAMED.

The application can set SQL_DESC_NAME field of IPD for an alias to specify the parameter name or stored named procedure parameter. SQL_DESC_NAME field of IRD is the read-only field. If the app lication tries to set this field, SQLSTATE HY091 (invalid descriptor field identifier) will be returned.

For IPD, if the driver does not support the named parameter, the field is not defined. If the driver s upports the named parameter and specifies the parameter, the name of the parameter is returned to this field.

SQL_DESC_NULLABLE[Implementation descriptors]

For IRD, this read-only SQLSMALLINT record field is SQL_NULLABLE if the column have NULL, other wise it is SQL_NO_NULLS. If it is unknown whether the column allows NULL, it is SQL_NULLABLE_U NKNOWN. The field exists particularly for the column in the result set.

For IPD, The field is always set to SQL_NULLABLE because the dynamic parameter can always be N ULL and it can not set by the application.

SQL_DESC_NUM_PREC_RADIX[All]

This SQLINTEGER field has the value of 2 when SQL_DESC_TYPE field is the approximate numeric data type. It is because SQL_DESC_PRECISION field includes the number of bits. The field has the v alue of 10 when SQL_DESC_TYPE field is the exact numeric data type and SQL_DESC_PRECISION c ontains the number of decimal places. The field sets 0 for a non-numeric data types.

SQL_DESC_OCTET_LENGTH[All]

This SQLLEN record field includes the length in bytes of a string or binary data type. For the fixed le ngth character or binary data, it is the actual length in bytes. For the variable length character or bi nary data, it is the maximum length in bytes. The value does not include the empty string of NULL t ermination character for the implement descriptor, and it includes the empty string of NULL termin ation character for the application descriptor. For the application data, the field contains the buffer size. For APD, the field is defined only for the output or input/output parameters.

SQL_DESC_OCTET_LENGTH_PTR[Application descriptors]

The SQLLEN* record field points to the variable including the total length in bytes of the dynamic a rgument(parameter descriptor) or the binding column values (row descriptor).

This value is ignored for all arguments excluding the string or binary data in APD. If the field is SQL_NTS, the dynamic argument should be terminated with NULL. The application sets the variable including the macro result of SQL_DATA_AT_EXEC or SQL_LEN_DATA_AT_EXEC in this record field of APD to indicate that the bound parameter is to be a data parameter at the run-time. If one or more fields exist, SQL_DESC_DATA_PTR can be set as the value which identifies the parameter required by the application, and which is helpful to determine the parameter.

If OCTET_LENGTH_PTR field of ARD is NULL pointer, the driver does not return the column length. If SQL_DESC_OCTET_LENGTH_PTR of APD is NULL pointer, the driver assumes that the string and binary values are terminated by NULL. (Binary values should not be terminated by NULL, but the length should be given to avoid data interruption.)

If SQLFetch or SQLFetchScroll which fills the buffer pointed by the field does not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, the buffer content is not defined. It is the deferred field. The field is not immediately used, but it is used later when the driver displays or determines the data length in octet.

SQL_DESC_PARAMETER_TYPE[IRDs]

This SQLSMALLINT record field sets the input parameter to SQL_PARAM_INPUT, the input/output parameter to SQL_PARAM_INPUT_OUTPUT, the output parameter to SQL_PARAM_OUTPUT, the stream input/output parameter to SQL_PARAM_INPUT_OUTPUT_STREAM or the stream output parameter to SQL_PARAM_OUTPUT_STREAM. It is set to SQL_PARAM_INPUT by default.

SQL_DESC_PRECISION[All]

This SQLSMALLINT record field includes the number of valid integers for the exact numeric data types, and it includes the number of bits in mantissa (the binary precision) for the approximate numeric data types. Or, it contains the number of valid integer of the fractional seconds parts of the SQL_TYPE_TIME, SQL_TYPE_TIMESTAMP, or SQL_INTERVAL_SECOND data type. The field is not defined for all other data types.

The field value is different from the "precision" value defined in ODBC 2.x.

SQL_DESC_ROWVER[Implementation descriptors]

This SQLSMALLINT record field indicates whether the column is automatically updated by DBMS when the row is updated(for example, "timestamp" in SQL Server). The record field value is set to SQL_TRUE for the row versioning column, otherwise it is set to SQL_FALSE. The column attribute is similar to setting and calling SQL_ROWVER in IdentifierType argument of SQLSpecialColumn to determine whether to automatically update column.

SQL_DESC_SCALE[All]

This SQLSMALLINT record field includes the number of decimal places defined in the decimal and numeric data types. The field is not defined for all other data types.

The field value is different from the "scale" value defined in ODBC 2.x.

SQL_DESC_SCHEMA_NAME[IRDs]

This read-only SQLCHAR* record field includes the schema name of the base table that includes the column. If the column is an expression or a part of the view, the return value is dependent on the

driver. If the data source does not support the schema or can not identify the schema name, then t his variable contains an empty string.

SQL_DESC_SEARCHABLE[IRDs]

This read-only SQLSMAINT record field sets one of the following values.

• If the column can not be used in WHERE clause, it is SQL_PRED_NONE. (It is as same as SQL_UN SEARCHABLE in ODBC 2.X.)

• If the column can only be used with LIKE predicate in WHERE clause, it is SQL_PRED_CHAR.

• If the column can be used with all comparison operators except for LIKE in WHERE clause, it is S QL_PRED_BASIC. (It is as same as the value of SQL_EXCEPT_LIKE in ODBC 2.x.)

• If the column can be used with any comparison operator in WHERE clause, it is SQL_PRED_SEAR CHABLE.

SQL_DESC_TABLE_NAME[IRDs]

This read-only SQLCHAR* record field includes the base table name which includes column. The re turn value is dependent on the driver if the column is an expression or a part of the view.

SQL_DESC_TYPE[All]

This SQLSMALLINT record field includes an abbreviated SQL data types or C data types for all data t ypes except for the interval and datetime data types. The field specifies SQL_DATETIME or SQL_INT ERVAL for datetime and interval data types.

Whenever this field includes SQL_DATETIME or SQL_INTERVAL, SQL_DESC_DATETIME_INTERVAL_ CODE field should include the appropriate subcode in the implied format. For datetime data types, SQL_DESC_TYPE includes SQL_DATETIME, and SQL_DESC_DATETIME_INTERVAL_CODE field inclu des the subcode that specifies the datetime data type. For interval data types, SQL_DESC_TYPE incl udes SQL_INTERVAL, and SQL_DESC_DATETIME_INTERVAL_CODE field includes the subcode that specifies the interval data type.

The values of SQL_DESC_TYPE and SQL_DESC_CONCISE_TYPE field are interdependent. When on e of the fields is set, the other field should also be set. SQL_DESC_TYPE can be set by calling SQLSe tDescField or SQLSetDescRec. SQL_DESC_CONCISE_TYPE can be set by calling SQLBindCol, SQLBin dParameter or SQLSetDescField.

If SQL_DESC_TYPE is set to the implied data type other than the interval or datetime data type, SQ L_DESC_CONCISE_TYPE field is set to the same value and SQL_DESC_DATETIME_INTERVAL_CODE is set to 0.

If SQL_DESC_TYPE is set to the long data type of datetime or interval data type (SQL_DATETIME or SQL_INTERVAL), SQL_DESC_DATETIME_INTERVAL_CODE field is set to the approprieate subcode, SQL_DESC_CONCISE_TYPE may be set to a value corresponding to the implied data type. If SQL_D ESC_TYPE is set to one of the implied datetime or interval data type, SQLSTATE HY021 (Inconsiste nt descriptor information) is returned.

When SQL_DESC_TYPE is set by calling SQLBindCol, SQLBindParameter or SQLSetDescField, the ne xt fields are set to the default values in the following table. The values of remaining fields in the sa

me record are not defined.

SQL_DESC_TYPE value	Setting other fields implicitly
SQL_CHAR, SQL_VARCHAR, SQL_C_CHAR, SQL_C_VARCHAR	SQL_DESC_LENGTH is set to 1. SQL_DESC_PRECISION is set to 0.
SQL_DATETIME	If SQL_DESC_DATETIME_INTERVAL_CODE is set to SQL_CODE_DATE or SQL_CODE_TIME, SQL_DESC_PRECISION is set to 0. If SQL_DESC_TIMESTAMP is set, SQL_DESC_PRECISION is set to 6.
SQL_DECIMAL, SQL_NUMERIC, SQL_C_NUMERIC	SQL_DESC_SCALE is set to 0. SQL_DESC_PRECISION is set to the implemented precision of each data type.
SQL_FLOAT, SQL_C_FLOAT	SQL_DESC_PRECISION is set to the default precision implemented in SQL_FLOAT.
SQL_INTERVAL	When SQL_DESC_DATETIME_INTERVAL_CODE is set to the interval data type, SQL_DESC_DATETIME_INTERVAL_PRECISION is set to 2 (default interval leading precision). When the interval has the part of seconds, SQL DESC PRECISION is set to 6 (the default interval seconds precision).

When the application sets the descriptor field by calling not SQLSetDescRec but SQLSetDescField, the application should define the data type firstly. Therefore other fields in the previous table are implicitly set. If it is not allowed to implicitly set any value, the application explicitly sets the value by calling SQLSetDescField or SQLSetDescRec.

SQL_DESC_TYPE_NAME[Implementation descriptors]

This read-only SQLCHAR* record field includes the data source dependent type name ("CHAR", "VARCHAR", etc.). If the data type name is unknown, the variable includes an empty string.

SQL_DESC_UNNAMED[Implementation descriptors]

If SQL_DESC_NAME field is set, the SQLSMALLINT record field in the row descriptor is set to one of SQL_NAME or SQL_UNNAMED by the driver. If SQL_DESC_NAME field does not include the column alias or if the column alias does not apply, the driver sets SQL_DESC_UNNAMED field to SQL_NAMED. If the application sets SQL_DESC_NAME field of IPD to the parameter name or alias, the driver sets SQL_DESC_UNNAMED field of IPD to SQL_NAMED. If neither column name nor does alias exist, the driver sets SQL_DESC_UNNAMED field of IPD to SQL_UNNAMED.

The application sets SQL_DESC_UNNAMED field of IPD to SQL_UNNAMED. If the application tries to set SQL_DESC_UNANMED field of IPD to SQL_NAMED, the driver returns SQLSTATE HY091 (Invalid descriptor field identifier). SQLSTATE HY091 (Invalid descriptor field identifier) is returned if the read-only application tries to set SQL_DESC_UNNAMED field of IRD.

SQL_DESC_UNSIGNED[Implemetation descriptors]

This read-only SQLSMALLINT record field is set to SQL_TRUE, if the column type is the unsigned or non-numeric data type. It is set to SQL_FALSE, if the column type is the signed data type.

SQL_DESC_UPDATABLE[IRDs]

The read-only SQLSMALLINT record field is set to one of the following values.

• If the result set column is read-only, it is SQL_ATTR_READ_ONLY.

• If the result set column is read and write, it is SQL_ATTR_WRITE.

• If the updatability for the result set column is unknown, it is SQL_ATTR_READWRITE_UNKNOWN.

SQL_DESC_UPDATABLE explains the updatability of the result set column (the column is not in the base table). The updatabilty of the column in the primary table which is a base of the result set columns may be different from the value in this field. The updatability is based on the data type, the user privileges and the result set definition. If it is not sure that the column can be updated, SQL_ATTR_READWITE_UNKNOWN should be returned.

SQL_DESC_CHAR_LENGTH_UNITS[Alls]

This SQLSMALLINT record field indicates the length units of the column whose SQL types are SQL_CHAR, SQL_VARCHAR and SQL_LONGVARCHAR.

• SQL_CLU_CHARACTERS: The length unit is CHARACTER. For example, the length of data "문자열" is 3 if the encoding method is UHC(Unified Hangul Code).

• SQL_CLU_OCTETS: The length unit is OCTETS. For example, the length of data "문자열" is 6 if the encoding method is UHC(Unified Hangul Code).

• SQL_CLU_NONE: The length unit is undefined. It is the value which is returned for the SQL types excluding the SQL types listed above.

SQLGetDescRec

Conformance

Introduced version: ODBC 1.0
Standards compliance: ISO 92

Overview

SQLGetDescRec returns the current value or setting for the multiple fields of the descriptor record. The returned field describes the name, the data type, the column size or the argument data.

Syntax

```
SQLRETURN SQLGetDescRec(
    SQLHDESC        DescriptorHandle,
    SQLSMALLINT     RecNumber,
    SQLCHAR *       Name,
    SQLSMALLINT     BufferLength,
    SQLSMALLINT *   StringLengthPtr,
    SQLSMALLINT *   TypePtr,
    SQLSMALLINT *   SubTypePtr,
    SQLLEN *        LengthPtr,
    SQLSMALLINT *   PrecisionPtr,
    SQLSMALLINT *   ScalePtr,
    SQLSMALLINT *   NullablePtr);
```

Arguments

DescriptorHandle

[Input] It is the descriptor handle.

RecNumber

[Input] It is the descriptor record of the information which the application wants to retrieve. The record number 0 is set as the bookmark record and the descriptor record starts from 1. If RecNumber is equal to or smaller than SQL_DESC_COUNT, but the data for the column or parameter is not included in the row, SQLGetDescField returns the default value of the field.

Name

[Output] It is the buffer pointer which returns SQL_DESC_NAME for the descriptor record. If Name is NULL, StringLengthPtr returns the buffer length of Name (including null-termination character).

BufferLength

[Input] It is the length of name buffer.

StringLengthPtr

[Output] It is the pointer which returns the number of characters returnable in *Name buffer excluding null-termination character. If the number of characters are equal to or bigger than BufferLength, the data of *Name is truncated to BufferLength minus the length of a null-termination character, and is null-terminated.

TypePtr

[Output] It is the buffer pointer which returns the value of SQL_DESC_TYPE for the descriptor field.

SubTypePtr

[Output] It is the buffer pointer which returns the value of SQL_DESC_DATETIME_INTERVAL_CODE field for the record of SQL_DATETIME or SQL_INTERAVL type.

LengthPtr

[Output] It is the buffer pointer which returns the value of SQL_DESC_OCTET_LENGTH field for the descriptor field.

PrecisionPtr

[Output] It is the pointer which returns the value of SQL_DESC_PRECISION field for the descriptor record.

ScalePtr

[Output] It is the pointer which returns the value of SQL_DESC_SCALE field for the descriptor record.

NullablePtr

[Output] It is the pointer which returns the value of SQL_DESC_NULLABLE field for the descriptor record.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_NO_DATA, SQL_INVALID_HANDLE

If RecNumber is bigger than the number of current descriptor records, SQL_NO_DATA is returned. If DescriptorHandle is the IRD handle and the statement is on the preparation or execution state but related cursor does not exist, then SQL_NO_DATA is returned.

Diagnosis

SQLSTATE	Error	Description
01000	General Warning	It is a specific information message of the driver. (The function returns SQL_SUCCESS_WITH_INFO.)
01004	String data, right truncated	The length of *ValuePtr buffer is shorter than the length of the descriptor field value, so the length is truncated. The remaining length of the descriptor field is returned to *StringLengthPtr. (The function returns SQL_SUCCESS_WITH_INFO.)
07009	Invalid descriptor index	The fieldIdentifier argument is the record field, and the value of RecNumber argument is set to 0, and DescriptorHandle argument is the IPD handle. RecNumber argument is set to 0, and SQL_ATTR_USE_BOOKMARKS attribute is set to SQL_UB_OFF, and DescriptorHandle argument is the IRD handle. RecNumber argument is smaller than 0.
08S01	Communication link failure	Before the function processing is completed, the connection between the driver and the data source is failed.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY007	Associated statement is not prepared	DescriptorHandle is related to the IRD handle, and the related statement handle is not on the preparation or execution state.
HY010	Function sequence error	StatementHandle related to DescriptorHandle asynchronously executes the function and it is still executung the function when called. After calling SQLExecute, SQLExecDirect, then SQL_NEED_DATA is returned and the function is called before sending all data-at-execution variables. The function for the connetion handle related to DescriptorHandle is called, and it is still executing when SQLGetDescRec is called.
HY013	Memory management error	The buffer size used as the argument is smaller than 0, or it can not access the memory.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	For more information about the suspended state, refer to **SQLEndTran** function.
HYT01	Connection timeout expired	The connection is expired before the data source responds to the request. The timeout can be set via SQL_ATTR_CONNECTION_TIMEOUT of SQLSetStmtAttr.
IM001	Driver does not support this function	The driver related to DescriptorHandle does not support the function.

Description

The application can call SQLGetDescRec to retrieve the following descriptor field value for a column or pa rameter.

- SQL_DESC_NAME
- SQL_DESC_TYPE
- SQL_DESC_DATETIME_INTERVAL_CODE
- SQL_DESC_OCTET_LENGTH
- SQL_DESC_PRECISION
- SQL_DESC_SCALE
- SQL_DESC_NULLABLE

SQLGetDescRec does not retrieve the header field value.

The application can set the argument corresponding to the null pointer field so that it prevents the return for the field setting.

When the application retrieves the value of the undefined field for the specific descriptor type by calling S QLGetDescRec, the function returns SQL_SUCCESS, but the return value for the field is not defined. For e xample, if SQLGetDescRec is called for SQL_DESC_NAME field or SQL_DESC_NULLABLE field of APD or A RD, then SQL_SUCCESS is returned but the field value is not defined.

When the application calls SQLGetDescRec to retrieve the value of the field which is defined as the specifi c descriptor type but is not set as the default, the function returns SQL_SUCCESS, but the field value is no t defined. For more information, refer to **Initializing Descriptor Field** of SQLSetDescField.

Each field value can be retrieved separately by calling SQLGetDescField. For more information about the d escriptor header or the fields in the record, refer to **SQLSetDescField**.

SQLGetDiagField

Conformance

Introduced version: ODBC 3.0
Standards compliance: ISO 92

Overview

SQLGetDiagField returns the current value of the record fields in the diagnostic data structure which cont ains an error, warning, and status information.

Syntax

```
SQLRETURN SQLGetDiagField(
    SQLSMALLINT     HandleType,
    SQLHANDLE       Handle,
    SQLSMALLINT     RecNumber,
    SQLSMALLINT     DiagIdentifier,
    SQLPOINTER      DiagInfoPtr,
    SQLSMALLINT     BufferLength,
    SQLSMALLINT *   StringLengthPtr);
```

Arguments

HandleType

[Input] It is the handle type identifier which requires diagnostics. The identifier should be one of th e followings.

- SQL_HANDLE_DBC
- SQL_HANDLE_DESC
- SQL_HANDLE_ENV
- SQL_HANDLE_STMT

Handle

[Input] It is the handle for the diagnostic data structure, of the type pointed by HandleType. If Han dleType is SQL_HANDLE_ENV, the Handle can be the shared or non-shared environment handle.

RecNumber

[Input] It indicates the state record of the information found by the application. The status record s tarts from 1. If DiagIdentifier points any field of the diagnostic header, RecNumber is ignored. Othe rwise, RecNumber should be bigger than 0.

DiagIdentifier

[Input] It refers to the diagnostic field whose value is returned. For more information, refer to **DiagI dentifier Argument** in the Description section.

DiagInfoPtr

[Output] It is the buffer pointer to which return the diagnostic information. The data type depnend s on the value of DiagIdentifier. If DiagInfoPtr is an integer type, the application should use the SQL ULEN buffer and initialize it to 0 before calling the function because some drivers record only the lo wer 32-bits or 16-bits of the buffer and leave the upper bits intact. If DiagInfoPtr is NULL, StringLen gthPtr returns the total number of returnable bytes excluding null-termination character in the buff er pointed by DiagInfoPtr.

BufferLength

[Input] If DiagIdentifier is the ODBC defined diagnosis, and DiagInfoPtr points to the string or binar y buffer, the argument should be the length of DiagInfoPtr. If DiagIdentifier is the ODBC defined di agnosis field, and DiagInfoPtr is a number, BufferLength is ignored. If the value of DiagInfoPtr is th e Unicode string (When calling SQLGetDiagFieldW), BufferLength should be an even number. If Di agIdentifier is a driver defined field, the application refers to the field characteristic to the driver ma nager by setting BufferLength argument. BufferLength can have the following values.

- If DiagInfoPtr is a string buffer pointer, BufferLength is the string length or SQL_NTS.
- If DiagInfoPtr is a binary buffer pointer, the application stores the macro result of SQL_LENG_BI NARY_ATTR(length) in BufferLength. A negative number is stored in BufferLength.
- If DiagInfoPtr is not a string or binary buffer pointer, BufferLength should have the value of SQL _IS_POINTER.
- If DiagInfoPtr is the fixed length data type, BufferLength is one of SQL_IS_INTEGER, SQL_IS_UIN TEGER, SQL_IS_SMALLINT, or SQL_IS_USMALLINT.

StringLengthPtr

[Output] It is the buffer pointer which returns the total length of bytes excluding the null-terminati on character which is returned to *DiagInfoPtr for the character data. If the returnable length in by tes is equal to or bigger than BufferLength, the text in *DiagInfoPtr is truncated to the length of Bu fferLength minus null-termination character.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE, SQL_NO_DATA

Diagnosis

SQLGetDiagField does not post the diagnostic record. The result of the return value is used as follows.

- SQL_SUCCESS: The function successfully returns the diagnosis information.
- SQL_SUCCESS_WITH_INFO: The data in the diagnostic field is truncated because *DiagInfoPtr is too s mall to store the requested diagnosis field. The application should compare BufferLength to the actu al byte number stored in *StringLengthPtr to check whether the truncation occurred.
- SQL_INVALID_HANDLE: The handle is not valid for a type indicating HandleType.
- SQL_ERROR: One of the followings occurs.
  - DiagIdentifier argument does not have the valid value.
  - DiagIdentifier argument is one of SQL_DIAG_CURSOR_ROW_COUNT, SQL_DIAG_DYNAMIC_FU NCTION, SQL_DIAG_DYNAMIC_FUNCTION_CODE, SQL_DIAG_ROW_COUNT , and Handle is not the statement handle.
  - When DiagIdentifier refers to the diagnosis record field, RecNumber is a negative number or 0. R ecNumber is ignored for the header field.
  - The requested value is a string, and BufferLength is smaller than 0.
  - If an asynchronous notification is used, the asynchronous execution of handle is not completed.
- SQL_NO_DATA: RecNumber is bigger than the number of diagnosis records for the handle specified i n Handle. If the diagnosis record for handle does not exist, then the function returns SQL_NO_DATA for RecNumber of any positive number.

Description

The application generally calls SQLGetDiagField for one of the three purposes.

- To obtain specific error or warning information when a function returns SQL_ERROR or SQL_SUCCES S_WITH_INFO (SQLBrowseConnect function returns SQL_NEED_DATA.)
- To obtain the number of rows in the data source when insert, delete, or update operations were perf ormed with a call to SQLExecute, SQLExecDirect, SQLBulkOperations or SQLSetPos. Or, to obtain the number of rows which exist in the current open cursor, if the driver can provide this information.
- To obtain the information of which function is executed by a calling SQLExecDirect or SQLExecute

Whenever the function is called, all ODBC functions can post 0 or more diagnosis records. Therefore, the application can call SQLGetDiagField after calling all functions. There is no limitation on the number of di agnosis records which can be stored at a time. SQLGetDiagField retrieves the diagnosis data structure spe cified in Handle and the most recent related diagnosis information. If the application calls another ODBC function instead of SQLGetDiagField or SQLGetDiagRec, diagnostic information obtained by the previous call of the same handle can be lost.

As long as SQLGetDiagField returns SQL_SUCCESS, the application can read the diagnosis record by incre asing RecNumber. The number of state records is displayed on the header field of SQL_DIAG_NUMBER. C alling SQLGetDiagField does not affect the header and record field. As long as a function except for the d

iagnostic function is not called, the application can retrieve a record field by calling SQLGetDiagField agai n using the same handle.

The application can call SQLGetDiagField anytime to return any diagnostic fields. If Handle is not the state ment handle, SQL_DIAG_CURSOR_ROW_COUNT or SQL_DIAG_ROW_COUNT is excluded because it retu rns SQL_ERROR. If any diagnosis field is not defined, calling SQLGetDiagField returns SQL_SUCCESS and t he undefined value.

API call other than the function executed asynchronously causes HY010 (Function sequence error). Howe ver, the error code is not retrieved before the asynchronous processing is completed.

HandleType Argument

Each handle type has the related diagnosis information, and HandleType argument refers to the handle ty pe.

Some headers and record fields are not returned for the environment, connection, statement, descriptor handles. The handles which are not applicable are described in the following Header Fields and Record Fi elds tables.

If HandleType is SQL_HANDLE_ENV, the handle can be the shared or non-shared environment handle.
A specific header diagnosis field of the driver is not related to the environment handle.
The diagnosis header defined for the descriptor handle is only SQL_DIAG_NUMBER and SQL_DIAG_RETU RNCODE.

DiagIdentifier Argument

The argument is the field identifier required in the diagnosis data structure. If RecNumber is equal to or bi gger than 1, the data in the field is the diagnostic information returned by the function. If RecNumber is 0, the field is in the header of the diagnostic data structure, and it has the data related to the function cal l returning the diagnostic information, not the specific information.

The driver may define a driver-specific header and record fields in the diagnosis data structure.

ODBC 3.x application which uses ODBC 2.x driver can call SQLGetDiagField when DiagIdentifier argumen t is SQL_DIAG_CLASS_ORIGIN, SQL_DIAG_CLASS_SUBCLASS_ORIGIN, SQL_DIAG_CONNECTION_NAME, SQL_DIAG_MESSAGE_TEXT, SQL_DIAG_NATIVE, SQL_DIAG_NUMBER, SQL_DIAG_RETURNCODE, SQL_ DIAG_SERVER_NAME, SQL_DIAG_SQLSTATE, and it returns SQL_ERROR for other diagnostic fields.

Header Field

DiagIdentifier	Return type	Returns
		This field contains the number of rows in the cursor. Its meaning depends o n the information type of SQLGetInfo. The information type is SQL_DYNAMI

DiagIdentifier	Return type	Returns
SQL_DIAG_CURSOR_ROW_COUNT	SQLLEN	C_CURSOR_ATTRIBUTES2, SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2, SQL_KEYSET_CURSOR_ATTRIBUTES2, SQL_STATIC_CURSOR_ATTRIBUTES2, and it refers to the number of rows which can be used in each cursor type. (It is in SQL_CA2_CRC_EXACT and SQL_CA2_CRC_APPROXIMATE bits.) This field content is defined after calling the statement handle, SQLExecute, SQLExecDirect, or SQLMoreResults. SQLGetDiagField returns SQL_ERROR if DiagIdentifier is the statement handle except for SQL_DIAG_CURSOR_ROW_COUNT.
SQL_DIAG_DYNAMIC_FUNCTION	SQLCHAR *	It is a string which describes the SQL statements executing the basic function. The field content is defined after calling SQLExecute, SQLExecDirect, or SQLMoreResults. Calling SQLGetDiagField returns SQL_ERROR if DiagIdentifier is the statement handle except for SQL_DIAG_DYNAMIC_FUNCTION.
SQL_DIAG_DYNAMIC_FUNCTION_CODE	SQLINTEGER	It is the numeric code which describes the SQL statement executing the basic function. The field content is defined after calling SQLExecute, SQLExecDirect, or SQLMoreResults. Calling SQLGetDiagField returns SQL_ERROR if DiagIdentifier is the statement handle except for SQL_DIAG_DYNAMIC_FUNCTION_CODE.
SQL_DIAG_NUMBER	SQLINTEGER	It is the number of records which are in the usable state of the specified handles.
SQL_DIAG_RETURNCODE	SQLRETURN	It is the code returned by the function. The driver does not need to implement SQL_DIAG_RETURNCODE, and it is implemented by the driver manager. If the handle is not called by any function, SQL_SUCCESS is returned for SQL_DIAG_RETURNCODE.
SQL_DIAG_ROW_COUNT	SQLLEN	It is the number of rows which are affected by the INSERT, DELETE, UPDATE statement executed by SQLExecute, SQLExecDirect, SQLBulkOperations, or SQLSetPos. It is defined by the driver after cursor specification is executed. The field content is defined only for the statement handle. Calling SQLGetDiagField returns SQL_ERROR if DiagIdentifier is the statement handle except for SQL_DIAG_ROW_COUNT. The field data is also returned to RowCountPtr of SQLRowCount. The row count returned to SQLRowCount among data of this field remains the same until statement is set back to the prepared or allocated state, but it is set again after calling the non-diagnostic function.

Record Field

DiagIdentifier	Return type	Returns
SQL_DIAG_CLASS_ORIGIN	SQLCHAR *	It is a string representing the document which defines the class of SQLSTATE value in this record. This value is ISO 9075 for all SQLSTATE defined by the Open Group and the ISO call level interface. This value is ODBC 3.0 for the s

DiagIdentifier	Return type	Returns
		pecified ODBC SQLSTATE (for All SQLSTATE classes having IM).
SQL_DIAG_COLUMN_NUMBER	SQLINTEGER	If SQL_DIAG_ROW_NUMBER is the valid row number in the row set or parameter set, this field is a value representing the column number of result set or parameter number of parameter sets. The result set column number always starts from 1. If the state record is related to the bookmark column, the field can be 0. The parameter number starts from 1. If the state record is not related to the column number or to the parameter number, the value can be SQL_NO_COLUMN_NUMBER. If the driver can not determine the column number or parameter number related to the record, the field value is SQL_COLUMN_NUMBER_UNKNOWN. The field content is defined only for the statement handle.
SQL_DIAG_CONNECTION_NAME	SQLCHAR *	It is the string representing the connection name related to the diagnostic record. The field is the driver definition. The field is a zero-length string for the diagnosis which is not related to any server and the environment handle related to the diagnostic data structure.
SQL_DIAG_MESSAGE_TEXT	SQLCHAR *	It is an information message about the error or warning.
SQL_DIAG_NATIVE	SQLINTEGER	It is the driver/ data source-specific native error code. If native error code does not exist, the driver returns 0.
SQL_DIAG_ROW_NUMBER	SQLLEN	This field contains the row number in the row set or the parameter number in the parameter set which is related to the state record. The row number and parameter number start from 1. If the state record is not related to the row number or parameter number, the field value is SQL_NO_ROW_NUMBER. If the driver can not determine the row number or parameter number related to the record, the field value is SQL_ROW_NUMBER_UNKNOWN. The field content is defined only for the statement handle.
SQL_DIAG_SERVER_NAME	SQLCHAR *	It is the string representing the server name related to the diagnostic record. It is as same as the value returned by calling SQLGetInfo with SQL_DATA_SOURCE_NAME option. The field is a zero-length string for the diagnosis which is not related to any server and the environment handle related to the diagnostic data structure.
SQL_DIAG_SQLSTATE	SQLCHAR *	It is SQLSTATE diagnostic code of the five characters.
SQL_DIAG_SUBCLASS_ORIGIN	SQLCHAR *	It is the string of the same format and valid value with SQL_DIAG_CLASS_ORIGIN which is defines the subclass part of the SQLSTATE code. The specific ODBC SQLSTATE codes in ODBC 3.0 are as follows. 01S00, 01S01, 01S02, 01S06, 01S07, 07S01, 08S01, 21S01, 21S02, 25S01, 25S02, 25S03, 42S01, 42S02, 42S11, 42S12, 42S21, 42S22, HY095, HY097, HY098, HY099, HY100, HY101, HY105, HY107, HY109, HY110, HY111,

DiagIdentifier	Return type	Returns
		HYT00, HYT01, IM001, IM002, IM003, IM004, IM005, IM006, IM007, IM008, IM010, IM011, IM012.

Dynamic Function Field Value

SQL statement executed	Value of SQL_DIAG_DYNAMIC_FUNCTION	Value of SQL_DIAG_DYNAMIC_FUNCTION_CODE
alter-domain-statement	"ALTER DOMAIN"	SQL_DIAG_ALTER_DOMAIN
alter-table-statement	"ALTER TABLE"	SQL_DIAG_ALTER_TABLE
assertion-definition	"CREATE ASSERTION"	SQL_DIAG_CREATE_ASSERTION
character-set-definition	"CREATE CHARACTER SET"	SQL_DIAG_CREATE_CHARACTER_SET
collation-definition	"CREATE COLLATION"	SQL_DIAG_CREATE_COLLATION
create-index-statement	"CREATE INDEX"	SQL_DIAG_CREATE_INDEX
create-table-statement	"CREATE TABLE"	SQL_DIAG_CREATE_TABLE
create-view-statement	"CREATE VIEW"	SQL_DIAG_CREATE_VIEW
cursor-specification	"SELECT CURSOR"	SQL_DIAG_SELECT_CURSOR
delete-statement-positioned	"DYNAMIC DELETE CURSOR"	SQL_DIAG_DYNAMIC_DELETE_CURSOR
delete-statement-searched	"DELETE WHERE"	SQL_DIAG_DELETE_WHERE
domain-definition	"CREATE DOMAIN"	SQL_DIAG_CREATE_DOMAIN
drop-assertion-statement	"DROP ASSERTION"	SQL_DIAG_DROP_ASSERTION
drop-character-set-stmt	"DROP CHARACTER SET"	SQL_DIAG_DROP_CHARACTER_SET
drop-collation-statement	"DROP COLLATION"	SQL_DIAG_DROP_COLLATION
drop-domain-statement	"DROP DOMAIN"	SQL_DIAG_DROP_DOMAIN
drop-index-statement	"DROP INDEX"	SQL_DIAG_DROP_INDEX
drop-schema-statement	"DROP SCHEMA"	SQL_DIAG_DROP_SCHEMA
drop-table-statement	"DROP TABLE"	SQL_DIAG_DROP_TABLE
drop-translation-statement	"DROP TRANSLATION"	SQL_DIAG_DROP_TRANSLATION
drop-view-statement	"DROP VIEW"	SQL_DIAG_DROP_VIEW
grant-statement	"GRANT"	SQL_DIAG_GRANT
insert-statement	"INSERT"	SQL_DIAG_INSERT
ODBC-procedure-extension	"CALL"	SQL_DIAG_ CALL
revoke-statement	"REVOKE"	SQL_DIAG_REVOKE
schema-definition	"CREATE SCHEMA"	SQL_DIAG_CREATE_SCHEMA
translation-definition	"CREATE TRANSLATION"	SQL_DIAG_CREATE_TRANSLATION
update-statement-positioned	"DYNAMIC UPDATE CURSOR"	SQL_DIAG_DYNAMIC_UPDATE_CURSOR
update-statement-searched	"UPDATE WHERE"	SQL_DIAG_UPDATE_WHERE
Unknown	empty string	SQL_DIAG_UNKNOWN_STATEMENT

Sequence of Status Record

The state record is sequentially located according to the row number and diagnostic type. The driver man ager determines the final step of returning the created state record.

If the diagnostic record is posted by the driver manager and the driver, the driver manager is responsible f or determining the order of the diagnostic records.

If two or more state records exist, the order of the record is determined first by the row number. The follo wing rules are applied to determine the order of the diagnostic record by the row.

- The record not corresponding to any row because SQL_NO_ROW_NUMBER is defined as -1, is positio ned in front of the record corresponding to the particular row.
- The record which does not know the row number because SQL_ROW_NUMBER_UNKNOWN is defin ed as -2 is positioned in front of all the other records.
- For all record which are related to a particular row, the records are sorted by the value of SQL_DIAG ROW_NUMBER. All errors and warnings for the first row are listed, and then all the errors and warnin gs for the next row are sequentially listed.

If SQLSTATE 01S01 is returned by the ODBC 2.x driver, or SQLSTATE 01S01 is returned by the O DBC 3.x driver when calling SQLExtendedFetch or when calling SQLSetPos for the cursor in SQL ExtendedFetch, then the ODBC 3.x driver manager does not require the state record in a diagno sis queue.

Within each row, or for all rows which do not correspond to the row or which do not know the row num ber or which have the row number as same as SQL_NO_ROW_NUMBER, the first record listed is determin ed by using the set of ordering rules. After the first record, the order of other records which affect the ro w is not defined. The application can not assume that an error precedes a warning after the first record. T he application should check the complete diagnostic data structure to get the complete information abou t the failed function call.

The following rules are used to determine the first record in the row. The highest grade record is the first record. The record source (driver manager, driver, gateway, etc.) does not affect the determination of the record ranks.

- Errors state record describing the error has the highest grade. The following rules are applied to align the errors.
  - The record which refers to the transaction failure or possible transaction failure has a higher grad e than any other record.
  - If two or more records describe the same error condition, SQLSTATE defined in the Open Group CLI specifications(Class 03 ~ HZ) has a higher grade than SQLSTATE defined in ODBC and the dri ver.

- Implementation-defined No Data Values state record which describes No Data values (class 02) defin ed by the driver has the second highest rank.
- Warnings state record(class 01) describing the warning has the lowest grade. If two or more records describe the same error condition, SQLSTATE defined in the Open Group CLI specifications has a high er grade than SQLSTATE defined in ODBC and the driver.

SQLGetDiagRec

Conformance

Introduced version: ODBC 3.0
Standards compliance: ISO 92

Overview

SQLGetDiagRec returns the current value of the record field of the diagnostic data source (related to the specified handle) which contains an error, warning, state information.

Syntax

```
SQLRETURN SQLGetDiagRec(
    SQLSMALLINT     HandleType,
    SQLHANDLE       Handle,
    SQLSMALLINT     RecNumber,
    SQLCHAR *       SQLState,
    SQLINTEGER *    NativeErrorPtr,
    SQLCHAR *       MessageText,
    SQLSMALLINT     BufferLength,
    SQLSMALLINT *   TextLengthPtr);
```

Arguments

HandleType

[Input] It is the handle type identifier of which diagnosis is required. The identifier should be one of the followings.

- SQL_HANDLE_DBC
- SQL_HANDLE_DESC
- SQL_HANDLE_ENV
- SQL_HANDLE_STMT

Handle

[Input] It is the handle for the diagnosis data structure of the type pointed by HandleType. If HandleType is SQL_HANDLE_ENV, the handle can be the shared or non-shared environment handle.

RecNumber

[Input] It indicates the state record of the information found by the application. The number of the state record starts from 1.

SQLState

[Output] It is the buffer pointer which returns five character SQLSTATE code for the diagnostic rec ord RecNumber. The first two characters indicate a class and the next three characters indicate a su bclass. The information is in the SQL_DIAG_SQLSTATE diagnostic field.

NativeErrorPtr

[Output] It is the buffer pointer which returns the specific native error code to the data source. The information is in the diagnostic field SQL_DIAG_NATIVE.

MessageText

[Output] It is the buffer point which returns the diagnostic message text string. The information is i n the SQL_DIAG_MESSAGE_TEXT diagnostic field.

If MessageText is NULL, TextLegnthPtr returns the total number of returnable characters (excluding null-termination character) to the buffer pointed by MessageText.

BufferLength

[Input] It is the length of characters within *MessageText buffer. Diagnostic message text does not have the maximum length.

TextLegnthPtr

[Output] It is the buffer pointer to which the total number of characters returnable to *MessageTe xt is returned (excluding null-termination character). If the number of returnable characters is bigg er than BufferLength, the diagnostic message text of *MessageText is truncated to the length of B ufferLength minus null-termination character.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLGetDiagRec does not post the diagnostic record. It uses the result of the return value as follows.

- SQL_SUCCESS: The function successfully returns the diagnosis information.
- SQL_SUCCESS_WITH_INFO: *MessageText buffer is too small to store the requested diagnosis messa ge. The diagnostic record is not generated. The application should compare BufferLength to the actu al byte number stored in *StringLengthPtr to check whether the truncation occurred.
- SQL_INVALID_HANDLE: The handle is not valid for a type indicating HandleType.
- SQL_ERROR: One of the followings occurs.
  - RecNumber argument is the negative number or 0.

    - BufferLength is smaller than 0.
    - If an asynchronous notification is used, the asynchronous execution of handle is not completed.
- SQL_NO_DATA: RecNumber is bigger than the number of diagnosis records for the handle specified i n Handle. If the diagnosis record for handle does not exist, then the function returns SQL_NO_DATA for RecNumber of any positive number.

Description

When the ODBC function returns SQL_ERROR or SQL_SUCCESS_WITH_INFO, generally the application ca lls SQLGetDiagRec. However, the application can call the SQLGetDiagRec after any function is called beca use any ODBC function can post zero or more diagnostic records. The application can call SQLGetDiagRec several times to return some or all records in the diagnostic data structure. ODBC does not have limit on t he number of diagnostic records which can be stored at a time.

SQLGetDiagRec can not be used to return the header field of the diagnostic data structure.(RecNumber s hould be bigger than 0.) The application should call SQLGetDiagField instead of SQLGetDiagRec.

SQLGetDiagRec retrieves only the handle specified to Handle and the latest related diagnostic informatio n. If the application calls any other ODBC function (excluding SQLGetDiagRec, SQLGetDiagField, SQLErro r), diagnostic information from the previous call of the same handle is lost.

The application can repeatedly retrieve the diagnostic record by increasing RecNumber as long as SQLGet DiagRec returns SQL_SUCCESS. SQLGetDiagRec call do not affect the header and record fields. If interven tion of other function does not exist except for SQLGetDiagRec, SQLGetDiagField, SQLError, then the ap plicatin can call SQLGetDiagRec again to retrieve the field in the record. The application can retrieve the t otal number of diagnostic records available by calling SQLGetDiagField to retrieve the value of the SQL_DI AG_NUMBER field, and calling SQLGetDiagRec several times.

HandleType Argument

Each handle type has the related diagnosis information, and HandleType argument refers to the Handle t ype.

Some header and record fields are not returned for the environment, connection, statement, descriptor h andles. In the header field and record field table of **SQLGetDiagField**, the inappropriate handles in the fiel d are described.

If HandleType is SQL_HANDLE_SENV which indicates the shared environment handle, then calling SQLGe tDiagRec returns SQL_INVALID_HANLDE. If HandleType is SQL_HANDLE_ENV, then the handle can be the shared or non-shared environment handle.

SQLGetEnvAttr

Conformance

Introduced version: ODBC 3.0
Standards compliance: ISO 92

Overview

SQLGetEnvAttr returns the current settings of the environment attributes.

Syntax

```
SQLRETURN SQLGetEnvAttr(
    SQLHENV        EnvironmentHandle,
    SQLINTEGER     Attribute,
    SQLPOINTER     ValuePtr,
    SQLINTEGER     BufferLength,
    SQLINTEGER *   StringLengthPtr);
```

Arguments

EnvironmentHandle

[Input] It is the environment handle.

Attribute

[Input] It is the attribute to be retrieved.

ValuePtr

[Output] It is the buffer pointer in which to return the current attribute value specified by Attribute. If ValuePtr is NULL, StringLengthPtr returns the total number of bytes returnable to the buffer pointed by ValuePtr (excluding null-termination character).

BufferLength

[Input] If ValuePtr points to a string, the argument should be the length of *ValuePtr. If ValuePtr is an integer, BufferLength is ignored. If *ValuePtr(when SQLGetEnvAttrW is called) is a unicode string, BufferLength should be an even number. If the attribute value is not a string, BufferLength is not used.

StringLengthPtr

[Output] It is the buffer pointer to which the total number of bytes to be returned to *ValuePtr (excluding null-termination character) is returned. If ValuePtr is the null pointer, the length is not returned. If the attribute value is a string and the number of bytes returnable is equal to or bigger than BufferLength, *ValuePtr is truncated to the length of BufferLength minus the length of a null-termination character and is a null-terminated by the driver.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01000	General Warning	It is the driver-specific informational message. (The function returns SQL_SUCCESS_WITH_INFO.)
01004	String data, right truncated	The data returned to *ValuePtr is truncated to the length of BufferLength minus null-termination character. The remaining length of string is returned to *StringLengthPtr. (The function returns SQL_SUCCESS_WITH_INFO.)
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY010	Function sequence error	SQL_ATTR_ODBC_VERSION is not set through SQLSetEnvAttr. If SQLAllocHandleStd is used, then SQL_ATTR_ODBC_VERSION does not need to be explicitly set.
HY013	Memory management error	The size of buffer used as an argument is smaller than 0 or it can not access the memory.
HY092	Invalid attribute/option identifier	The value specified in attribute argument is not valid for ODBC version supported by the driver.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	For more information about the suspended state, refer to **SQLEndTran**.
HYC00	Optional feature not implemented	The value specified in attribute argument is valid for ODBC environment of ODBC version supported by the driver, but it is not supported by the driver.
IM001	Driver does not support this function	The driver related to DescriptorHandle does not support the function.

Description

The driver-specific environment attributes does not exist. If Attribute specifies the attribute which returns a string, ValuePtr should be the buffer pointer whose string is returned. The maximum string length including the null-termination character is the BufferLength in bytes.

SQLGetEnvAttr can be called at any point between the allocation and release of the environment handle. All environment attributes successfully allocated by the application for the environment are maintained until SQLFreeHandle is called with Handletype of SQL_HANDLE_ENV in EnvrionmentHandle. It is recommended to use only one environment handle.

SQL_ATTR_OUTPUT_NTS environment attributes are supported by the application which complies with standards. If SQLGetEnvAttr is called, ODBC 3.x driver manager always returns SQL_TRUE for the attribute. SQL_ATTR_OUTPUT_NTS is set to SQL_TRUE only by calling SQLSetEnvAttr.

The following table describes attribute list which can be queried through SQLGetEnvAttr.

Attribute	ValuePtr Content
SQL_ATTR_CONNECTION_POOLING (ODBC 3.8)	It is not supported by the driver.
SQL_ATTR_CP_MATCH (ODBC 3.0)	It is not supported by the driver.
SQL_ATTR_ODBC_VERSION (ODBC 3.0)	It is 32-bit integer which indicates whether a particular feature is operated as ODBC 2.x or ODBC 3.x. The following values are used to set the attribute. SQL_OV_ODBC3_80 = The driver manager or driver performs the following ODBC 3.8 behaviors. • The driver expects and returns ODBC 3.x code values for DATE, TIME, TIMESTAMP. • The driver returns ODBC 3.x SQLSTATE codes when SQLError, SQLGetDiagField or SQLGetDiagRec is called. • CatalogName argument of SQLTables allows the pattern matching. SQL_OV_ODBC3 = The driver manager or driver performs the following ODBC 3.x behaviors. • The driver expects and returns ODBC 3.x code values for DATE, TIME, TIMESTAMP. • The driver returns ODBC 3.x SQLSTATE codes when SQLError, SQLGetDiagField or SQLGetDiagRec is called. • CatalogName argument of SQLTables allows the pattern matching. • The driver manager does not support C data type extensibility.

Attribute	ValuePtr Content
	SQL_OV_ODBC2 = The driver manager or driver performs the following ODBC 2.x behaviors. It is very helpful when ODBC 2.x applications operate in the ODBC 3.x driver. • The driver expects and returns ODBC 2.x code values for DATE, TIME, TIMESTAMP. • The driver returns ODBC 2.x SQLSTATE codes when SQLError, SQLGetDiagField or SQLGetDiagRec is called. • CatalogName argument of SQLTables does not allow the pattern matching. • The driver manager does not support C data type extensibility. An application should set this environment attribute before it calls any function which has an SQLHENV argument or a function which returns SQLSTATE HY010 (Function sequence error). The driver should specify whether the additional operations for the environmental flag exists.
SQL_ATTR_OUTPUT_NTS (ODBC 3.0)	It is not supported by the driver.

SQLGetFunctions

Conformance

Introduced version: ODBC 1.0
Standards compliance: ISO 92

Overview

SQLGetFunctions returns information whether the driver supports the given ODBC function. The function is implemented by the driver manager or the driver. If the driver implements SQLGetFunctions, the driver manager will call the function in the driver.

Syntax

```
SQLRETURN SQLGetFunctions(
    SQLHDBC           ConnectionHandle,
    SQLUSMALLINT      FunctionId,
    SQLUSMALLINT *    SupportedPtr);
```

Arguments

ConnectionHandle

[Input] It is the connection handle.

FunctionId

[Input] It is the #define value which identifies the ODBC function related to SQL_API_ODBC3_ALL_FUNCTIONS or SQL_API_ALL_FUNCTIONS. SQL_API_ODBC3_ALL_FUNCTIONS is used in ODBC 3.x application to determine to support the function in ODBC 3.x and in earlier version. SQL_API_ALL_FUNCTIONS is used in ODBC 2.x to determine whether to support the function in ODBC 2.x and in earlier version.

For more information about the value list of #define which identifies the ODBC function, refer to t he table in the description section.

SupportedPtr

[Output] If FunctionId identifies the single ODBC, SupportedPtr points to the single SQLUSMALLIN T value. If the specified function is supported by the driver, it is SQL_TRUE. Otherwise, it is SQL_FA LSE.

If FunctionId is SQL_API_ODBC3_ALL_FUNCTIONS, SupprtedPtr points to the SQLSMALLINT array

which has the number of elements equal to SQL_API_ODBC3_ALL_FUNCTIONS_SIZE. The array is managed by the driver manager as 4000-bit bitmap which can be used to determine to support th e function in ODBC 3.x or in earlier version. SQL_FUNC_EXISTS macro is called to check whether th e function is supported. ODBC 3.x application can call SQLGetFunctions by using SQL_API_ODBC3 _ALL_FUNCTIONS in preparation for ODBC 2.x or ODBC 3.x.

If FunctionId is SQL_API_ALL_FUNCTIONS, SupportedPtr points to the array with 100 elements. Th e array is indexed by #define value used by the FunctionId to identify each ODBC function, and sev eral elements are reserved without being used. If the function in ODBC 2.x or earlier version is supp orted by the driver, the element is SQL_TRUE. If the ODBC function is not supported by the driver o r it is not the ODBC function, the element is SQL_FALSE.

The array that is returned to *SupportedPtr uses the 0-based indexing.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01000	General Warning	It is the driver-specific informational message. (The function returns SQL_SUC CESS_WITH_INFO.)
08S01	Communication link f ailure	Before the function processing is completed, the connection between the driv er and the data source is failed.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation er ror	It is a memory allocation error.
HY010	Function sequence er ror	SQLGetFunctions is called before SQLConnect, SQLBrowseConnect, SQLDriver Connect. SQLBrowseConnect is called for ConnectionHandle, and SQL_NEED_DATA is r eturned. SQLGetfunction is called before SQLBrowseConnect returns SQL_SUCCESS_ WITH_INFO or SQL_SUCCESS. SQLExecute, SQLExecDirect, SQLMoreResults is called for ConnectionHandle, and SQL_PARAM_DATA_AVAILABLE is returned. The function is called, befor e the data retrieves all streamed parameters.
HY013	Memory manageme nt error	The size of buffer used as an argument is smaller than 0, or it can not access t he memory.
HY095	Function type out of range	FunctionId value is not valid.

SQLSTATE	Error	Description
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	For more information about the suspended state, refer to **SQLEndTran**.
HYT01	Connection timeout expired	The connection is expired before the data source response to the request. The connection timeout interval can be set to SQL_ATTR_CONNECTION_TIMEOUT through SQLSetConnectAttr.
IM001	Driver does not support this function	The driver related to DescriptorHandle does not support the function.

Description

SQLGetFunctions returns the supported SQLGetFunctions, SQLDataSources, SQLDrivers. This is because the function is implemented in the driver manager. If an unicode function exist, the driver manager maps the ANSI function corresponding to the Unicode function. If an ANSI function exist, the driver manager maps the unicode function corresponding to the ANSI function.

The following is the list of valid values for FunctionId for the functions which comply with ISO 92 standards compliance level.

- SQL_API_SQLALLOCHANDLE
- SQL_API_SQLBINDCOL
- SQL_API_SQLCANCEL
- SQL_API_SQLCLOSECURSOR
- SQL_API_SQLCOLATTRIBUTE
- SQL_API_SQLCONNECT
- SQL_API_SQLCOPYDESC
- SQL_API_SQLDATASOURCES
- SQL_API_SQLDESCRIBECOL
- SQL_API_SQLDISCONNECT
- SQL_API_SQLDRIVERS
- SQL_API_SQLENDTRAN
- SQL_API_SQLEXECDIRECT
- SQL_API_SQLEXECUTE
- SQL_API_SQLFETCH
- SQL_API_SQLFETCHSCROLL
- SQL_API_SQLFREEHANDLE
- SQL_API_SQLFREESTMT

- SQL_API_SQLGETCONNECTATTR
- SQL_API_SQLGETCURSORNAME
- SQL_API_SQLGETDATA
- SQL_API_SQLGETDESCFIELD
- SQL_API_SQLGETDESCREC
- SQL_API_SQLGETDIAGFIELD
- SQL_API_SQLGETDIAGREC
- SQL_API_SQLGETENVATTR
- SQL_API_SQLGETFUNCTIONS
- SQL_API_SQLGETINFO
- SQL_API_SQLGETSTMTATTR
- SQL_API_SQLGETTYPEINFO
- SQL_API_SQLNUMRESULTCOLS
- SQL_API_SQLPARAMDATA
- SQL_API_SQLPREPARE
- SQL_API_SQLPUTDATA
- SQL_API_SQLROWCOUNT
- SQL_API_SQLSETCONNECTATTR
- SQL_API_SQLSETCURSORNAME
- SQL_API_SQLSETDESCFIELD
- SQL_API_SQLSETDESCREC
- SQL_API_SQLSETENVATTR
- SQL_API_SQLSETSTMTATTR

The following is the list of valid values for FunctionId for the functions which comply with open group sta ndard compliance level.

- SQL_API_SQLCOLUMNS
- SQL_API_SQLSPECIALCOLUMNS
- SQL_API_SQLSTATISTICS
- SQL_API_SQLTABLES

The following is the list of valid values for FunctionId for the functions which comply with ODBC standard s compliance level.

- SQL_API_SQLBINDPARAMETER
- SQL_API_SQLBROWSECONNECT
- SQL_API_SQLBULKOPERATIONS[1]
- SQL_API_SQLCOLUMNPRIVILEGES
- SQL_API_SQLDESCRIBEPARAM
- SQL_API_SQLDRIVERCONNECT
- SQL_API_SQLFOREIGNKEYS

- SQL_API_SQLMORERESULTS
- SQL_API_SQLNATIVESQL
- SQL_API_SQLNUMPARAMS
- SQL_API_SQLPRIMARYKEYS
- SQL_API_SQLPROCEDURECOLUMNS
- SQL_API_SQLPROCEDURES
- SQL_API_SQLSETPOS
- SQL_API_SQLTABLEPRIVILEGES

[1] When operating with ODBC 2.x driver, SQLBulkOperations is supported and returned only when the both of the followings are true. ODBC 2.x driver supports SQLSetPos, and the SQL_POS_OPERATIONS information type returns SQL_POS_ADD bit as set.

The valid value of FunctionId for the functions introduced after ODBC 3.8 is SQL_API_SQLCANCELHANDLE. [2]

[2]SQLCancelHandle is returned only when the driver supports both of SQLCancel and SQLCancelHandle. If SQLCancel is supported but SQLCancelHandle is not suported, the application can still call SQLCancelHandle for the statement handle, because it is mapped to SQLCancel.

SQL_FUNC_EXISTS Macro

SQL_FUNC_EXISTS (SupportedPtr, FunctionID) macro is used to call SQLGetFunctions to SQL_API_ODBC3_ALL_FUNCTIONS by using FunctionId argument and to check the function supported in ODBC 3.x or in the earlier version. The application uses SQL_FUNC_EXISTS by setting SupportedPtr transferred from SQLGetFuncions to SupportedPtr argument and by setting the FunctionID argument to the #define for the function. If the function is supported, SQL_FUNC_EXIST returns SQL_TRUE. Otherwise, it returns SQL_FALSE.

When operating with ODBC 2.x driver, ODBC 3.x driver manager returns SQL_TRUE for SQLAllocHandle and SQLFreeHandle. It is because SQLAllocHandle is mapped to SQLAllocEnv, SQLAllocConnect orSQLAllocStmt, and SQLFreeHandle is mapped to SQLFreeEnv, SQLFreeConnect or SQLFreeStmt. However, although SQLFreeHandle returns SQL_TRUE, SQLFreeHandle whose SQL_HANDLE_DESC is used as HandleType argument is not supported. It is because function mapped to ODBC 2.x function for this case does not exist.

SQLGetGroupCount

Conformance

Standards compliance: It is not available.

Overview

SQLGetGroupCount returns the number of cluster groups.

Syntax

```
SQLRETURN SQLGetGroupCount(
    SQLHDBC      ConnectionHandle,
    SQLINTEGER * GroupCountPtr );
```

Arguments

ConnectionHandle

[Input] It is the connection handle.

GroupCountPtr

[Output] It is the number of cluster groups.

Returns

SQL_SUCCESS, SQL_NO_DATA, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
08003	Connection not open	ConnectionHandle is not in a connected state.
08S01	Communication link failure	Before the function processing is completed, the connection between the driver and the data source is failed.
HY010	Function sequence error	This function should be called only when the connection property SQL_ATTR_LOCALITY_AWARE_TRANSACTION is set.

Description

SQLGetGroupCount can be called only when the connection property SQL_ATTR_LOCALITY_AWARE_TRANSACTION is set.

SQLGetGroupIDs

Conformance

Standards compliance: It is not available.

Overview

SQLGetGroupIDs returns IDs of cluster groups.

Syntax

```
SQLRETURN SQLGetGroupIDs(
    SQLHDBC      ConnectionHandle,
    SQLINTEGER * GroupIDArray );
```

Arguments

ConnectionHandle

[Input] It is the connection handle.

GroupIDArray

[output] It is the array of cluster group IDs.

Returns

SQL_SUCCESS, SQL_NO_DATA, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
08003	Connection not open	ConnectionHandle is not in a connected state.
08S01	Communication link failure	Before the function processing is completed, the connection between the driver and the data source is failed.
HY010	Function sequence error	This function should be called only when the connection property SQL_ATTR_LOCALITY_AWARE_TRANSACTION is set.

Description

SQLGetGroupID can be called only when the connection property SQL_ATTR_LOCALITY_AWARE_TRANSACTION is set.
The number of elements of GroupIDArray should be as same as the number of cluster groups which are returned by SQLGetGroupCount.

SQLGetGroupName

Conformance

Standards compliance: It is not available.

Overview

SQLGetGroupName returns the name of the cluster group which corresponds to GroupID.

Syntax

```
SQLRETURN SQLGetGroupName(
    SQLHDBC       ConnectionHandle,
    SQLINTEGER    GroupID,
    SQLCHAR     * GroupName,
    SQLSMALLINT   BufferLength,
    SQLSMALLINT * NameLengthPtr );
```

Arguments

ConnectionHandle

[Input] It is the connection handle.

GroupID

[Input] It is the cluster group ID.

GroupName

[output] It is the buffer pointer which returns the name of the cluster group which is terminated with null.

BufferLength

[Input] It is the length of *GroupName.

NameLengthPtr

[Output] It is the buffer pointer which returns the total number of bytes returnable to *ColumnName (excluding the null-termination character). If the returnable length is equal to or bigger than BufferLength, *ColumnName is truncated to the length of BufferLength minus null.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NO_DATA, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01004	String data, right truncated	The size of *GroupName is not large enough to return the name of the cluster group, so the name of the cluster group is truncated. The length of the cluster name which is not truncated is returned in *NameLengthPtr. (The function returns SQL_SUCCESS_WITH_INFO.)
08003	Connection not open	ConnectionHandle is not in a connected state.
08S01	Communication link failure	Before the function processing is completed, the connection between the driver and the data source is failed.
HY010	Function sequence error	This function should be called only when the connection property SQL_ATTR_LOCALITY_AWARE_TRANSACTION is set.

Description

SQLGetGroupIDs can be called only when the connection property SQL_ATTR_LOCALITY_AWARE_TRANSACTION is set.

SQLGetInfo

Conformance

Introduced version: ODBC 1.0
Standards compliance: ISO 92

Overview

SQLGetInfo returns the general information on the connection associated with the driver and the data source.

Syntax

```
SQLRETURN SQLGetInfo(
    SQLHDBC         ConnectionHandle,
    SQLUSMALLINT    InfoType,
    SQLPOINTER      InfoValuePtr,
    SQLSMALLINT     BufferLength,
    SQLSMALLINT *   StringLengthPtr);
```

Arguments

ConnectionHandle

[Input] It is the connection handle.

InfoType

[Input] It is the information type.

InfoValuePtr

[Output] It is the buffer pointer whose information is returned. According to InfoType requested, one of the following information will be returned. Null termination character, SQLUSMALLINT value, SQLUINTEGER bit mask, SQLUINTEGER flag, SQLUINTEGER binary value, or SQLULEN value.

If InfoType argument is SQL_DRIVER_HDESC or SQL_DRIVER_HSTMT, InfoValuePtr argument is input and output.

If InfoValuePtr is NULL, StringLengthPtr returns the total number of returnable bytes to the buffer pointed by InfoValuePtr (null termination character is excluded).

BufferLength

[Input] It is the length of *InfoValuePtr buffer. If *InfoValuePtr is not a string or InfoValuePtr is the null pointer, BufferLength argument is ignored. The driver assumes the size of *InfoValuePtr as SQLUSMALLINT or SQLUINTEGER according to InfoType. (When SQLGetInfoW is called) if InfoValuePtr is a unicode string, BufferLength must be an even number. Otherwise SQLSTATE HY090 is returned.

StringLengthPtr

[Output] It is the buffer pointer which returns the total number of bytes returnable to *InfoValuePtr (excluding null-termination character for the character data).

If the number of bytes returnable for the character is equal to or bigger than BufferLength, the information in *InfoValuePtr is truncated to the length of BufferLength in byte (excluding null-termination character) and is null terminated by the driver.

BufferLength is ignored for the data of the other data types, and the driver assumes the size of *InfoValuePtr as SQLUSMALLINT or SQLUINTEGER according to InfoType.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, or SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01000	General Warning	It is the driver-specific informational message. (The function returns SQL_SUCCESS_WITH_INFO.)
01004	String data, right truncated	The buffer length of *InfoValuePtr is not large enough to return all required information, so the information is truncated. The length of untruncated information is returned to *StringLengthPtr. (The function returns SQL_SUCCESS_WITH_INFO.)
08003	Connection not open	The connection should be open for the type of information requested in InfoType. SQL_ODBC_VER which is the reserved information in ODBC can be returned without open connection.
08S01	Communication link failure	Before the function processing is completed, the connection between the driver and the data source is failed.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY010	Function sequence error	SQLExecute, SQLExecDirect, or SQLMoreResults is called for StatementHandle, and SQL_PARAM_DATA_AVAILABLE is returned. The function is called, before the data retrieves all streamed parameters.
HY013	Memory management error	The size of buffer used as an argument is smaller than 0, or it can not access the memory.

SQLSTATE	Error	Description
HY024	Invalid attribute valu e	The InfoType argument is SQL_DRIVER_HSTMT, and the value pointed by InfoValuePtr is an invalid statetment handle. The InfoType argument is SQL_DRIVER_HDESC, the value pointed by Info ValuePtr is an invalid descriptor handle.
HY090	Invalid string or buff er length	The value for BufferLength argument is smaller than 0. The value for BufferLength argument is an odd number, and *InfoValueP tr is the unicode data type.
HY096	Information type ou t of range	The value specified in InfoType argument is not valid for ODBC version w hich is supported by the driver.
HY117	Connection is suspe nded due to unkno wn transaction state. Only disconnect and read-only functions are allowed.	For more information about the suspended state, refer to **SQLEndTran**.
HYC00	Optional field not i mplemented	The value specified in InfoType is the specific driver value which is not sup ported by the driver.
HYT01	Connection timeout expired	The connection is expired before the data source responds to the request. The timeout can be set via SQL_ATTR_CONNECTION_TIMEOUT of SQLSet StmtAttr.
IM001	Driver does not sup port this function	The driver which is related to DescriptorHandle does not support the func tion.

Description

The currently defined information types will be described in the following sections. The range of informati on types is reserved by ODBC. The driver developer should reserve the values for the driver-specific use of their own from Open Group. SQLGetInfo does not perform unicode conversion or thunking for InfoTypes defined by the driver. The information type returned to *InfoValuePtr is determined according to InfoTyp es requested.

SQLGetInfo returns one of the following five types.

- Null-termination string
- SQLUSMALLINT value
- SQLUINTEGER bit mask
- SQLUINTEGER value
- SQLUINTEGER binary value

The application should cast according to the value returned in *InfoValuePtr.

The driver should return a value for each information type defined in the following table. If the informatio n type is not applied to the driver or the data source, the driver has to return one of the following values.

Character string ("Y" or "N")

"N"

Character string (not "Y" or "N")

Empty string

SQLUSMALLINT

0

SQLUINTEGER bit mask or SQLUINTEGER binary value

0L

For example, if the data source does not support the procedure, SQLGetInfo returns the following value f or the value of InfoType related to the procedure.

SQL_PROCEDURES

"N"

SQL_ACCESSIBLE_PROCEDURES

"N"

SQL_MAX_PROCEDURE_NAME_LEN

0

SQL_PROCEDURE_TERM

Empty string

SQLGetInfo returns SQLSTATE HY096 for the value of InfoType which is in the range of information type reserved for use by ODBD but which is not defined by the ODBC version supported by the driver. The driv er compiles and calls SQLGetInfo as the SQL_DRIVER_ODBC_VER information type of an application to de termine the ODBC version. SQLGetInfo returns SQLSTATE HYC00 for the value of InfoType which is in the range of reserved information type for the separate use of the driver but which is not supported by the dr iver.

Calling all SQLGetInfo should be open for connection, except for InfoType which returns the driver mana ger version is SQL_ODBC_VER.

Driver Information

The following values of InfoType argument return ODBC information such as an active statement, data so urce name, interface standards compliance level.

- SQL_ACTIVE_ENVIRONMENTS
- SQL_ASYNC_DBC_FUNCTIONS
- SQL_ASYNC_MODE
- SQL_ASYNC_NOTIFICATION
- SQL_BATCH_ROW_COUNT
- SQL_BATCH_SUPPORT
- SQL_DATA_SOURCE_NAME
- SQL_DRIVER_AWARE_POOLING_SUPPORTED
- SQL_DRIVER_HDBC
- SQL_DRIVER_HDESC
- SQL_DRIVER_HENV
- SQL_DRIVER_HLIB
- SQL_DRIVER_HSTMT
- SQL_DRIVER_NAME
- SQL_DRIVER_ODBC_VER
- SQL_DRIVER_VER
- SQL_DYNAMIC_CURSOR_ATTRIBUTES1
- SQL_DYNAMIC_CURSOR_ATTRIBUTES2
- SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1
- SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2
- SQL_FILE_USAGE
- SQL_GETDATA_EXTENSIONS
- SQL_INFO_SCHEMA_VIEWS
- SQL_KEYSET_CURSOR_ATTRIBUTES1
- SQL_KEYSET_CURSOR_ATTRIBUTES2
- SQL_MAX_ASYNC_CONCURRENT_STATEMENTS
- SQL_MAX_CONCURRENT_ACTIVITIES
- SQL_MAX_DRIVER_CONNECTIONS
- SQL_ODBC_INTERFACE_CONFORMANCE
- SQL_ODBC_STANDARD_CLI_CONFORMANCE
- SQL_ODBC_VER
- SQL_PARAM_ARRAY_ROW_COUNTS
- SQL_PARAM_ARRAY_SELECTS
- SQL_ROW_UPDATES
- SQL_SEARCH_PATTERN_ESCAPE
- SQL_SERVER_NAME
- SQL_STATIC_CURSOR_ATTRIBUTES1
- SQL_STATIC_CURSOR_ATTRIBUTES2

When performing SQLGetInfo, the driver can improve the performance by minimizing the number of requests or of the number of transmitting information from the server.

DBMS Product Information

The following values of InfoType argument return information on the DBMS product such as DBMS name and information.

- SQL_DATABASE_NAME
- SQL_DBMS_NAME
- SQL_DBMS_VER

Data Source Information

The following values of InfoType argument return information on the data source such as cursor characteristics and transaction features.

- SQL_ACCESSIBLE_PROCEDURES
- SQL_ACCESSIBLE_TABLES
- SQL_BOOKMARK_PERSISTENCE
- SQL_CATALOG_TERM
- SQL_COLLATION_SEQ
- SQL_CONCAT_NULL_BEHAVIOR
- SQL_CURSOR_COMMIT_BEHAVIOR
- SQL_CURSOR_ROLLBACK_BEHAVIOR
- SQL_CURSOR_SENSITIVITY
- SQL_DATA_SOURCE_READ_ONLY
- SQL_DEFAULT_TXN_ISOLATION
- SQL_DESCRIBE_PARAMETER
- SQL_MULT_RESULT_SETS
- SQL_MULTIPLE_ACTIVE_TXN
- SQL_NEED_LONG_DATA_LEN
- SQL_NULL_COLLATION
- SQL_PROCEDURE_TERM
- SQL_SCHEMA_TERM
- SQL_SCROLL_OPTIONS
- SQL_TABLE_TERM
- SQL_TXN_CAPABLE
- SQL_TXN_ISOLATION_OPTION
- SQL_USER_NAME

Supported SQL

The following values of InfoType argument return Information on the SQL statements supported by the d ata source.

- SQL_AGGREGATE_FUNCTIONS
- SQL_ALTER_DOMAIN
- SQL_ALTER_SCHEMA
- SQL_ALTER_TABLE
- SQL_ANSI_SQL_DATETIME_LITERALS
- SQL_CATALOG_LOCATION
- SQL_CATALOG_NAME
- SQL_CATALOG_NAME_SEPARATOR
- SQL_CATALOG_USAGE
- SQL_COLUMN_ALIAS
- SQL_CORRELATION_NAME
- SQL_CREATE_ASSERTION
- SQL_CREATE_CHARACTER_SET
- SQL_CREATE_COLLATION
- SQL_CREATE_DOMAIN
- SQL_CREATE_SCHEMA
- SQL_CREATE_TABLE
- SQL_CREATE_TRANSLATION
- SQL_DDL_INDEX
- SQL_DROP_ASSERTION
- SQL_DROP_CHARACTER_SET
- SQL_DROP_COLLATION
- SQL_DROP_DOMAIN
- SQL_DROP_SCHEMA
- SQL_DROP_TABLE
- SQL_DROP_TRANSLATION
- SQL_DROP_VIEW
- SQL_EXPRESSIONS_IN_ORDERBY
- SQL_GROUP_BY
- SQL_IDENTIFIER_CASE
- SQL_IDENTIFIER_QUOTE_CHAR
- SQL_INDEX_KEYWORDS
- SQL_INSERT_STATEMENT
- SQL_INTEGRITY
- SQL_KEYWORDS
- SQL_LIKE_ESCAPE_CLAUSE
- SQL_NON_NULLABLE_COLUMNS

- SQL_SQL_CONFORMANCE
- SQL_OJ_CAPABILITIES
- SQL_ORDER_BY_COLUMNS_IN_SELECT
- SQL_OUTER_JOINS
- SQL_PROCEDURES
- SQL_QUOTED_IDENTIFIER_CASE
- SQL_SCHEMA_USAGE
- SQL_SPECIAL_CHARACTERS
- SQL_SUBQUERIES
- SQL_UNION

SQL Restrictions

The following values of the InfoType argument return restrictions applied to identifiers and clauses in SQL statements, such as the maximum lengths of identifiers and the maximum number of columns in a select list. Restriction can be imposed by the driver or data source.

- SQL_MAX_BINARY_LITERAL_LEN
- SQL_MAX_CATALOG_NAME_LEN
- SQL_MAX_CHAR_LITERAL_LEN
- SQL_MAX_COLUMN_NAME_LEN
- SQL_MAX_COLUMNS_IN_GROUP_BY
- SQL_MAX_COLUMNS_IN_INDEX
- SQL_MAX_COLUMNS_IN_ORDER_BY
- SQL_MAX_COLUMNS_IN_SELECT
- SQL_MAX_COLUMNS_IN_TABLE
- SQL_MAX_CURSOR_NAME_LEN
- SQL_MAX_IDENTIFIER_LEN
- SQL_MAX_INDEX_SIZE
- SQL_MAX_PROCEDURE_NAME_LEN
- SQL_MAX_ROW_SIZE
- SQL_MAX_ROW_SIZE_INCLUDES_LONG
- SQL_MAX_SCHEMA_NAME_LEN
- SQL_MAX_STATEMENT_LEN
- SQL_MAX_TABLE_NAME_LEN
- SQL_MAX_TABLES_IN_SELECT
- SQL_MAX_USER_NAME_LEN

Scalar Funtion Information

The following values of InfoType argument return information on the scalar functions supported by the d ata source or driver.

- SQL_CONVERT_FUNCTIONS
- SQL_NUMERIC_FUNCTIONS
- SQL_STRING_FUNCTIONS
- SQL_SYSTEM_FUNCTIONS
- SQL_TIMEDATE_ADD_INTERVALS
- SQL_TIMEDATE_DIFF_INTERVALS
- SQL_TIMEDATE_FUNCTIONS

Conversion Information

The following values of InfoType argument return the list of SQL data types of which the data source can be converted to the specified SQL data type with CONVERT scalar function.

- SQL_CONVERT_BIGINT
- SQL_CONVERT_BINARY
- SQL_CONVERT_BIT
- SQL_CONVERT_CHAR
- SQL_CONVERT_DATE
- SQL_CONVERT_DECIMAL
- SQL_CONVERT_DOUBLE
- SQL_CONVERT_FLOAT
- SQL_CONVERT_INTEGER
- SQL_CONVERT_INTERVAL_YEAR_MONTH
- SQL_CONVERT_INTERVAL_DAY_TIME
- SQL_CONVERT_LONGVARBINARY
- SQL_CONVERT_LONGVARCHAR
- SQL_CONVERT_NUMERIC
- SQL_CONVERT_REAL
- SQL_CONVERT_SMALLINT
- SQL_CONVERT_TIME
- SQL_CONVERT_TIMESTAMP
- SQL_CONVERT_TINYINT
- SQL_CONVERT_VARBINARY
- SQL_CONVERT_VARCHAR

Added Information Type in ODBC 3.x

The following values of InfoType argument are added to ODBC 3.x.

- SQL_ACTIVE_ENVIRONMENTS
- SQL_AGGREGATE_FUNCTIONS
- SQL_ALTER_DOMAIN

- SQL_ALTER_SCHEMA
- SQL_ANSI_SQL_DATETIME_LITERALS
- SQL_ASYNC_DBC_FUNCTIONS
- SQL_ASYNC_MODE
- SQL_ASYNC_NOTIFICATION
- SQL_BATCH_ROW_COUNT
- SQL_BATCH_SUPPORT
- SQL_CATALOG_NAME
- SQL_COLLATION_SEQ
- SQL_CONVERT_INTERVAL_YEAR_MONTH
- SQL_CONVERT_INTERVAL_DAY_TIME
- SQL_CREATE_ASSERTION
- SQL_CREATE_CHARACTER_SET
- SQL_CREATE_COLLATION
- SQL_CREATE_DOMAIN
- SQL_CREATE_SCHEMA
- SQL_CREATE_TABLE
- SQL_CREATE_TRANSLATION
- SQL_CURSOR_SENSITIVITY
- SQL_DDL_INDEX
- SQL_DESCRIBE_PARAMETER
- SQL_DM_VER
- SQL_DRIVER_AWARE_POOLING_SUPPORTED
- SQL_DRIVER_HDESC
- SQL_DROP_ASSERTION
- SQL_DROP_CHARACTER_SET
- SQL_DROP_COLLATION
- SQL_DROP_DOMAIN
- SQL_DROP_SCHEMA
- SQL_DROP_TABLE
- SQL_DROP_TRANSLATION
- SQL_DROP_VIEW
- SQL_DYNAMIC_CURSOR_ATTRIBUTES1
- SQL_DYNAMIC_CURSOR_ATTRIBUTES2
- SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1
- SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES2
- SQL_INFO_SCHEMA_VIEWS
- SQL_INSERT_STATEMENT
- SQL_KEYSET_CURSOR_ATTRIBUTES1
- SQL_KEYSET_CURSOR_ATTRIBUTES2
- SQL_MAX_ASYNC_CONCURRENT_STATEMENTS

- SQL_MAX_IDENTIFIER_LEN
- SQL_PARAM_ARRAY_ROW_COUNTS
- SQL_PARAM_ARRAY_SELECTS
- SQL_STATIC_CURSOR_ATTRIBUTES1
- SQL_STATIC_CURSOR_ATTRIBUTES2
- SQL_XOPEN_CLI_YEAR

Renamed Information Type in ODBC 3.x

The following values of InfoType argument are renamed for ODBC 3.x.

- SQL_ACTIVE_CONNECTIONS: SQL_MAX_DRIVER_CONNECTIONS
- SQL_ACTIVE_STATEMENTS: SQL_MAX_CONCURRENT_ACTIVITIES
- SQL_MAX_OWNER_NAME_LEN: SQL_MAX_SCHEMA_NAME_LEN
- SQL_MAX_QUALIFIER_NAME_LEN: SQL_MAX_CATALOG_NAME_LEN
- SQL_ODBC_SQL_OPT_IEF: SQL_INTEGRITY
- SQL_OWNER_TERM: SQL_SCHEMA_TERM
- SQL_OWNER_USAGE: SQL_SCHEMA_USAGE
- SQL_QUALIFIER_LOCATION: SQL_CATALOG_LOCATION
- SQL_QUALIFIER_NAME_SEPARATOR: SQL_CATALOG_NAME_SEPARATOR
- SQL_QUALIFIER_TERM: SQL_CATALOG_TERM
- SQL_QUALIFIER_USAGE: SQL_CATALOG_USAGE

Deprecated Information Type in ODBC 3.x

The following InfoType argument values are information types which are deprecated in ODBC 3.x driver b ut are still supported for compatibility with ODBC 2.x applications.

- SQL_FETCH_DIRECTION
- SQL_LOCK_TYPES
- SQL_ODBC_API_CONFORMANCE
- SQL_ODBC_SQL_CONFORMANCE
- SQL_POS_OPERATIONS
- SQL_POSITIONED_STATEMENTS
- SQL_SCROLL_CONCURRENCY
- SQL_STATIC_SENSITIVITY

SQLGetStmtAttr

Conformance

Introduced version: ODBC 3.0
Standards compliance: ISO 92

Overview

SQLGetStmtAttr returns the current setting of the statement attribute.

Syntax

```
SQLRETURN SQLGetStmtAttr(
    SQLHSTMT        StatementHandle,
    SQLINTEGER      Attribute,
    SQLPOINTER      ValuePtr,
    SQLINTEGER      BufferLength,
    SQLINTEGER *    StringLengthPtr);
```

Arguments

StatementHandle

[Input] It is the statement handle.

Attribute

[Input] It is the attribute to be retrieved.

ValuePtr

[Output] It is the buffer pointer in which to return the attribute value specified in Attribute. If ValuePtr is NULL, StringLengthPtr returns the total number of bytes returnable to the buffer pointed by ValuePtr (excluding null-termination character).

BufferLength

[Input] If Attribute is the attribute defined in ODBC and ValuePtr points to a string or binary buffer, the argument is the length of *ValuePtr. If Attribute is the attribute defined in ODBC and *ValuePtr is an integer, BufferLength is ignored. If *ValuePtr (when SQLGetStmtAttrW is called) is a unicode string, BufferLength should be an even number.

If Attribute is the driver defined attribute, the application displays the attribute characteristic to the driver manager by setting BufferLength argument. BufferLength has one of the following values.

• If *ValuePtr is a string pointer, BufferLength is the string length or SQL_NTS.

• If *ValuePtr is a binary buffer pointer, the application stores SQL_LEN_BINARY_ATTR (length) macro result in BufferLength. It stores an negative number in BufferLength.

• If *ValuePtr is a pointer of the value other than a string or a binary string, BufferLength has the value of SQL_IS_POINTER.

• If *ValuePtr has the fixed length data type, BufferLength has SQL_IS_INTEGER or SQL_IS_UINTEGER.

StringLengthPtr

[Output] It is the buffer pointer which returns the total number of bytes returnable to *ValuePtr (excluding null-termination character). If ValuePtr is null, the length is not returned. If the attribute value is a string and the bytes returnable is equal to or bigger than BufferLength, the data in *ValuePtr is truncated to the length of BufferLength minus the null-termination character, and it is null-terminated by the driver).

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01000	General warning	It is the driver-specific informational message. (The function returns SQL_SUCCESS_WITH_INFO.)
01004	String data, right truncated	The data returned to *ValuePtr is truncated to the length of BufferLength minus null-termination character. The untruncated string value is returned in *StringLengthPtr. (The function returns SQL_SUCCESS_WITH_INFO.)
24000	Invalid cursor state	Attribute argument is SQL_ATTR_ROW_NUMBER, and the cursor is not open or the cursor is positioned before or after the result set.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY010	Function sequence error	The asynchronously executing function is called for the connection handle related to StatementHandle, and this function is still being executed when SQLGetStmtAttr is called. The asynchronously executing function is called for StatementHandle, and is still being executed when the function is called. SQLExecute, SQLExecDirect, SQLBulkOperation or SQLSetPos are called for StatementHandle, and SQL_NEED_DATA is returned. This functio

SQLSTATE	Error	Description
		n is called before data is sent for all data-at-execution parameters or columns.
HY013	Memory management error	The size of buffer used as an argument is smaller than 0, or it can not access the memory.
HY090	Invalid string or buffer length	*ValuePtr is a string, and BufferLength is smaller than 0 but it is not equal to SQL_NTS.
HY092	Invalid attribute/ option identifier	The specified value of Attribute argument is not valid for ODBC version supported by the driver.
HY109	Invalid cursor position	Attribute argument is SQL_ATTR_ROW_NUMBER, and the row is deleted or can not be retrieved.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	For more information about the suspended state, refer to **SQLEndTran**.
HYC00	Optional field not implemented	The value specified in attribute argument is valid ODBC statement attribute in ODBC version supported by the driver, but it is not supported by the driver.
HYT01	Connection timeout expired	The connection is expired before the data source responds to the request. The timeout can be set via SQL_ATTR_CONNECTION_TIMEOUT of SQLSetStmtAttr.
IM001	Driver does not support this function	The driver which is related to DescriptorHandle does not support the function.

Description

SQLGetStmtAttr returns the statement attribute value specified in Attribute to *ValuePtr. The value returned can be the SQLULEN value or null-termination string. If the value is SQLULEN, some drivers write the lower 32 bits or 16 bits of the buffer and keep the upper bits as they are. Therefore, the application should use the buffer of SQLULEN and initialize it to 0 before the function is called. Also BufferLength and StringLengthPtr are not used. If the value is the null termination string, the application specifies the maximum string length of BufferLength argument, and the driver returns the string length to *StringLengthPtr buffer.

If an application which is operated with ODBC 2.x driver wants to call SQLGetStmtAttr, SQLGetStmtAttr should be mapped to the SQLGetStmtOption of the driver manager.

The following statement can be retrieved by SQLGetStmtAttr but it can not set by SQLSetStmtAttr because the attributes are read-only.

- SQL_ATTR_IMP_PARAM_DESC

- SQL_ATTR_IMP_ROW_DESC
- SQL_ATTR_ROW_NUMBER

Statement Attributes

The following table describes currently defined attributes and its introduced ODBC version.

Attribute	ValuePtr contents
SQL_ATTR_APP_PARAM_DESC (ODBC 3.0)	It is the APD handle for the time after calling SQLExecute and SQLExecDirect on the statement handle. The initial value of this attribute is the descriptor implicitly allocated when the statement is initially allocated. If the value of this attribute is set to the descriptor which is originally allocated by SQL_NULL_DESC or the handle, an explicitly allocated APD handle which is previously related to the statement handle is detached from it and the statement handle is returned to the implicitly allocated APD handle. This attribute can not be set to a descriptor handle which is implicitly allocated for another statement, nor set to another descriptor handle which is implicitly allocated for the same statement. The implicitly allocated descriptor handles cannot be related to a statement or a descriptor handle.
SQL_ATTR_APP_ROW_DESC (ODBC 3.0)	It is the ARD handle for the next fetches of the statement handle. The initial value of the attribute is the descriptor implicitly allocated when the statement is initially allocated. If the attribute is set to SQL_NULL_DESC or to the descriptor allocated by the handle, ARD handle which is explicitly allocated and related to the previous statement is detached and the statement handle is returned to ARD handle implicitly allocated. The attribute can not be set to a descriptor handle which is implicitly allocated for another statement, nor set to another descriptor handle which is implicitly set on the same statement. The implicitly allocated descriptor handle can not be related to a statement or descriptor handle.
SQL_ATTR_ASYNC_ENABLE (ODBC 1.0)	It is not supported by the driver.
SQL_ATTR_ASYNC_STMT_EVENT (ODBC 3.8)	It is not supported by the driver.
SQL_ATTR_ASYNC_STMT_PCALLBACK (ODBC3.8)	It is not supported by the driver.
SQL_ATTR_ASYNC_STMT_PCONTEXT (ODBC 3.8)	It is not supported by the driver.
	• SQLULEN: It is the value which specifies the concurrency of cursor. • SQL_CONCUR_READ_ONLY: The cursor is read-only, and the update is not allowed.

Attribute	ValuePtr contents
SQL_ATTR_CONCURRENCY (ODBC 2.0)	• SQL_CONCUR_LOCK: The cursor uses the locking of the minimum level which is enough to complete update the row. • SQL_CONCUR_POWVER: The cursor uses the concurrency which controls and compares the row version such as SQLBase ROWID or Sybase TIMESTAMP. • SQL_CONCUR_VALUES: The cursor uses the concurrency which controls and compares the value. The default value of SQL_ATTR_CONCURRENCY is SQL_CONCUR_READ_ONLY. If the Attribute of SQL_ATTR_CURSOR_TYPE is changed to the value not supported by SQL_ATTR_CONCURRENCY, then the value of SQL_ATTR_CONCURRENCY is changed at the execution time, and the warning will be issued when SQLExecDirect or SQLPrepare is called. If SQL_ATTR_CONCURRENCY is changed to SQL_CONCUR_READ_ONLY, while the driver supports and executes SELECT FOR UPDATE statement, then an error is returned. If SQL_ATTR_CONCURRENCY value is changed to SQL_ATTR_CURSOR_TYPE value supported by the driver or the value not supported, SQL_ATTR_CURSOR_TYPE value is changed at the execution time, SQLSTATE 01S02 (Option value changed) is issued when SQLExecDirect or SQLPrepare is executed. If the specified concurrency is not supported in the data source, the driver replaces it with another concurrency and returns SQLSTATE 01S02 (Option value changed). The driver replaces SQL_CONCUR_VALUES with SQL_CONCUR_ROWVER, or vice versa. Also SQL_CONCUR_LOCK is replaced with SQL_CONCUR_ROWVER, SQL_CONCUR_VALUES in order. The validity of the replaced value is not confirmed until the execution time.
SQL_ATTTR_CURSOR_SCROLLABLE (ODBC 3.0)	• SQL_NONSCROLLABLE: It is the default value. The scroll cursor is not required in the statement handle. If the application calls SQLFetchScroll on the handle, the only valid value of FetchOrientation is SQL_FETCH_NEXT. • SQLULEN: It is the value which specifies the required support level of the application. Setting the attribute affects the following SQLExecDirect and SQLExecute. • SQL_SCROLLABLE: The scroll cursor is required in the statement handle. When SQLFetchScroll is called, the application positions the cursor in a mode except sequential mode and specify a valid value of FetchOrientation.
iSQL_ATTR_CURSOR_SENSITIVITY	• SQL_UNSPECIFIED: It is the default value. It is unspecified what the cursor type is and whether cursors on the statement handle can view the changes made to a result set by another cursor. The cursor of the statement handle probably view is none, some, or all. • SQLULEN: It is the value which determines whether the cursor of the statement handle can view a change in the result set generated by another cursor. Setting

Attribute	ValuePtr contents
(ODBC 3.0)	the attribute affects the following SQLExecDirect and SQLExecute. The application can read the attribute again to obtain the initial state or state most recently set by the application. • SQL_INSENSITIVE: All cursors of the statement handle can not view the reflection of any changes made by another cursor and they can view only the results. The insensitive cursor is read-only. It corresponds to a static cursor with a read-only concurrency. • SQL_SENSITIVE: All cursors of the statement handle can view all changes in the result set.
SQL_ATTR_CURSOR_TYPE (ODBC 2.0)	• SQLULEN: It is the value which specifies the cursor type. • SQL_CURSOR_FORWARD_ONLY: The cursor can only move forward. • SQL_CURSOR_KEYSET_DRIVEN: The driver stores and uses keys as many as the number of rows specified in SQL_ATTR_KEYSET_SIZE which is the statement attribute. • SQL_CURSOR_DYNAMIC: The driver stores and uses the key for the row in the row set. The default value is SQL_CURSOR_FORWARD_ONLY. The attribute can not be specified in the prepared SQL statement. If the specified cursor type is not supported by the data source, then the driver replaces it with other type and returns SQLSTATE 01S02 (Option value changed). The driver replaces a mixed or dynamic cursor with a key set-driven or static cursor. The driver replaces a key set-driven cursor with a static cursor.
SQL_ATTR_ENABLE_AUTO_IPD (ODBC 3.0)	It is not supported by the driver.
SQL_ATTR_FETCH_BOOKMARK_PTR (ODBC 3.0)	It is not supported by the driver.
SQL_ATTR_IMP_PARAM_DESC (ODBC 3.0)	It is the handle of IPD. This attribute value is the descriptor allocated when the statement is initially allocated. The application can not set the attribute. The attribute can be retrieved by calling SQLGetStmtAttr, but it can not be set via SQLSetStmtAttr.
SQL_ATTR_IMP_ROW_DESC (ODBC 3.0)	It is the handle of IRD. This attribute value is the descriptor allocated when the statement is initially allocated. The application can not set the attribute. The attribute can be retrieved by calling SQLGetStmtAttr, but it can not be set via SQLSetStmtAttr.
SQL_ATTR_KEYSET_SIZE	

Attribute	ValuePtr contents
(ODBC 2.0)	It is not supported by the driver.
SQL_ATTR_MAX_LENGTH (ODBC 1.0)	It is not supported by the driver.
SQL_ATTR_MAX_ROWS (ODBC 1.0)	• SQLULEN: It is the value which corresponds to the maximum number of rows which are returned from SELECT statement. If *ValuePtr is equal to 0, the driver returns all rows. The attribute is intended to reduce the network traffic. Notionally it is applied when the result set is generated, it limits the result set of the first ValuePtr row. If the number of rows in the result set are bigger than ValuePtr, then the result set is reduced. SQL_ATTR_MAX_ROWS includes those which are applied to all result set of the statement and returned by the catalog function. SQL_ATTR_MAX_ROWS sets the maximum number of the cursor rows. If SQL_ATTR_MAX_ROWS can not be guaranteed to be correctly implemented (if the result set size limit can not be implemented in the data source), then the driver should not imitate SQL_ATTR_MAX_ROWS operation for SQLFetch or SQLFetchScroll. The driver defines whether to apply SQL_ATTR_MAX_ROWS to the statements except for SELECT statement(such as the catalog functions). The attribute can be set in the open cursor. However, it does not immediately bring the effect. In this case, the driver returns SQLSTATE 01S02 (Option value changed) and sets the attribute to its original value.
SQL_ATTR_METADATA_ID (ODBC 3.0)	• SQLULEN: It is the value which determines how to treat the string arguments of the catalog functions. If it is SQL_TRUE, the catalog functions treat the string arguments as the identifiers. In this case, it is not case sensitive. The driver removes the trailing spaces and converts to uppercase for the string whose scope is not defined. The driver removes the leading and trailing spaces and literally takes the string between the delimiters for the string whose scope is defined. If one of the arguments is set to NULL pointer, the function returns SQL_ERROR and SQLSTATE HY009 (Invalid use of null pointer). If it is SQL_FALSE, the catalog functions do not treat the string arguments as the identifiers. In this case, it is case sensitive. The arguments may or may not include a string search pattern depending on the argument. The default value is SQL_FALSE. The list of values of TableType argument of SQLTables is not affected by the attribute. SQL_ATTR_METADATA_ID can be set on the connection level.(SQL_ATTR_METADAT

Attribute	ValuePtr contents
	A_ID and SQL_ATTR_ASYNC_ENABLE are unique and they are the statement attributes and connection attributes.) For more information, refer to **Arguments of Catalog Function**.
SQL_ATTR_NOSCAN (ODBC 1.0)	It is not supported by the driver.
SQL_ATTR_PARAM_BIND_OFFSET_PTR (ODBC 3.0)	• SQLULEN*: It is the value which points to the offset to add a pointer to change the dynamic parameter binding. If the field is not NULL, the driver dereferences the pointer, and additionally it dereferences each value of the deferred fields in the descriptor record(SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, SQL_DESC_OCTET_LENGTH_PTR) and it uses the new pointer values when bound. It is set to NULL by default. The bind offset is always directly added to SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, SQL_DESC_OCTET_LENGTH_PTR fields. If the offset is changed to another value, the new value is directly added to the descriptor field value. The new offset will not be added to any previous offset value in the field. SQL_DESC_BIND TYPE field in APD header is set by setting this statement attribute.
SQL_ATTR_PARAM_BIND_TYPE (ODBC 3.0)	• SQLULEN: It is the value which refers to the binding direction used in the dynamic parameter. This field is set to SQL_PARAM_BIND_BY_COLUMN which selects the column-wise binding. (default value) The field is set to the instance of the buffer to be bound to the structure length or dynamic parameter set to select the row-wise binding. This length should include the space for all bound parameters and the structure padding, or the address of the binding parameter is increased to the specified length, then the result should be buffered to point to the beginning of the next parameter. The sizeof operation of ANSI C guarantees this behavior. SQL_DESC_BIND_TYPE field in APD header is set by setting this statement attribute.
SQL_ATTR_PARAM_OPERATION_PTR (ODBC 3.0)	• SQLUSMALLINT*: It is the value which points to the array of SQLSMALLINT values used to ignore the parameter while executing the SQL statement. Each value is SQL_PARAM_PROCEED(to execute the parameter) or SQL_PARAM_IGNORE(to ignore the parameter). The parameter set can be ignored during processing by setting the status value of the array pointed by SQL_DESC_ARRAY_STATUS_PTR in APD. The parameter set is processed when the status value is SQL_PARAM_PROCEED or the array elements are not set. The statement attribute can be set to a NULL pointer and the driver does not return the status value of the parameter. The attribute can be set at any time but the new valu

Attribute	ValuePtr contents
	e is not used until SQLExecDirect or SQLExecute is called. The attribute is ignored if bound parameter does not exist. SQL_DESC_ARRAY_STATUS_PTR field in APD header is set by setting this statement a ttribute.
SQL_ATTR_PARAM_STATUS_PTR (ODBC 3.0)	• SQLUSMALLINT*: It is the value points to SQLUSMALLINT array which includes t he status information value of each row parameter value after calling SQLExecut e or SQLExecDirect. The field is required only when PARAMSET_SIZE is bigger th an 1. The status value can include other values. • SQL_PARAM_SUCCESS: SQL statement is successfully executed for the paramete r set • SQL_PARAM_SUCCESS_WITH_INFO: SQL statement is successfully executed for t he parameter set but the warning information exists in the diagnostic data struct ure. • SQL_PARAM_ERROR: An error occurs when processing the parameter set. The a dditional error information is in the diagnostic data structure. • SQL_PARAM_UNUSED: The parameter set is not used because some previous pa rameter set caused an error which interrupts the processing, or the parameter se t of the array specified by SQL_ATTR_PARAM_OPERATION_PTR is set to SQL_PA RAM_IGNORE. • SQL_PARAM_DIAG_UNAVAILABLE: The driver treats the parameter array as a un iform unit because the error information level is not generated. The statement attribute can be set to a NULL pointer and the driver does not return th e status value of the parameter. The attribute can always be set. But the new value is not used until SQLExecDirect or SQLExecute is called. The attribute may affect the ope ration that the parameter outputs in the driver. SQL_DESC_ARRAY_STATUS_PTR field of IPD header is set by setting this statement at tribute.
SQL_ATTR_PARAMS_PROCESSED PTR (ODBC 3.0)	• SQLULEN*: It is the record field pointing to the buffer in which to return the nu mber of parameter sets processed and it includes the error set. If it is a NULL poi nter, any number will not be returned. SQL_DESC_ROWS_PROCESSED_PTR field of IPD is set by setting this statement attrib ute. If SQLExecDirect or SQLExecute which will fill the buffer specified in this attribute doe s not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, the buffer content is not de fined.

Attribute	ValuePtr contents
SQL_ATTR_PARAMSET_SIZE (ODBC 3.0)	• SQLULEN: It is the value which specifies the number of each parameter values. If SQL_ATTR_PARAMSET_SIZE is bigger than 1, then SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, SQL_DESC_OCTET_LENGTH_PTR of APD points to the array. Each array constant is equal to the field value. If binding parameter does not exist, this attribute is ignored. SQL_DESC_ARRAY_SIZE field of APD header is set by setting the statement attribute.
SQL_ATTR_QUERY_TIMEOUT (ODBC 1.0)	• SQLULEN: It is the value which value in seconds which wait for the SQL statement to execute before returing to the application. If ValuePtr is 0(default value), the timeout does not occur. If the specified timeout value exceeds the maximum value of the data source or it is smaller than the minimum value, then SQLSetStmtAttr replaces the value and returns SQLSTATE 01S02 (Option value changed). Even if SELECT statement is timed out, SQLCloseCursor does not need to be called when reuse the statement. It is valid to set the query timeout of the statement attribute for both synchronous and asynchronous mode.
SQL_ATTR_RETRIEVE_DATA (ODBC 2.0)	It is not supported by the driver.
SQL_ATTR_ROW_ARRAY_SIZE (ODBC 3.0)	• SQLULEN: It is the value which points to the number of rows returned by calling each SQLFetch or SQLFetchScroll. It is also the number of rows of the bookmark array used in a bulk bookmark operation in SQLBulkOperations. The default value is 1. If the specified row set size exceeds the row set size supported by the data source, the driver replaces the value and returns SQLSTATE 01S02 (option value changed). SQL_DESC_ARRAY_SIZE field of ARD header is set by setting this statement attribute.
SQL_ATTR_ROW_BIND_OFFSET_PTR (ODBC 3.0)	• SQLULEN: It is the value which points to the offset added to pointers to change binding of the column data. If the field is not NULL, the driver dereferences the pointer and adds the value dereferenced to each field of the descriptor record (SQL_DESC_DATA_PTR, SSQL_DESC_INDICATOR_PTR, SQL_DESC_OCTET_LENGTH_PTR), and it uses the new pointer value when binding occurs. The default value is NULL. SQL_DESC_BIND_OFFSET_PTR field of ARD header is set by setting the statement attribute.
	• SQLULEN: It is the value which sets the binding direction when SQLFetch or SQL

Attribute	ValuePtr contents
SQL_ATTR_ROW_BIND_TYPE (ODBC 1.0)	FetchScroll is called in the related statement. If SQL_BIND_BY_COLUMN is set, the column-wise binding is selected. If the structure or result column sets the length for an instance of the buffer bound, then the row-wise binding is selected. If the length is specified and the address of the all columns and bound columns are increased to the specified length, then all columns should include enough space for the strucure or the buffer padding. The behavior is guaranteed when sizeof operator is used with the structure or union in ANSI C. The column-wise binding is the default bindng direction of SQLFetch and SQLFetchScroll. SQL_DESC_BIND_TYPE field of ARD header is set by setting this statement attribute.
SQL_ATTR_ROW_NUMBER (ODBC 2.0)	• SQLULEN: It is the current row order in the entire result set. If the number of the current rows can not be determined or the the current row does not exist, then the driver returns 0. The attribute can be retrieved by calling SQLGetStmtAttr. It can not be set by callingSQLSetStmtAttr.
SQL_ATTR_ROW_OPERATION_PTR (ODBC 3.0)	It is not supported by the driver.
SQL_ATTR_ROW_STATUS_PTR (ODBC 3.0)	• SQLUSMALLINT*: It is the value which points to SQLUSMALLINT array containing the status values after calling SQLFetch or SQLFetchScroll. The array has the elements as many as the number of rows in the row set. The statement attribute can be set to a NULL pointer, and in this case the driver does not return the row status value. The attribute can be set at any time but the new value is not used until the next SQLBulkOperations, SQLFetch, SQLFetchScroll or SQLSetPos is called. SQL_DESC_ARRAY_STATUS_PTR field of IRD header is set by setting the statement attribute. The attribute is mapped to rgbRowStatus array of SQLExtendedFetch in ODBC 2.x driver.
SQL_ATTR_ROWS_FETCHED_PTR (ODBC 3.0)	• SQLULEN*: It points to the buffer which returns the number of collected rows after calling SQLFetch or SQLFetchScroll. The number of rows are determined by calling SQLSetPos by setting SQL_REFRESH in operation argument, or determined by processing the bulk operation via SQLBulkOperations. The number of rows include the error rows. SQL_DESC_ROWS_PROCESSED_PTR field of IRD header is set by setting the statemen

Attribute	ValuePtr contents
	t attribute. If calling SQLFetch or SQLFetchScroll which fills in the buffer pointed by the attribute does not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, the buffer content is no t defined.
SQL_ATTR_SIMULATE_CU RSOR (ODBC 2.0)	It is not supported by the driver.
SQL_ATTR_USE_BOOKMA RKS (ODBC 2.0)	It is not supported by the driver.

SQLGetStmtOption

Conformance

Introduced version: ODBC 1.0
Standards compliance: It is not available.

Overview

In ODBC 3.x, SQLGetStmtOption function is replaced with SQLGetStmtAttr function.
For more information, refer to **SQLGetStmtAttr**.

SQLGetSuitableGroupID

Conformance

Standards compliance: It is not available.

Overview

SQLGetSuitableGroupID returns the ID of the cluster group which is the most suitable for the prepared st atement to perform in cluster system.

Syntax

```
SQLRETURN SQLGetSuitableGroupID(
    SQLHSTMT     StatementHandle,
    SQLINTEGER * GroupIDPtr );
```

Arguments

ConnectionHandle

[Input] It is the connection handle.

GroupIDPtr

[Output] It is the ID of the cluster group.

Returns

SQL_SUCCESS, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
08S01	Communication l ink failure	Before the function processing is completed, the connection between the dr iver and the data source is failed.
HY010	Function sequenc e error	This function is called before SQLPrepare. This function should be called only when the connection property SQL_ATT R_LOCALITY_AWARE_TRANSACTION is set.

Description

SQLGetSuitableGroupID can be called only when the connection property SQL_ATTR_LOCALITY_AWARE_TRANSACTION is set and the statement is prepared.
SQLGetSuitableGroupID returns the ID of the cluster group which is the most suitable for the preparedstatement to perform in cluster system by using the current value of the parameter marker when the parameter exists in the statement. If the cluster group which is the most suitable for the prepared statement to perform can not be determined, then it returns SQL_INVALID_GROUP_ID(-1).

SQLGetTypeInfo

Conformance

Introduced version: ODBC 1.0
Standards compliance: ISO 92

Overview

SQLGetTypeInfo returns the information of data type supported by the data source. The driver returns inf ormation in the form of SQL result set. Data types are used for Data Definition Language (DDL).

Syntax

```
SQLRETURN SQLGetTypeInfo(
    SQLHSTMT      StatementHandle,
    SQLSMALLINT   DataType);
```

Arguments

StatementHandle

[Input] It is the statement handle for the result set.

DataType

[Input] It is the SQL data type. It is data type or the specified driver SQL data type. SQL_ALL_TYPES specifies that the information about all data types should be returned.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01000	General Warning	It is the driver-specific informational message. (The function returns SQL _SUCCESS_WITH_INFO.)
		It is temporarily replaced with the similar value because the specified sta tement attribute is invalid due to implementation working condition. Th

SQLSTATE	Error	Description
01S02	Option value changed	e replaced value is valid for StatementHandle until the cursor is closed. The updatable statement attributes are as follows. SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ATTR_KEYSET_SIZE, SQL_ATTR_MAX_LENGTH, SQL_ATTR_MAX_ROWS, SQL_ATTR_QUERY_TIMEOUT, SQL_ATTR_SIMULATE_CURSOR. (The function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure	Before the function processing is completed, the connection between t he driver and the data source is failed.
24000	Invalid cursor state	The cursor is open in StatementHandle, and SQLFetch or SQLFetchScroll is called. If SQLFetch or SQLFetchScroll returns SQL_NO_DATA, the driver returns this error. If SQLFetch or SQLFetchScroll does not return SQL_NO_DATA, the driver manager returns this error. The result set is open in StatementHandle, but SQLFetch or SQLFetchScr oll is not called.
40001	Serailization failure	The transaction is rolled back due to a resource deadlock of other trans actions.
40003	Statement completion unk nown	The related connection fails during the function execution and the statu s of the transaction is not able to be checked.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY004	Invalid SQL data type	The value which is specified in DataType argument is neither ODBC SQL data type identifier supported by the driver nor the driver-specific data t ype identifier.
HY008	Operation canceled	Asynchronous processing for StatementHandle is available and SQLCan cel or SQLCancelHandle is called on StatementHandle before the functi on is called and completed. Then this function is called again on Statem entHandle. SQLCancel or SQLCancelHandle is called on StatementHandle from oth er thread in the multithreaded application before the function is called and completed.
HY010	Function sequence error	The asynchronously executing function is called for the connection han dle related to StatementHandle, but it is still being asynchronously exec uted when SQLGetStmtAttr is called. The asynchronously executing function is called for the StatementHandl e, it is still being asynchronously executed when the function is called. SQLExecute, SQLExecDirect, SQLBulkOperation or SQLSetPos is called f

SQLSTATE	Error	Description
		or StatementHandle, and SQL_NEED_DATA is returned. This function is called before data is sent for all data-at-execution parameters or columns.
HY013	Memory management error	The size of buffer used as an argument is smaller than 0, or it can not access the memory.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	For more information about the suspended state, refer to **SQLEndTran**.
HYC00	Optional field not implemented	The combination for the current setting of SQL_ATTR_CONCURRENCY and SQL_ATTR_CURSOR_TYPE statement attributes is not supported by the driver or by the data source. SQL_ATTR_USE_BOOKMARKS statement attribute is not set in SQL_UB_VARIABLE, and SQL_ATTR_CURSOR_TYPE statement attribute is set to the cursor type for the bookmarkwhich is not supported by the driver.
HYT00	Timeout expired	Before the data source returns the result set, the query timeout period is expired. The timeout can be set via SQL_ATTR_QUERY_TIMEOUT of SQLSetStmtAttr.
HYT01	Connection timeout expired	The connection is expired before the data source responds to the request. The timeout can be set via SQL_ATTR_CONNECTION_TIMEOUT of SQLSetStmtAttr.
IM001	Driver does not support this function	The driver which is related to DescriptorHandle does not support the function.
IM017	Polling is disabled in asynchronous notification mode	Whenever using the notification model, polling can not be used.
Im018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	If the previous function call for the handle returns SQL_STILL_EXECUTING and if the notification mode is activated, then SQLCompleteAsync should be called for the handle to do post processing and complete the operation.

Description

SQLGetTypeInfo returns the same result as the standard result set, and it is sorted to be closely mapped to the data types corresponding to the DATA_TYPE, ODBC SQL data type. The data types defined by the data source takes precedence over user defined data types. Therefore, the sorting order can be generalized in the ascending order and whose first order is DATA TYPE and then TYPE_NAME though the order is not always consistent.

For example, if the data source defines an INTEGER and COUNTER data types and COUNTER is automatically increased, and the user defined data type WHOLENUM is defined, then INTEGER, WHOLENUM and C

OUNTER will be returned in order. The result is because WHOLENUM is closely mapped to SQL_INTEGER and ODBC SQL data type. On the other hand, even though the automatic increasing data types are suppo rted by the data source, they are not mapped closely with ODBC SQL data types.

If DataType argument is valid for ODBC version supported by the driver but the driver does not support it, then an empty result set is returned.

The following column names are changed in ODBC 3.x. The column name change does not affect the co mpatibility with the previous version because the application is bound by the column number.

ODBC 2.0 column	ODBC 3.x column
PRECISION	COLUMN_SIZE
MONEY	FIXED_PREC_SCALE
AUTO_INCREMENT	AUTO_UNIQUE_VALUE

In ODBC 3.x, the following columns are added to the result set returned by SQLGetTypeInfo.

- SQL_DATA_TYPE
- INTERVAL_PRECISION
- SQL_DATETIME_SUB
- NUM_PREC_RADIX

The following table lists the columns in the result set. The additional row after column 19 (INTERVAL_PRE CISION) is defined by the driver. The applications should get access to a specific driver column by perform ing the countdown from the end of the result set instead of specifying an explicit ordinal position.

SQLGetTypeInfo may not return for all data types. For example, the driver may not return the us er defined data type. The application can use the valid data types regardless of returning SQLGe tTypeInfo. Data types returned by SQLGetTypeInfo are supported by the data source. They are in tended to be used in Data Definition Language (DDL). The driver can return the data types used in the result set other than the data types that returned by SQLGetTypeInfo. The driver may use the data types which are not supported by the data source in generating the result set for the c atalog function.

Column name	Column number	Data type	Comment
TYPE_NAM E (ODBC 2.0)	1	Varchar not NULL	It is data source-dependent data type name. For example, it is "CHAR()" , "VARCHAR()", "MONEY", "LONG VARBINARY", or "CHAR ( ) FOR BIT DATA". The application should use this name in CREATE TABLE and AL TER TABLE statements.
DATA_TYPE	2	Smallint not	It is SQL data type. It can be the ODBC SQL data type or driver specific SQL data. For DATETIME or INTERVAL data type, the column returns th

Column name	Column number	Data type	Comment
(ODBC 2.0)		NULL	e simple data type (such as SQL_TYPE_TIME or SQL_INTERVAL_YEAR_TO_MOUNT).
COLUMN_SIZE (ODBC 2.0)	3	Integer	It is the maximum column size for the data type supported by the server. For numeric data, it is the maximum precision. For string data, it returns a length of characters. For datetime data types, it returns a length of represented characters. For interval data, it returns the length of characters in the character representation of the interval literal. NULL is returned for data types whose column size is not applicable.
LITERAL_PREFIX (ODBC 2.0)	4	Varchar	A character or string is used as a prefix. For example, a single quote (') is for a character, 0x, or binary data type. NULL is returned for a data type which can not be used as a prefix.
LITERAL_SUFFIX (ODBC 2.0)	5	Varchar	A character or string is used as the termination character. For example, a single mark (') is for the character data type. NULL is returned for data types that can not be used as a suffix.
CREATE_PARAMS (ODBC 2.0)	6	Varchar	It is a keyword list corresponding to each parameter (separated by commas) which is specified by the application in parentheses when the name which is returned to TYPE_NAME is used. Keywords in the list can be length, precision, or scale. They are listed as the order in which the grammar uses keywords. For example, CREATE PARAMS for DECIMAL can be "precision, scale"". CRATE_PARAMS for VARCHAR can be the same "length". If parameter does not exist for defining data type, NULL is returned (e.g. INTERGER). The driver provides CREATE_PARAMS text in the language of the country/region.
NULLABLE (ODBC 2.0)	7	Smallint not NULL	It is whether the data type accepts NULL. SQL_NO_NULLS does not accept NULL as a data type. SQL_NULLABLE does accept NULL as a data type. SQL_NULLABLE_UNKNOWN do not know whether the column accepts NULL.
CASE_SENSITIVE (ODBC 2.0)	8	Smallint not NULL	It is whether the character data type is case-sensitive in sorting and comparison. It is SQL_TRUE if the data type is a character data type and it is case-sensitive. It is SQL_FALSE if the data type is not a character data type or it is case-insensitive.
SEARCHABLE (ODBC 2.0)	9	Smallint not NULL	It is how the data type is used in a WHERE clause. If a column can not be used in WHERE clause, it is SQL_PRED_NONE. (It is as same as the value of SQL_UNSEARCHABLE in ODBC 2.x.) If a column can be used only with LIKE condition in WHERE clause, it is SQL_PRED_CHAR. (It is as same as the value of SQL_LIKE_ONLY in ODBC 2.x.) If a column can be used with all comparison operators except LIKE condition in WHERE clause, it is SQL_PRED_BASIC. (It is as same as the val

Column name	Column number	Data type	Comment
			ue of SQL_ALL_EXCEPT_LIKE in ODBC 2.x.) If a column can be used with all comparison operators in WHERE claus e, it is SQL_SEARCHABLE.
UNSIGNED_ ATTRIBUTE (ODBC 2.0)	10	Smallint	It is whether the data type is signed. If a data type is unsigned, it is SQL_TRUE. If a data type is signed, it is SQL_FALSE. If the attribute can not be used in data type or it is not a numeric data t ype NULL is returned.
FIXED_PREC _SCALE (OD BC 2.0)	11	Smallint not NULL	It is whether the data type (specific data source) has a predefined fixed precision and scale. If the data type has a predefined fixed precision and scale, it is SQL_TR UE. If the data type does not have predefined fixed precision and scale, it is SQL_FALSE.
AUTO_UNIQ UE_VALUE (ODBC 2.0)	12	Smallint	It is whether data type is automatically increased. If the data type is automatically increased, it is SQL_TRUE. If the data type is not automatically increased, it is SQL_FALSE If the attribute can not be used in data type or it is not a numeric data t ype, NULL is returned. The application can insert the value in the column having this attribute, but can not typically update the value of the column. When insertion o ccurs in the automatically increased column, a unique value is inserted i n the column. The increment is not defined but is data source-specific. The application should not assume that the auto-increment column sta rts at a specific point or it is increased by a certain value.
LOCAL_TYP E_NAME (O DBC 2.0)	13	Varchar	It is a localized version of the data type name dependent on the data s ource. If the localized name is not supported by the data source, NULL i s returned. The name is used for display only, like as a dialog box.
MINIMUM_ SCALE (ODB C 2.0)	14	Smallint	It is the minimum scale of the data type on the data source. If the data type is a fixed Scale, MINIMUM_SCALE and MAXIMUM_SCALE column s have this value. For example, SQL_TYPE_TIMESTAMP column has the fixed scale for fractional seconds. NULL is returned if scale can not be u sed.
MAXIMUM_ SCALE (ODB C 2.0)	15	Smallint	It is the maximum scale of the data type on the data source. If scale ca n not be used, NULL is returned. If the maximum scale is not separately defined on the data source, but it is defined as same as the maximum precision, then the column has the same value as COLUMN_SIZE.
SQL_DATA_ TYPE (ODBC 3.0)	16	Smallint NO T NULL	It is the value of SQL data type which appears in SQL_DESC_TYPE field of the descriptor. The column is as same as DATA_TYPE column except for INTERVAL and DATETIME data types. SQL_DATE_TYPE field in the r esult set returns SQL_INTERVAL or SQL_DATETIME for INTERVAL and D ATETIME data types, and SQL_DATETIME_SUB field returns the sub-co

Column name	Column number	Data type	Comment
			de for INTERVAL or DATETIME data types.
SQL_DATETIME_SUB (ODBC 3.0)	17	Smallint	If a value of SQL_DATE_TYPE is SQL_DATETIME or SQL_INTERVAL, this column has the sub code of DATETIME/ INTERVAL. For other data types, the field is NULL. For INTERVAL or DATETIME data types, SQL_DATE_TYPE of the result set returns SQL_INTERVAL or SQL_DATETIME, SQL_DATETIME_SUB field returns the sub code for INTERVAL or DATETIME data type.
NUM_PREC_RADIX (ODBC 3.0)	18	Integer	If the data type is an approximate numeric type, the column has the value 2 to indicate that the COLUMN_SIZE specifies the number of bits. For the exact numeric type, the column has the value 10 to indicate that COLUMN_SIZE specifies decimal values. Otherwise, this column is NULL.
INTERVAL_PRECISION (ODBC 3.0)	19	Smallint	For INTERVAL data type, the column has the value of "INTERVAL leading precision". Otherwise, the column has NULL.

Attribute information can be applied to a particular column in the result set or the data type. SQLGetTypeInfo returns information about the attributes related to a data type. SQLColAttribute returns information about the attributes related to a column in the result set.

SQLMoreResults

Conformance

Introduced version: ODBC 1.0
Standards compliance: ODBC

Overview

SQLMoreResults checks that more results are available in the statements including SELECT, UPDATE, INSERT, or DELETE statement, and if so, the processing for the results is initialized.

Syntax

```
SQLRETURN SQLMoreResults(
    SQLHSTMT     StatementHandle);
```

Arguments

StatementHandle

[Input] It is the statement handle.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_NO_DATA, SQL_ERROR, SQL_INVALID_HANDLE, SQL_PARAM_DATA_AVAILABLE

Diagnosis

SQLSTATE	Error	Description
01000	General Warning	It is the driver-specific informational message. (The function returns SQL_SUCCESS_WITH_INFO.)
01S02	Option value changed	The value of the statement attribute is changed while the batch is being processed. (The function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure	Before the function processing is completed, the connection between the driver and the data source is failed.
40001	Serailization failure	The transaction is rolled back due to a resource deadlock of other transactions.

SQLSTATE	Error	Description
40003	Statement completion unknown	The related connection is failed during the function execution, and the status of transaction can not be checked.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY008	Operation canceled	Asynchronous processing for StatementHandle is available and SQLCancel or SQLCancelHandle is called on StatementHandle before the function is called and completed. Then this function is called again on StatementHandle. SQLCancel or SQLCancelHandle is called on StatementHandle from other thread in the multithreaded application before the function is called and completed.
HY010	Function sequence error	The asynchronously executing function is called for the connection handle related to StatementHandle, but it is still being asynchronously executed when SQLMoreResultsr is called. The asynchronously executing function is called for the StatementHandle, it is still being asynchronously executed when the function is called. SQLExecute, SQLExecDirect, SQLBulkOperation or SQLSetPos is called for StatementHandle, and SQL_NEED_DATA is returned. This function is called before data is sent for all data-at-execution parameters or columns.
HY013	Memory management error	The size of buffer used as an argument is smaller than 0, or it can not access the memory.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	For more information about the suspended state, refer to **SQLEndTran**.
HYT01	Connection timeout expired	The connection is expired before the data source responds to the request. The timeout can be set via SQL_ATTR_CONNECTION_TIMEOUT of SQLSetStmtAttr.
IM001	Driver does not support this function	The driver which is related to DescriptorHandle does not support the function.
IM017	Polling is disabled in asynchronous notification mode	Whenever using the notification model, polling can not be used.
Im018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	If the previous function call for the handle returns SQL_STILL_EXECUTING and if the notification mode is activated, then SQLCompleteAsync should be called for the handle to do post processing and complete the operation.

Description

SELECT statement returns the result set. UPDATE, INSERT, and DELETE statements return the number of r ows affected.

If one of these statements are batched or an array of parameters are passed or in progress, then multiple result sets or the number of rows can be returned.

The batch is processed, and the application is positioned in the first result set. The application can call SQ LBindCol, SQLBulkOperations, SQLFetch, SQLGetData, SQLFetchScroll, SQLSetPos and all the meta data f unctions as if a single result set exists for the first or subsequent result sets. When SQLMoreResults is exec uted as the first result set, the application calls SQLMoreResults to move to the next result set. If another r esult set or its number is available, SQLMoreResults returns SQL_SUCCESS and initializes the result set or performs the aggregate for the additional operations. If any count row generation statement appears bet ween the result set generation statements, the row generation statement can be sent to SQLMoreResults call. If SQLMoreResults is called for UPDATE, INSERT, or DELETE statement, the application can call SQLR owCount.

If a current result set with unfetched rows exists, SQLMoreResults discards the result set and generates th e next result set or aggregates it. If all result sets are processed, SQLMoreResults returns SQL_NO_DATA. For some drivers, the output parameter and the return value can not be used until all result sets and row aggregation are processed. In this case, the output parameter and return value can be used after SQLMor eResults returned SQL_NO_DATA.

All bindings made for the previous result set still remains valid. If the column structure is different from th e result set, then calling SQLFetch or SQLFetchScroll can cause an error or truncation. To prevent it, the a pplication should call SQLBindCol for the explicit rebinding. The application can call SQLFreeStmt with SQ L_UNBIND option to release the binding for all column buffers.

The statement attribute values such as the cursor type, cursor concurrency, key set size, or maximum leng th can be changed while the application calls SQLMoreResults and performs the batch processing. SQLM oreResults can return SQL_SUCCESS_WITH_INFO and SQLSTATE 01S02 (Option value has changed).

If SQLCloseCursor or SQLFreeStmt is called with an SQL_CLOSE option, then the number of available row s such as all the result set and batch processing result is discarded. The statement handle returns one of t he assigned state and prepared state. When the batch is processed and the statement handle is executed, then SQLCancel is called to cancel the asynchronously executing function. If SQLCancel is executed succe ssfully, all asynchronous status results which are generated by the batch processing, and the number of r ows can be discarded. The statement handle returns one of the assigned state and prepared state.

If the batch or procedure of the statement mixes SELECT, UPDATE, INSERT, DELETE statements with othe r SQL statements, these statements does not affect SQLMoreReuslt.

If a searched update, insert or delete statement in batch statements does not affect to any row of the dat a source, SQLMoreResults returns SQL_SUCCESS. It is different from the case of when SQLExecDirect, SQ LExecute, or SQLParamData returns SQL_NO_DATA. If the application calls SQLRowCount to retrieve the number of rows after SQLMoreResults does not affect any row, then SQLRowCount may return SQL_NO_ DATA.

Availability of the Number of Rows

If the batch processing includes the statement for consecutive multiple row aggregate generation, the ro w aggregation can be rolled up to a single row. For example, if a batch includes five INSERT statements, t he particular data source can return the five individual rows. Some other data sources return a single row that represents the sum of the total five individual rows.

If a batch processing includes the combination of the result set generation and row aggregate generation statement, then the number of rows may not be available.

The driver operation for the availability of the number of rows is listed in SQL_BATCH_ROW_COUNT infor mation type used via SQLGetInfo call. For example, if the batch processing includes two INSERT statemen ts and SELECT statement followed by the other SELECT statement, then the following cases are available.

- The number of rows corresponding to the two INSERT statements can not be used. The first call of S QLMoreResults is located in the result set of the second SELECT statement.
- The number of rows corresponding to the two INSERT statements can be individually used.(SQLGetIn fo does not return SQL_BRC_ROLLED_UP bit for SQL_BATCH_ROW_COUNT information type.) The fi rst call of SQLMoreResults is located in the number of rows of the first INSERT statement, and the sec ond call is positioned in the number of rows of the second INSERT statement. The third call may be lo cated in result set of the second SELECT statement.
- The number of rows corresponding to the two INSERT statements is rolled up to a single available ro w.(Calling SQLGetInfo returns SQL_BRC_ROLLED_UP bit for SQL_BATCH_ROW_COUNT information t ype.) The first SQLMoreResults may be located overlapping with the number of rows, and the second SQLMoreResults may be located in the result set of the second SELECT statement.

A specific driver makes the number of rows available only for the explicit batch processing.

SQLNativeSql

It is not supported.

Conformance

Introduced version: ODBC 1.0
Standards compliance: ODBC

Overview

SQLNativeSql returns the SQL string modified by the driver. SQLNativeSql does not executeSQL statement.

Syntax

```
SQLRETURN SQLNativeSql(
    SQLHDBC        ConnectionHandle,
    SQLCHAR *      InStatementText,
    SQLINTEGER     TextLength1,
    SQLCHAR *      OutStatementText,
    SQLINTEGER     BufferLength,
    SQLINTEGER *   TextLength2Ptr);
```

SQLNumParams

Conformance

Introduced version: ODBC 1.0
Standards compliance: ISO 92

Overview

SQLNumParams returns the number of parameters in the SQL statement.

Syntax

```
SQLRETURN SQLNumParams(
    SQLHSTMT        StatementHandle,
    SQLSMALLINT *   ParameterCountPtr);
```

Arguments

StatementHandle

[Input] It is the statement handle.

ParameterCountPtr

[Output] It is the buffer pointer to which the number of parameters of the statement is to be returned.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01000	General warning	It is the driver-specific informational message. (The function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure	Before the function processing is completed, the connection between the driver and the data source is failed.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.

SQLSTATE	Error	Description
HY008	Operation canceled	Asynchronous processing for StatementHandle is available and SQLCancel or SQLCancelHandle is called on StatementHandle before the function is called and completed. Then this function is called again on StatementHandle. SQLCancel or SQLCancelHandle is called on StatementHandle from other thread in the multithreaded application before the function is called and completed.
HY010	Function sequence error	The function is called in StatementHandle before calling SQLPrepare or SQLExecDirect. The asynchronously executing function is called for the connection handle related to StatementHandle, but it is still being asynchronously executed when SQLNumParams is called. The asynchronously executing function is called for the StatementHandle, it is still being asynchronously executed when the function is called. SQLExecute, SQLExecDirect, SQLBulkOperation or SQLSetPos is called for StatementHandle, and SQL_NEED_DATA is returned. This function is called before data is sent for all data-at-execution parameters or columns.
HY013	Memory management error	The internal memory can not be accessed or the available memory size is small.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	For more information about the suspended state, refer to **SQLEndTran**.
HYT01	Connection timeout expired	The connection is expired before the data source responds to the request. The timeout can be set via SQL_ATTR_CONNECTION_TIMEOUT of SQLSetStmtAttr.
IM001	Driver does not support this function	The driver which is related to StatementHandle does not support the function.
IM017	Polling is disabled in asynchronous notification mode	Whenever using the notification model, polling can not be used.
IM018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	If the previous function call for the handle returns SQL_STILL_EXECUTING and if the notification mode is activated, then SQLCompleteAsync should be called for the handle to do post processing and complete the operation.

Description

SQLNumParams can be called only after SQLPrepare is called.

If the statement related to StatementHandle does not have any parameter, then SQLNumParams sets*Par ameterCountPtr to 0.

The number of parameters which are returned by SQLNumParams is as same as the value of SQL_DESC_ COUNT field of IPD.

SQLNumResultCols

Conformance

Introduced version: ODBC 1.0
Standards compliance: ISO 92

Overview

SQLNumResultCols returns the number of columns in the result set.

Syntax

```
SQLRETURN SQLNumResultCols(
    SQLHSTMT        StatementHandle,
    SQLSMALLINT *   ColumnCountPtr);
```

Arguments

StatementHandle

[Input] It is the statement handle.

ColumnCountPtr

[Output] It is the buffer pointer to which the number of columns in the result set is to be returned. The number does not include the bookmark column.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01000	General warning	It is the driver-specific informational message. (The function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure	Before the function processing is completed, the connection between the driver and the data source is failed.
HY000	General error	It is an error without any specific SQLSTATE.
	Memory allocation err	

SQLSTATE	Error	Description
HY001	or	It is a memory allocation error.
HY008	Operation canceled	Asynchronous processing for StatementHandle is available and SQLCancel or SQLCancelHandle is called on StatementHandle before the function is called and completed. Then this function is called again on StatementHandle. SQLCancel or SQLCancelHandle is called on StatementHandle from other thread in the multithreaded application before the function is called and completed.
HY010	Function sequence error	The function is called in StatementHandle, before SQLPrepare or SQLExecDirect is called. The asynchronously executing function is called for the connection handle related to StatementHandle, but it is still being asynchronously executed when SQLNumResultsCols is called. The asynchronously executing function is called for the StatementHandle, it is still being asynchronously executed when the function is called. SQLExecute, SQLExecDirect, SQLBulkOperation or SQLSetPos is called for StatementHandle, and SQL_NEED_DATA is returned. This function is called before data is sent for all data-at-execution parameters or columns.
HY013	Memory management error	The internal memory can not be accessed or the available memory size is small.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	For more information about the suspended state, refer to **SQLEndTran**.
HYT01	Connection timeout expired	The connection is expired before the data source responds to the request. The timeout can be set via SQL_ATTR_CONNECTION_TIMEOUT of SQLSetStmtAttr.
IM001	Driver does not support this function	The driver which is related to DescriptorHandle does not support the function.
IM017	Polling is disabled in asynchronous notification mode	Whenever using the notification model, polling can not be used.
IM018	SQLCompleteAsync has not been called to complete the previous asynchronous operati	If the previous function call for the handle returns SQL_STILL_EXECUTING and if the notification mode is activated, then SQLCompleteAsync should be called for the handle to do post processing and complete the operation.

SQLSTATE	Error	Description
	on on this handle.	

Description

SQLNumResultCols is successfully called only when the statement is in the prepared, executed, positioned state.

If the statement related to StatementHandle does not return the column, then SQLNumResultCols sets*ColumnCountPtr to 0.

The number of rows returned by SQLNumResultCols is as same as the value of SQL_DESC_COUNT field of IRD.

SQLParamData

Conformance

Introduced version: ODBC 1.0
Standards compliance: ISO 92

Overview

SQLParamData is used together with SQLPutData to provide the parameters at the statement execution ti me.

Syntax

```
SQLRETURN SQLParamData(
    SQLHSTMT       StatementHandle,
    SQLPOINTER *   ValuePtrPtr);
```

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NEED_DATA, SQL_NO_DATA, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE, SQL_PARAM_DATA_AVAILABLE

Diagnosis

SQLSTATE	Error	Description
01000	General warning	It is the driver-specific informational message. (The function returns SQL_SU CCESS_WITH_INFO.)
08S01	Communication l ink failure	Before the function processing is completed, the connection between the dr iver and the data source is failed.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocatio n error	It is a memory allocation error.
HY010	Function sequenc e error	The previous function is not SQLExecute nor is SQLExecDirect which returns SQL_NEED_DATA. The previous function is SQLParamData.
HY013	Memory manage ment error	The internal memory can not be accessed or the available memory size is sm all.

SQLSTATE	Error	Description
HYT01	Connection time out expired	The connection is expired before the data source responds to the request. The timeout can be set via SQL_ATTR_CONNECTION_TIMEOUT of SQLSetStmtAttr.

If SQLParamData is called to transfer the data for a parameter, then it can return SQLSTATE of SQLExecute or SQLExecDirect.

Description

The driver returns SQL_NEED_DATA when the application calls SQLExecute or SQLExecDirect which requires data-at-execution. The application calls SQLParamData to determine the data to be transferred. If the driver needs the parameter data, then the value of *ValuePtr entered by the application is returned. The application can use the value to determine the parameter data requested by the driver.

The application calls SQLPutData as many times as necessary to transfer data-at-execution parameter. The application calls SQLParamData again after all parameter data is transferred. When SQLParamData returns SQL_NEED_DATA again, the application should call SQLPutData to transfer the other parameter data again. If all parameter data are transferred, then SQLParamData returns SQL_SUCCESS or SQL_SUCCESS_WITH_INFO and the value of *ValuePtr is not defined and the SQL statement can be executed.

SQLParamOptions

Conformance

Introduced version: ODBC 1.0
Standards compliance: It is not available.

Overview

SQLParamOptions function in ODBC 2.0 is replace with **SQLSetStmtAttr** in ODBC 3.x.

SQLPrepare

Conformance

Introduced version: ODBC 1.0
Standards compliance: ISO 92

Overview

SQLPrepare prepares the SQL string for execution.

Syntax

```
SQLRETURN SQLPrepare(
    SQLHSTMT      StatementHandle,
    SQLCHAR *     StatementText,
    SQLINTEGER    TextLength);
```

Arguments

StatementHandle

[Input] It is the statement handle.

StatementText

[Input] It is the SQL text string.

TextLength

[Input] It is the length of *StatementText in characters.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01000	General warning	It is the driver-specific informational message. (The function returns SQL_SUCCESS_WITH_INFO.)
		It is temporarily replaced with the similar value because the specified statem

SQLSTATE	Error	Description
01S02	Option value changed	ent attribute is invalid due to implementation working condition. (SQLGetStmtAttr can be called to see which value is temporarily changed.) The replaced value is valid for StatementHandle until the cursor is closed, and it is changed to the previous value when the cursor is closed. The updatable statement attributes are as follows. SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ATTR_KEYSET_SIZE, SQL_ATTR_MAX_LENGTH, SQL_ATTR_MAX_ROWS, SQL_ATTR_QUERY_TIMEOUT, SQL_ATTR_SIMULATE_CURSOR. (The function returns SQL_SUCCESS_WITH_INFO.)
21S01	Insert value list does not match column list	INSERT statement is in *StatementText, and the number of values to be inserted do not match the derived table.
21S02	Degree of derived table does not match column list	CREATE view statement is in *StatementText, and the number of specified names are not as same as the derived table defined by the query specifications.
22018	Invalid charcter value for cast specification	The SQL statement which includes a string or parameter is in *StatementText, and the value is not compatible with the data type of related table column.
22019	Invalid escape character	StatementText argument includes LIKE predicate such as ESCAPE in WHERE clause, and a control character length of ESCAPE is not 1.
22025	Invalid escape sequence	StatementText argument includes "LIKE pattern value ESCAPE escape character" in WHERE clause, and the pattern value control character is neither "%" nor is "-".
24000	Invalid cursor state	The cursor is open in StatementHandle, and SQLFetch or SQLFetchScroll is called.
34000	Invalid cursor name	*StatementText includes the positioned DELETE or positioned UPDATE, and the cursor referenced by the prepared statement is not open.
3D000	Invalid catalog name	The catalog name specified in StatementText is not valid.
3F000	Invalid schema name	The schema name specified in StatementText is not valid.
42000	Syntax error or access violation	*StatementText includes an SQL statement which is not preparable or it includes a syntax error. *StatementText includes a user without required privileges in a statement.
42S01	Base table or view already exists	*StatementText includes CREATE TABLE or CREATE VIEW statement, and the specified table name or the view name already exists.
		*StatementText includes DROP TABLE or DROP VIEW statement, and the specified table name or view name does not exists. *StatementText includes ALTER TABLE statement, and the specified table name does not exist.

SQLSTATE	Error	Description
42S02	Base table or view not found	*StatementText includes CREATE VIEW statement, and the table name or view name defined does not exist in the query specifications. *StatementText includes CREATE INDEX statement, and the specified table name does not exist. *StatementText includes GRANT or REVOKE statement, and the specified table name or view name does not exist. *StatementText includes SELECT statement, and the table or view name specified does not exist. *StatementText includes DELETE, INSERT or UPDATE statement, and the specified table name does not exist. *StatementText includes CREATE TABLE statement, and the table (refers to other tables) whose constraint is specified does not exist.
42S11	Index already exists	CREATE INDEX statement is in *StatementText, and the specified INDEX name already exists.
42S12	Index not found	DROP INDEX statement is in *StatementText, and the specified INDEX name does not exist.
42S21	Column already exist	ALTER TABLE statement is in *StatementText, and the specified column in ADD clause is not unique or it identifies the existing column in the base table.
42S22	Column not found	CREATE INDEX statement is in *StatementText, and one or more column names do not exist in the specified column list. GRANT or REVOKE statement is in *StatementText, and the specified column name does not exist. SELECT, DELETE or UPDATE statement is in *StatementText, and the specified column name does not exist. CREATE TABLE statement is in *StatementText, the column whose constraint is specified does not exist.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY008	Operation canceled	Asynchronous processing for StatementHandle is available and SQLCancel or SQLCancelHandle is called on StatementHandle before the function is called and completed. Then, this function is called again on StatementHandle. SQLCancel or SQLCancelHandle is called on StatementHandle from another

SQLSTATE	Error	Description
		thread in the multithreaded application before the function is called and co mpleted.
HY009	Invalid use of null pointer	StatementText is a NULL pointer.
HY010	Function sequenc e error	The function is called in StatementHandle before calling SQLPrepare or SQL ExecDirect. The asynchronously executing function is called for the connection handle r elated to StatementHandle, but it is still being asynchronously executed whe n SQLPrepare is called. The asynchronously executing function is called for the StatementHandle, it is still being asynchronously executed when the function is called. SQLExecute, SQLExecDirect, SQLBulkOperation or SQLSetPos is called for St atementHandle, and SQL_NEED_DATA is returned. This function is called be fore data is sent for all data-at-execution parameters or columns.
HY013	Memory manage ment error	The internal memory can not be accessed or the available memory size is sm all.
HY090	Invalid string or b uffer length	TextLength is equal to or smaller than 0, or it is not as same as SQL_NTS.
HY117	Connection is sus pended due to u nknown transacti on state. Only dis connect and read -only functions ar e allowed.	For more information about the suspended state, refer to **SQLEndTran**.
HYC00	Optional feature not implemented	The concurrency set is not valid for the defined cursor type. SQL_ATTR_USE_BOOKMARKS statement attribute is set to SQL_UB_VARIAB LE, and SQL_ATTR_CURSOR_TYPE statement attribute is set to the cursor ty pe for the bookmark that is not supported by the driver.
HYT00	Timeout expired	The query timeout is expired before getting the result set from the data sour ce. The timeout can be set via SQL_ATTR_QUERY_TIMEOUT of SQLSetStmt Attr.
HYT01	Connection time out expired	The connection is expired before the data source responds to the request. T he timeout can be set via SQL_ATTR_CONNECTION_TIMEOUT of SQLSetStm tAttr
IM001	Driver does not s upport this functi on	The driver which is related to StatementHandle does not support the functio n.
	Polling is disable d in asynchronou	

SQLSTATE	Error	Description
IM017	s notification mo de	Whenever using the notification model, polling can not be used.
IM018	SQLCompleteAsy nc has not been called to complet e the previous as ynchronous oper ation on this han dle.	If the previous function call for the handle returns SQL_STILL_EXECUTING a nd if the notification mode is activated, then SQLCompleteAsync should be called for the handle to do post processing and complete the operation.

Description

The application calls SQLPrepare to send the SQL statements to the data source for preparation. The appli cation can include one or more parameter markers in the SQL statement. The application describes the q uestion mark (?) at an appropriate position in the SQL string to include the parameter marker.

If the application uses SQLPrepare to prepare, or uses SQLExecute to submit COMMIT or ROLLB ACK statement, then it is impossible to interoperate between DBMS products.

The driver can update the statement to use the SQL types used by the data source, and pass it to the data source for the preparation. Especially, the driver updates the extended bit string which is used to define t he SQL statement for a particular function. In the driver, the statement handle is similar to the statement i dentifier in the embedded SQL code. If the data source supports statement identifiers, the driver can send a statement identifier and parameter values to the data source.

After the statement is prepared, the application uses the statement handle to refer to the statement in th e later function calls. The prepared statement related to the statement handle can be executed again wit h calling SQLExecute until the application releases the statement by calling SQLFreeStmt with SQL_DROP option or the statement handle uses one of SQLPrepare, SQLExecDirect or catalog function(SQLColumns, SQLTables, etc.). Once the application prepares the statement, information about the format of the result set may be requested. For some implementations, after SQLPrepare, calling of SQLDescribeCol or SQLDes cribeParam is not as effective as calling after SQLExecute or SQLExecDirec.

The driver can not return an syntax error or access violation when the application calls SQLPrepare. The dr iver can handle all syntax errors and access violations, or it can handle only syntax errors, or it can not han dle all syntax errors nor does return access violations. The application should be able to handle these con ditions when calling a subsequent related functions (the subsequent functions such as SQLNumResultCol s, SQLDescribeCol, SQLColAttribute, SQLExecute).

According to the features of the driver and data source, parameter information (such as the data type) ca n be checked when the statement is prepared (when all parameters are bound) or is executed (when all p arameters are not bound).
The application should release the binding of all parameters previously applied to the SQL statement befo re preparing the new SQL statement in the same statement. It can prevent an error whose previous para meter information is applied to the new statement.

Committing a transaction by explicitly calling SQLEndTran or by working in autocommit mode, c an cause the data source to delete the access plans for all statements on a connection. For more information, refer to SQL_CURSOR_COMMIT_BEHAVIOR and SQL_CURSOR_ROLLBACK_BEHA VIOR information types in **SQLGetInfo**.

SQLPrimaryKeys

Conformance

Introduced version: ODBC 1.0
Standards compliance: ODBC

Overview

SQLPrimaryKeys returns the column name consists of the primary key of a table. The driver returns information as a result set. The function does not support returning the primary key from the multiple tables in a single call.

Syntax

```
SQLRETURN SQLPrimaryKeys(
    SQLHSTMT       StatementHandle,
    SQLCHAR *      CatalogName,
    SQLSMALLINT    NameLength1,
    SQLCHAR *      SchemaName,
    SQLSMALLINT    NameLength2,
    SQLCHAR *      TableName,
    SQLSMALLINT    NameLength3);
```

Arguments

StatementHandle

[Input] It is the statement handle.

CatalogName

[Input] It is the catalog name. If the driver supports the catalog only for a few tables, an empty string ("") indicates the table with no catalog. CatalogName can not include the string search pattern.

If SQL_ATTR_METADATA_ID is set to SQL_TRUE, CatalogName is treated as an identifier, and it is not case sensitive. If it is set to SQL_FALSE, CatalogName is treated as an ordinary argument literal, and it is case sensitive.

NameLength1

[Input] It is the length of *CatalogName in characters.

SchemaName

[Input] It is the schema name. If the driver supports the schema only for a few tables, an empty string ("") indicates the table with no schema. SchemaName can not include the string search pattern.

If SQL_ATTR_METADATA_ID is set to SQL_TRUE, SchemaName is treated as an identifier, and it is not case sensitive. If it is set to SQL_FALSE, SchemaName is treated as a pattern value string, and it is not case sensitive.

NameLength2

[Input] It is the length of *SchemaName in characters.

TableName

[Input] It is the table name. The argument can not be a null pointer. TableName can not include the string search pattern.

If SQL_ATTR_METADATA_ID is set to SQL_TRUE, TableName is treated as an identifier, and it is not case sensitive. If it is set to SQL_FALSE, TableName is treated as an ordinary literal, and it is not case sensitive.

NameLength3

[Input] It is the length of *TableName in characters.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01000	General warning	It is the driver-specific informational message. (The function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure	Before the function processing is completed, the connection between the driver and the data source is failed.
24000	Invalid cursor state	SQLFetch or SQLFetchScroll is called, and the cursor is open. SQLFetch or SQLFetchScroll is not called, but the cursor is open.
40001	Serialization failure	The transaction is rolled back due to a resource deadlock of other transactions.
40003	Statement completion unknown	The related connection fails during the function execution and the status of the transaction is not able to be checked.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
		Asynchronous processing for StatementHandle is available and SQLCancel or SQLCancelHandle is called on StatementHandle before the function is called a

SQLSTATE	Error	Description
HY008	Operation canceled	nd completed. Then this function is called again on StatementHandle. SQLCancel or SQLCancelHandle is called on StatementHandle from another t hread in the multithreaded application before this function is called and com pleted.
HY009	Invalid use of null poi nter	TableName argument is a null pointer. SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, and Catal ogName argument is a null pointer. SQLGetInfo with the SQL_CATALOG_NAME information type returns that cat alog names are supported. SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, and Sche maName argument is a null pointer.
HY010	Function sequence er ror	The asynchronously executing function is called for the connection handle rel ated to StatementHandle, and this function is still being executed when SQLP rimaryKeys is called. SQLExecute, SQLExecDirect, or SQLMoreResults is called for StatementHandle, and SQL_PARAM_DATA_AVAILABLE is returned. This function is called befor e the data is checked for all connected parameters. The asynchronously executing function is called for StatementHandle, and is s till being executed when this function is called. SQLExecute, SQLExecDirect, SQLBulkOperation or SQLSetPos is called for Stat ementHandle, and SQL_NEED_DATA is returned. This function is called befor e data is sent for all data-at-execution parameters or columns.
HY013	Memory manageme nt error	The internal memory can not be accessed or the available memory size is smal l.
HY090	Invalid string or buffe r length	A name length argument value is smaller than 0 but it is not as same as SQL_ NTS. The name argument is not a null pointer. A name length argument value is bigger than the maximum length correspon ding to the name.
HY117	Connection is suspen ded due to unknown transaction state. Onl y disconnect and rea d-only functions are allowed.	For more information about the suspended state, refer to **SQLEndTran**.
		The catalog is specified, but the driver or the data source does not support th e catalog.

SQLSTATE	Error	Description
HYC00	Optional feature not implemented	The schema is specified, but the driver or the data source does not support th e schema. The combination of current setting of SQL_ATTR_CONCURRENCY and SQL_A TTR_CURSOR_TYPE statement attributes is not supported by the driver or dat a source. SQL_ATTR_USE_BOOKMARKS statement attribute is set to SQL_UB_VARIABL E, and SQL_ATTR_CURSOR_TYPE statement attribute is set to the cursor type for the bookmark that is not supported by the driver.
HYT00	Timeout expired	The query timeout is expired before getting the result set from the data sourc e. The timeout can be set via SQL_ATTR_QUERY_TIMEOUT of SQLSetStmtAtt r.
HYT01	Connection timeout expired	The connection is expired before the data source responds to the request. Th e timeout can be set via SQL_ATTR_CONNECTION_TIMEOUT of SQLSetStmtA ttr.
IM001	Driver does not supp ort this function	The driver which is related to StatementHandle does not support the function.
IM017	Polling is disabled in asynchronous notific ation mode	Whenever using the notification model, polling can not be used.
IM018	SQLCompleteAsync h as not been called to complete the previou s asynchronous oper ation on this handle.	If the previous function call for the handle returns SQL_STILL_EXECUTING an d if the notification mode is activated, then SQLCompleteAsync should be call ed for the handle to do post processing and complete the operation.

Description

SQLPrimaryKeys returns the result as the standard result set sorted by TABLE_CAT, TABLE_SCHEM, TABLE _NAME, and KEY_SEQ. For more information about how to use this information, refer to **Using Catalog Data**.

The names in the following columns are changed in ODBC 3.x. The column name change does not affect compatibility with the previous version because the application binds by the column number.

ODBC 2.0 column	ODBC 3.x column
TABLE_QUALIFIER	TABLE_CAT
TABLE_OWNER	TABLE_SCHEM

SQLGetInfo is called together with SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_SCHEMA_NAME_LEN,

SQL_MAX_TABLE_NAME_LEN, and SQL_MAX_COLUMN_NAME_LEN options to determine the actual col umn length of TABLE_CAT, TABLE_SCHEM, TABLE_NAME, and COLUMN_NAME.

For more information about general use, arguments, and the returned data of the ODBC catalo g functions, refer to **Catalog Function**.

The following table is a column list in the result set. An additional column after the column 6 can be defin ed by the driver. The application should access the related column by counting down from the end in the result set rather than by explicitly specifying the position. For more information, refer to **Data Returning o f Catalog Function**.

Column name	Column number	Data type	Description
TABLE_CAT (ODBC 1.0)	1	Varchar	It is the primary key table catalog name. If it can not be used in the dat a source, it is NULL. If the driver supports the catalog only for some tabl es, such as when the drive retrieves the data from another DBMS, it ret urns an empty string ("") for the table which does not have a catalog.
TABLE_SCH EM (ODBC 1 .0)	2	Varchar	It is the primary key table schema name. If it can not be used in the dat a source, it is NULL. If the driver supports the schema only for some tab le, such as when the drive retrieves the data from another DBMS, it ret urns an empty string ("") for the table which does not have a schema.
TABLE_NA ME (ODBC 1 .0)	3	Varchar not NULL	It is the primary key table name.
COLUMN_N AME (ODBC 1.0)	4	Varchar not NULL	It is the primary key column name. The driver returns an empty string f or a column which does not have a name.
KEY_SEQ (ODBC 1.0)	5	Smallint not NULL	It is the column sequence number of the key (starting from 1).
PK_NAME (ODBC 2.0)	6	Varchar	It is the primary key name. If it can not be applied to the data source, it is NULL.

SQLProcedureColumns

Conformance

Introduced version: ODBC 1.0
Standards compliance: ODBC

Overview

SQLProcedureColumns returns the list of input output parameters and the columns which configure the r esult set of the specified procedure. The driver returns information as the result set for the specified state ment.

Syntax

```
SQLRETURN SQLProcedureColumns(
    SQLHSTMT      StatementHandle,
    SQLCHAR *     CatalogName,
    SQLSMALLINT   NameLength1,
    SQLCHAR *     SchemaName,
    SQLSMALLINT   NameLength2,
    SQLCHAR *     ProcName,
    SQLSMALLINT   NameLength3,
    SQLCHAR *     ColumnName,
    SQLSMALLINT   NameLength4);
```

Arguments

StatementHandle

[Input] It is the statement handle.

CatalogName

[Input] It is the name of the procedure catalog. If the driver does not support the catalog, it returns an empty string ("") and the procedure does not include catalog. CatalogName can not include the string search pattern.

If SQL_ATTR_METADATA_ID is set to SQL_TRUE, CatalogName is treated as an identifier, and it is not case sensitive. If it is set to SQL_FALSE, CatalogName is treated as an ordinary argument literal, and it is case sensitive. For more information, refer to Arguments of Catalog Function.

NameLength1

[Input] It is the length of *CatalogName in characters.

SchemaName

[Input] It is the name of procedure schema. It is the string search pattern for the schema name. If t he driver does not support the schema, it returns an empty string ("") and the procedure does not i nclude the schema. SchemaName can not include the string search pattern.

If SQL_ATTR_METADATA_ID is set to SQL_TRUE, SchemaName is treated as an identifier, and it is not case sensitive. If it is set to SQL_FALSE, SchemaName is treated as a pattern value string argum ent, and it is case sensitive.

NameLength2

[Input] It is the length of *SchemaName in characters.

ProcName

[Input] It is the procedure name. This argument can not be a null pointer. ProcName can not includ e the string search pattern.

If SQL_ATTR_METADATA_ID is set to SQL_TRUE, ProcName is treated as an identifier, and it is not case sensitive. If it is set to SQL_FALSE, ProcName is treated as an ordinary literal, and it is not case sensitive.

NameLength3

[Input] It is the length of *ProcName in characters.

ColumnName

[Input] It is the column name. This argument can not be a null pointer. ColumnName can not inclu de the string search pattern.

If SQL_ATTR_METADATA_ID is set to SQL_TRUE, ColumnName is treated as an identifier, and it is not case sensitive. If it is set to SQL_FALSE, ColumnName is treated as an ordinary literal, and it is n ot case sensitive.

NameLength4

[Input] It is the length of *ColumnName in characters.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01000	General warning	It is the driver-specific informational message. (The function returns SQL_SUC CESS_WITH_INFO.)

SQLSTATE	Error	Description
08S01	Communication link failure	Before the function processing is completed, the connection between the driver and the data source is failed.
24000	Invalid cursor state	The cursor is open in StatementHandle, and SQLFetch or SQLFetchScroll is called. If SQLFetch or SQLFetchScroll returns SQL_NO_DATA, the driver returns this error. If SQLFetch or SQLFetchScroll does not return SQL_NO_DATA, the driver manager returns this error. The result set is open in StatementHandle, but SQLFetch or SQLFetchScroll is not called.
40001	Serialization failure	The transaction is rolled back due to a resource deadlock of other transactions.
40003	Statement completion unknown	The related connection fails during the function execution and the status of the transaction is not able to be checked.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY009	Invalid use of null pointer	TableName argument is a null pointer. SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, and CatalogName argument is a null pointer. SQLGetInfo with the SQL_CATALOG_NAME information type returns that catalog names are supported. SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, and SchemaName, ProcName or ColumnName argument is a null pointer.
HY010	Function sequence error	The asynchronously executing function is called for the connection handle related to StatementHandle, and this function is still executing when SQLProcedureColumns is called. SQLExecute, SQLExecDirect, or SQLMoreResults is called for StatementHandle, and SQL_PARAM_DATA_AVAILABLE is returned. This function is called before the data is checked for all connected parameters. SQLExecute, SQLExecDirect, SQLBulkOperation or SQLSetPos is called for StatementHandle, and SQL_NEED_DATA is returned. This function is called before data is sent for all data-at-execution parameters or columns. The asynchronously executing function is called for StatementHandle, and is still being executed when this function is called.
HY090	Invalid string or buffe	A name length argument value is smaller than 0 but it is not as same as SQL_NTS.

SQLSTATE	Error	Description
	r length	A name length argument value is bigger than the maximum length correspon ding to the name.
HY117	Connection is suspen ded due to unknown transaction state. Onl y disconnect and rea d-only functions are allowed.	For more information about the suspended state, refer to **SQLEndTran**.
HYC00	Optional feature not implemented	The catalog is specified, but the driver or the data source does not support th e catalog. The schema is specified, but the driver or the data source does not support th e schema. The combination of current setting of SQL_ATTR_CONCURRENCY and SQL_A TTR_CURSOR_TYPE statement attributes is not supported by the driver or dat a source. SQL_ATTR_USE_BOOKMARKS statement attribute is set to SQL_UB_VARIABL E, and SQL_ATTR_CURSOR_TYPE statement attribute is set to the cursor type for the bookmark that is not supported by the driver.
HYT00	Timeout expired	The query timeout is expired before getting the result set from the data sourc e. The timeout can be set via SQL_ATTR_QUERY_TIMEOUT of SQLSetStmtAtt r.
HYT01	Connection timeout expired	The connection is expired before the data source responds to the request. Th e timeout can be set via SQL_ATTR_CONNECTION_TIMEOUT of SQLSetStmtA ttr.
IM001	Driver does not supp ort this function	The driver which is related to StatementHandle does not support the function.
IM017	Polling is disabled in asynchronous notific ation mode	Whenever using the notification model, polling can not be used.
IM018	SQLCompleteAsync h as not been called to complete the previou s asynchronous oper ation on this handle.	If the previous function call for the handle returns SQL_STILL_EXECUTING an d if the notification mode is activated, then SQLCompleteAsync should be call ed for the handle to do post processing and complete the operation.

Description

SQLProcedureColumns returns the list of input output parameters and the columns which configure the r esult set of the specified procedure. The driver returns information as the result set for the specified state ment.

SQLProcedureColumns returns the results as a standard result set, sorted in an order of PROCEDURE_CAT, PROCEDURE_SCHEM, PROCEDURE_NAME, COLUMN TYPE. Column names are returned in an order of e ach parameter name (in call order) and the name of each column in the result set returned by the proced ure.

SQLGetInfo is called together with SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_SCHEMA_NAME, SQL_ MAX_PROCEDURE_NAME_LEN options in an application to determine the actual column length of PROC EDURE_CAT, PROCEDURE_SCHEM, PROCEDURE_NAME, COLUMN_NAME.

The names in the following columns are changed in ODBC 3.x. The column name change does not affect compatibility with the previous version because the application binds by the column number.

ODBC 2.0 column	ODBC 3.x column
PROCEDURE_QUALIFIER	PROCEDURE_CAT
PROCEDURE_OWNER	PROCEDURE_SCHEM
PRECISION	COLUMN_SIZE
LENGTH	BUFFER_LENGTH
SACLE	DECIMAL_DIGITS
RADIX	NUM_PREC_RADIX

In ODBC 3.x, the following columns are added to the result set returned by SQLProcedureColumns.

- COLUMN_DEF
- DATETIME_CODE
- CHAR_OCTET_LENGTH
- ORDINAL_POSITION
- IS_NULLABLE

The following table lists the columns in the result set. The additional row after column 19 (IS_NULLABLE) is defined by the driver. The applications should get access to a specific driver column by performing the c ountdown from the end of the result set instead of specifying an explicit ordinal position.

Column name	Column number	Data type	Description
PROCEDUR E_CAT (ODB	1	Varchar	It is the procedure catalog name. If it can not be used in the data sourc e, it is NULL. If the driver supports the catalog only for some tables, suc

Column name	Column number	Data type	Description
C 2.0)			h as when the drive retrieves the data from another DBMS, it returns a n empty string ("") for the table which does not have a catalog.
PROCEDURE_SCHEM (ODBC 2.0)	2	Varchar	It is the procedure schema name. If it can not be used in the data sourc e, it is NULL. If the driver supports the schema only for some table, suc h as when the drive retrieves the data from another DBMS, it returns a n empty string ("") for the table which does not have a schema.
PROCEDURE_NAME (ODBC 2.0)	3	Varchar not NULL	It is the procedure name. If the procedure does not have a name, then i t returns an empty string.
COLUMN_NAME (ODBC 2.0)	4	Varchar not NULL	It is the procedure column name. If the procedure does not have a nam e, then the driver returns an empty string.
COLUMN_TYPE (ODBC 2.0)	5	Smallint not NULL	It defines the procedure types. • SQL_PARAM_TYPE_UNKNOWN: The procedure column type is no t known. (ODBC 1.0) • SQL_PARAM_INPUT: The procedure column is an input parameter. (ODBC 1.0) • SQL_PARAM_INPUT_OUTPUT: The procedure column is an input/ output parameter. (ODBC 1.0) • SQL_PARAM_OUTPUT: The procedure column is an output param eter. (ODBC 2.0) • SQL_RETURN_VALUES: The procedure column is a return value of the procedure. (ODBC 2.0) • SQL_RESULT_COL: The procedure column is a column of result set. (ODBC 1.0)
DATA_TYPE (ODBC 1.0)	6	Smallint not NULL	It is SQL data type. For datetime and interval data type, this column ret urns a concise data type such as SQL_TYPE_DATE, SQL_INTERVAL_YEA R_TO_MONTH.
TYPE_NAME (ODBC 2.0)	7	Varchar not NULL	It is the name of a data source dependent data type. For example, it is CHAR(), VARCHAR(), MONEY, LONG VARBINARY or CHAR ( ) FOR BIT DATA. An application should use this name in CREATE TABLE and ALT ER TABLE statement.
COLUMN_SIZE (ODBC 2. 0)	8	Integer	It is the maximum column size of the data type supported by the server. The maximum precision is returned for the numeric data type, string da ta type returns return the character length, DATATIME data type return s the length of expressed character, INTERVAL data type returns the ch aracter length of literal INTERVAL character, and the datatype to which the column size is not applicable returns NULL.
BUFFER_LENGTH (ODBC 1.0)	9	Integer	It is the byte length transferred from SQLGetData or SQLFetch operatio n when SQL_C_DEFAULT is specified. The size of numeric data can be different from that of the data stored in the data source. For a string or a binary data, this value is as same as COLUMN_SIZE column.

Column name	Column number	Data type	Description
DECIMAL_DIGITS (ODBC 1.0)	10	Smallint	It is the practional digit in a column of the data source. If the decimal place of the data type can not be applicable, it returns NULL.
NUM_PREC_RADIX (ODBC 2.0)	11	Smallint	It is 2 or 10 for a numeric data type. If it is 2, COLUMN_SIZE and DECIMAL_DIGITS are number of bits allowed for a column. If it is 10, COLUMN_SIZE and DECIMAL_DIGITS are number of digits allowed for a column. The data type to which NUM_PREC_RADIX can not be applicable returns NULL.
NULLABLE (ODBC 2.0)	12	Smallint not NULL	It is whether the data type allows NULL value. If it is SQL_NULLABLE, the data type allows NULL value. If it is SQL_NO_NULLS, the data type does not allow NULL value. If it is SQL_NULLABLE_UNKNOWN, it is unknown if the column allows NULL value.
REMAKRS (ODBC 2.0)	13	Varchar	It is a description about the procedure column.
COLUMN_DEF (ODBC 3.0)	14	Varchar	It is the default value of a column. If this value is enclosed in double quotes, then this column should be interpreted as a string.
SQL_DATA_TYPE (ODBC 3.0)	15	Smallint not NULL	It is the SQL data type value appears in the SQL_DESC_TYPE field of a descriptor. This column is as same as DATA_TYPE column, except for INTERVAL and DATETIME data type. For INTERVAL and DATETIME data types, SQL_DATE_TYPE field in the result set returns SQL_INTERVAL or SQL_DATETIME, and SQL_DATETIME_SUB field returns the subcode for INTERVAL or DATETIME data type.
SQL_DATETIME_SUB (ODBC 3.0)	16	Smallint	It is the subtype code ofdatetime and interval data types. It returns NULL for other data types.
CHAR_OCTET_LENGTH (ODBC 3.0)	17	Integer	It is the maximum length in byte of character or binary data type column. It returns NULL for other data types.
ORDINAL_POSITION (ODBC 3.0	18	Integer not NULL	It is the column location in a table.
IS_NULLABLE (ODBC 3.0)	19	Varchar	• "YES": A column can include NULL. • "NO": A column can not include NULL. • It returns a string whose length is 0 when it is unknown whether to allow NULL.

SQLProcedures

Conformance

Introduced version: ODBC 1.0
Standards compliance: ODBC

Overview

SQLProcedures returns the list of procedure names stored in the specified data source. Procedure is a general term used to describe a named object which can be called by using the executable object or input and output parameter.

Syntax

```
SQLRETURN SQLProcedures(
    SQLHSTMT       StatementHandle,
    SQLCHAR *      CatalogName,
    SQLSMALLINT    NameLength1,
    SQLCHAR *      SchemaName,
    SQLSMALLINT    NameLength2,
    SQLCHAR *      ProcName,
    SQLSMALLINT    NameLength3);
```

Arguments

StatementHandle

[Input] It is the statement handle.

CatalogName

[Input] It is the name of the procedure catalog. If the driver does not support the catalog, it returns an empty string ("") and the procedure does not include catalog. CatalogName can not include the string search pattern.

If SQL_ATTR_METADATA_ID is set to SQL_TRUE, CatalogName is treated as an identifier, and it is not case sensitive. If it is set to SQL_FALSE, CatalogName is treated as an ordinary argument literal, and it is case sensitive. For more information, refer to **Arguments of Catalog Function**.

NameLength1

[Input] It is the length of *CatalogName in characters.

SchemaName

[Input] It is the name of procedure schema. The string search pattern for the schema name. If the d river does not support the schema, it returns an empty string ("") and the procedure does not inclu de the schema. SchemaName can not include the string search pattern.

If SQL_ATTR_METADATA_ID is set to SQL_TRUE, SchemaName is treated as an identifier, and it is not case sensitive. If it is set to SQL_FALSE, SchemaName is treated as a pattern value string argum ent, and it is case sensitive.

NameLength2

[Input] It is the length of *SchemaName in characters.

ProcName

[Input] It is the procedure name. This argument can not be a null pointer. ProcName can not includ e the string search pattern.

If SQL_ATTR_METADATA_ID is set to SQL_TRUE, ProcName is treated as an identifier, and it is not case sensitive. If it is set to SQL_FALSE, ProcName is treated as a pattern value string argument, an d it is case sensitive.

NameLength3

[Input] It is the length of *ProcName in characters

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDL E.

Diagnosis

SQLSTATE	Error	Description
01000	General warning	It is the driver-specific informational message. (The function returns SQL_SUC CESS_WITH_INFO.)
08S01	Communication link f ailure	Before the function processing is completed, the connection between the driv er and the data source is failed.
24000	Invalid cursor state	The cursor is open in StatementHandle, and SQLFetch or SQLFetchScroll is call ed. If SQLFetch or SQLFetchScroll returns SQL_NO_DATA, the driver returns this e rror. If SQLFetch or SQLFetchScroll does not return SQL_NO_DATA, the driver manager returns this error. The result set is open in StatementHandle, but SQLFetch or SQLFetchScroll is not called.
		The transaction is rolled back due to a resource deadlock of other transaction

SQLSTATE	Error	Description
40001	Serialization failure	s.
40003	Statement completio n unknown	The related connection fails during the function execution and the status of t he transaction is not able to be checked.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation er ror	It is a memory allocation error.
HY009	Invalid use of null poi nter	TableName argument is a null pointer. SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, and Catal ogName argument is a null pointer. SQLGetInfo with the SQL_CATALOG_NAME information type returns that cat alog names are supported. SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, and Sche maName, ProcName or ColumnName argument is a null pointer.
HY010	Function sequence er ror	The asynchronously executing function is called for the connection handle rel ated to StatementHandle, and this function is still being executed when SQLP rocedureColumns is called. SQLExecute, SQLExecDirect, or SQLMoreResults is called for StatementHandle, and SQL_PARAM_DATA_AVAILABLE is returned. This function is called befor e the data is checked for all connected parameters. The asynchronously executing function is called for StatementHandle, and is s till being executed when this function is called. SQLExecute, SQLExecDirect, SQLBulkOperation or SQLSetPos is called for Stat ementHandle, and SQL_NEED_DATA is returned. This function is called befor e data is sent for all data-at-execution parameters or columns.
HY013	Memory manageme nt error	The internal memory can not be accessed or the available memory size is smal l.
HY090	Invalid string or buffe r length	A name length argument value is smaller than 0 but it is not as same as SQL_ NTS. A name length argument value is bigger than the maximum length correspon ding to the name.
HY117	Connection is suspen ded due to unknown transaction state. Onl y disconnect and rea d-only functions are allowed.	For more information about the suspended state, refer to **SQLEndTran**.
		The catalog is specified, but the driver or the data source does not support th

SQLSTATE	Error	Description
HYC00	Optional feature not implemented	e catalog. The schema is specified, but the driver or the data source does not support th e schema. The combination of current setting of SQL_ATTR_CONCURRENCY and SQL_A TTR_CURSOR_TYPE statement attributes is not supported by the driver or dat a source. SQL_ATTR_USE_BOOKMARKS statement attribute is set to SQL_UB_VARIABL E, and SQL_ATTR_CURSOR_TYPE statement attribute is set to the cursor type for the bookmark that is not supported by the driver.
HYT00	Timeout expired	The query timeout is expired before getting the result set from the data sourc e. The timeout can be set via SQL_ATTR_QUERY_TIMEOUT of SQLSetStmtAtt r.
HYT01	Connection timeout expired	The connection is expired before the data source responds to the request. Th e timeout can be set via SQL_ATTR_CONNECTION_TIMEOUT of SQLSetStmtA ttr.
IM001	Driver does not supp ort this function	The driver which is related to StatementHandle does not support the function.
IM017	Polling is disabled in asynchronous notific ation mode	Whenever using the notification model, polling can not be used.
IM018	SQLCompleteAsync h as not been called to complete the previou s asynchronous oper ation on this handle.	If the previous function call for the handle returns SQL_STILL_EXECUTING an d if the notification mode is activated, then SQLCompleteAsync should be call ed for the handle to do post processing and complete the operation.

Description

SQLProcedures lists all procedures in the requested range. A user may or may not have privileges to execu te these procedures. Refer SQL_ACCESSIBLE_PROCEDURES of SQLGetInfo to check accessibility. If a user selects the procedure which can not be executed, the application should be able to handle the situation. SQLProcedures returns the standard result set, and it is sorted in an order of PROCEDURE_CAT, PROCED URE_SCHEMA, PROCEDURE_NAME

The following column names are changed in ODBC 3.x. The column name change does not affect the co mpatibility with the previous version because the application binds by the column number.

ODBC 2.0 column	ODBC 3.x column

ODBC 2.0 column	ODBC 3.x column
PROCEDURE_QUALIFIER	PROCEDURE_CAT
PROCEDURE_OWNER	PROCEDURE_SCHEM

SQLGetInfo is called together with SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_SCHEMA_NAME, SQL_MAX_PROCEDURE_NAME_LEN options in an application to determine the actual column length of PROCEDURE_CAT, PROCEDURE_SCHEM, PROCEDURE_NAME, COLUMN_NAME

The following table lists the columns in the result set. The additional row after column 8 (PROCEDURE_TYPE) is defined by the driver. The applications should get access to a specific driver column by performing the countdown from the end of the result set instead of specifying an explicit ordinal position.

Column name	Column number	Data type	Comments
PROCEDURE_CAT (ODBC 2.0)	1	Varchar	It is the procedure catalog name. If it can not be used in the data source, it is NULL. If the driver supports the catalog only for some tables, such as when the drive retrieves the data from another DBMS, it returns an empty string ("") for the table which does not have a catalog.
PROCEDURE_SCHEM (ODBC 2.0)	2	Varchar	It is the procedure schema identifier. If it can not be used in the data source, it is NULL. If the driver supports the schema only for some table, such as when the drive retrieves the data from another DBMS, it returns an empty string ("") for the table which does not have a schema.
PROCEDURE_NAME (ODBC 2.0)	3	Varchar not null	It is the procedure identifier.
REMARK (ODBC 2.0)	4	Varchar	It is the description on the procedure.
PROCEDURE_TYPE (ODBC 2.0)	5	Smallint	It defines a procedure type. • SQL_PT_UNKNOWN: Where the procedure returns the value or not is unknown. • SQL_PT_PROCEDURE: The returned object is a procedure. In other words, a return value does not exist. • SQL_PT_FUNCTION: The returned object is a function. In other words, a return value exists.

SQLPutData

Conformance

Introduced version: ODBC 1.0
Standards compliance: ISO 92

Overview

SQLPutData allows the application to transfer a parameter to the driver or transfer the data to a column at statement execution time.
The function can be used to transfer a character or binary data value in parts to a column with a character, binary, or data source specific data type (For example, the parameter of SQL_LONGVARBINARY or SQL_LONGVARCHAR types).
Even if the default driver does not support unicode, SQLPutData supports the binding to Unicode C data types.

Syntax

```
SQLRETURN SQLPutData(
    SQLHSTMT     StatementHandle,
    SQLPOINTER   DataPtr,
    SQLLEN       StrLen_or_Ind);
```

Arguments

StatementHandle

[Input] It is the statement handle.

DataPtr

[Input] It is the buffer pointer which includess the actual data for the parameter or column. The data should be the C data types specified by ValueType argument of SQLBindParameter (parameter data) or TargetType argument of SQLBindCol (column data).

StrLen_or_Ind

[Input] It is the length of *DataPtr. It specifies the amount of data transferred to SQLPutData call. The amount of data can be varied depending on each call to the given parameter or the column. StrLen_or_Ind is ignored if one of the following conditions is not satisfied.

- StrLen_or_Ind is SQL_NTS, SQL_NULL_DATA, or SQL_DEFAULT_PARAM.

• The C data type that is specified in SQLBindParameter or SQLBindCol is SQL_C_CHAR or SQL_C_BINARY.

• The C data type is SQL_C_DEFAULT, and the default C data type for the specified SQL data type is SQL_C_CHAR or SQL_C_BINARY.

For all other types of C data, if StrLen_or_Ind is not SQL_NULL_DATA or SQL_DEFAULT_PARAM, the driver determines that the size of *DataPtr is the size of C data type specified in ValueType or TargetType, and transfers the entire data value.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01000	General warning	It is the driver-specific informational message. (The function returns SQL_SUCCESS_WITH_INFO.)
01004	String data, right truncated	The string or binary data returned to the input/output or output parameters are truncated. If a string is truncated, the right part of the string is truncated. (The function returns SQL_SUCCESS_WITH_INFO.)
07006	Restricted data type attribute violation	The data value identified by the ValueType argument in SQLBindParameter for the bound parameter can not be converted to the data type identified by the ParameterType argument in SQLBindParameter.
07S01	Invalid use of default parameter	The parameter value set in SQLBindParameter is SQL_DEFAULT_PARAM, and the corresponded parameter does not have the default value.
08S01	Communication link failure	Before the function processing is completed, the connection between the driver and the data source is failed.
22001	String data, right truncation	The string or binary data is truncated. SQL_NEED_LONG_DATA_LEN information type is "Y" in SQLGetInfo, and more data is transferred for a long parameter than is specified with StrLen_or_IndPtr argument in SQLBindParameter. SQL_NEED_LONG_DATA_LEN information type is "Y" in SQLGetInfo, and more data is transferred for a long column than is specified in the buffer length corresponding to the data rows which are added or updated with SQLBulkOperation or updated with SQLSetPos.
22003	Numeric value out of range	The bound numeric parameter or the data which is transferred to the column causes the truncation for the numeric value when the related table column is allocated. The numeric value which is returned for the input/output parameter or output parameter is truncated.

SQLSTATE	Error	Description
22007	Invalid datetime format	The data sent for a parameter or column which is bound to a date, time, or timestamp structure is invalid for each type. An input/output or output parameter is bound to a date, time, or timestamp C structure, and a value in the returned parameter is invalid for each type. (The function returns SQL_SUCCESS_WITH_INFO.)
22008	Datetime field overflow	DATETIME expression is not valid for the bound DATE, TIME or TIMESTAMP C structure.
22012	Division by zero	An arithmetic expression calculated for an input/output or output parameter is in divided by zero.
22015	Interval field overflow	The significant digits are lost due to the data transfer for an exact numeric or interval column or parameter to an interval SQL data type. The data transferred to one or more INTERVAL columns or parameters is converted into a numeric data type, but it can not be expressed as a numeric data type. The data transferred for the column or parameter data is assigned to INTERVAL SQL type, but it can not be expressed as the value of C type in INTERVAL SQL type. The significant digits of the data transferred for the exact number or the INTERVAL C column or parameter are lost. The data transferred for the column or parameter data is assigned to INTERVAL C type, but it can not be expressed as INTERVAL data structure.
22018	Invalid character value for cast specification	The character not represented as the character set of C buffer is included in the character column of the result set. The C type is the exact or approximate numeric, datetime, interval data type, and if the SQL type is the character data type, then the value of the column bound to the C type is invalid.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY008	Operation canceled	Asynchronous processing for StatementHandle is available and SQLCancel or SQLCancelHandle is called on StatementHandle before the function is called and completed. Then, this function is called again on StatementHandle. SQLCancel or SQLCancelHandle is called on StatementHandle from other thread in the multithreaded application before this function is called and completed.
HY009	Invalid use of null	DataPtr argument is a null pointer, and StrLen_or_Ind argument is neither S

SQLSTATE	Error	Description
	pointer	QL_DEFAULT_PARAM nor SQL_NULL_DATA.
HY010	Function sequence error	The previous function is not a call to SQLPutData or SQLParamData. The asynchronously executing function is called for the connection handle related to StatementHandle, but it is still being asynchronously executed when SQLPrimaryKeys is called. The asynchronously executing function is called for StatementHandle, but it is still being asynchronously executed when the function is called. SQLExecute, SQLExecDirect, SQLBulkOperation or SQLSetPos is called for StatementHandle, and SQL_NEED_DATA is returned. This function is called before data is sent for all data-at-execution parameters or columns.
HY013	Memory management error	The internal memory can not be accessed or the available memory size is small.
HY019	Non-character and non-binary data sent in pieces	SQLPutData is called for the parameter or column one or more times, and it is not used to transfer the character C data to a column with the character, the binary or the data source-specific data type, or to transfer the binary C data to a column with the character, the binary or the data source-specific data type.
HY020	Attempt to concatenate a null value	SQLPutData is called once or more after SQL_NEED_DATA is returned, and StrLen_or_Ind argument includes SQL_NULL_DATA or SQL_DEFAULT_PARAM in one of these calls.
HY090	Invalid string or buffer length	DataPtr argument is not a null pointer, and StrLen_or_Ind is not equal to SQL_NTS or SQL_NULL_DATA, or is smaller than than 0.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	For more information about the suspended state, refer to **SQLEndTran**.
HYT01	Connection timeout expired	The connection is expired before the data source responds to the request. The timeout can be set via SQL_ATTR_CONNECTION_TIMEOUT of SQLSetStmtAttr.
IM001	Driver does not support this function	The driver related to StatementHandle does not support the function.
IM017	Polling is disabled in asynchronous notification mode	Whenever using the notification model, polling can not be used.
	SQLCompleteAsy	

SQLSTATE	Error	Description
IM018	nc has not been called to complete the previous asynchronous operation on this handle.	If the previous function call for the handle returns SQL_STILL_EXECUTING and if the notification mode is activated, then SQLCompleteAsync should be called for the handle to do post processing and complete the operation.

Description

SQLPutData can be called to provide data-at-execution data for two uses. One is to use it as parameter data when calling SQLExecute or SQLExecDirect. The other is to use it as column data when updating or adding a row by calling SQLBulkOperations, or when updating a row by calling SQLSetPos.

When the application calls SQLParamData to determine which data should be transferred, the driver returns an indicator that the application can use to determine which parameter data to send or where column data can be found. The function also returns SQL_NEED_DATA which is an indicator to the application that it should call SQLPutData to transfer the data. In the DataPtr argument to SQLPutData, the application passes a pointer to the buffer including the actual data for the parameter or column.

When the driver returns SQL_SUCCESS for SQLPutData, the application calls SQLParamData again. For transferring more data, SQLParamData returns SQL_NEED_DATA, and the application calls SQLPutData again. If all data-at-execution data are transferred, SQL_SUCCESS is returned. And then the application calls SQLParamData again. If the driver returns SQL_NEED_DATA and another marker in *ValuePtrPtr, then it requests the data for other parameter or column and calls SQLPutData again. If the driver returns SQL_SUCCESS, then all data-at-execution data are transferred, and the SQL statement can be executed or SQLBulkOperations or SQLSetPos can be processed.

The application is allowed to use SQLPutData only when the character C data or binary C data is transferred to the character, binary or data source specific data type. If SQLPutData is called once or more under different conditions, SQL_ERROR and SQLSTATE HY019 are returned. (Non-character and non-binary data sent in pieces)

SQLRowCount

Conformance

Introduced version: ODBC 1.0
Standards compliance: ISO 92

Overview

SQLRowCount returns the number of rows whih are affected by UPDATE, INSERT, or DELETE statement. (SQL_ADD, SQL_UPDATE_BY_BOOKMARK or SQL_DELETE_BY_BOOKMARK operation in SQLBulkOperations. Or SQL_UPDATE or SQL_DELETE operation in SQLSetPos)

Syntax

```
SQLRETURN SQLRowCount(
    SQLHSTMT    StatementHandle,
    SQLLEN *    RowCountPtr);
```

Arguments

StatementHandle

[Input] It is the statement handle.

RowCountPtr

[Output] It is the buffer pointer to which the number of rows is to be returned. For UPDATE, INSERT, DELETE statements, for the SQL_ADD, SQL_UPDATE_BY_BOOKMARK, SQL_DELETE_BY_BOOKMARK operations in SQLBulkOperations, and for the SQL_UPDATE or SQL_DELETE operations in SQLSetPos, the value returned to *RowCountPtr is the number of rows affected by the request. Or, it is -1 if the number of affected rows is not available.

When SQLExecute, SQLExecDirect, SQLBulkOperations, SQLSetPos or SQLMoreResults is called, SQL_DIAG_ROW_COUNT field of the diagnostic data structure is set to the number of rows, and the number of rows are cached in a way that depends on the implementation. SQLRowCount returns the number of cashed rows. The number of cashed rows are valid until when the statement handle is set again in the prepared or assigned state, or when the statement is executed again or when SQLCloseCursor is called. If SQL_DIAG_ROW_COUNT field is set and the function is called, then SQL_DIAG_ROW_COUNT field is set to 0 by calling any function, and the return value by SQLRowCount can be different from the value in SQL_DIAG_ROW_COUNT field.

For other statements and functions, the driver can define the value returned in *RowCountPtr. For

example, some data source can return the number of rows returned by SELECT statement before r ows are patched.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01000	General warning	It is the driver-specific informational message. (The function returns SQL_S UCCESS_WITH_INFO.)
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY010	Function sequence error	The asynchronously executing function is called for the connection handle related to StatementHandle, and this function is still being executed when SQLRowCount is called. SQLExecute, SQLExecDirect, SQLMoreResults is called for StatementHandl e, and SQL_PARAM_DATA_AVAILABLE is returned. The function is called before data is retrieved for all streamed parameters. The function is called for StatementHandle, before SQLExecute, SQLExecD irect, SQLBulkOperations, or SQLSetPos is called. The asynchronously executing function is called for StatementHandle, and this function is still being executed when this function is called. SQLExecute, SQLExecDirect, SQLBulkOperation or SQLSetPos are called fo r StatementHandle, and SQL_NEED_DATA is returned. This function is call ed before data is sent for all data-at-execution parameters or columns.
HY013	Memory management er ror	The internal memory can not be accessed or the available memory size is s mall.
HY117	Connection is suspended due to unknown transac tion state. Only disconne ct and read-only functio ns are allowed.	For more information about the suspended state, refer to **SQLEndTran**.
HYT00	Timeout expired	The query timeout is expired before getting the entire result set from the data source. The timeout can be set via SQL_ATTR_QUERY_TIMEOUT of S QLSetStmtAttr.
HYT01	Connection timeout expi	The connection is expired before the data source responds to the request. The timeout can be set via SQL_ATTR_CONNECTION_TIMEOUT of SQLSet

SQLSTATE	Error	Description
	red	StmtAttr.
IM001	Driver does not support t his function	The driver which is related to StatementHandle does not support the func tion.

Description

If the last SQL statement executed for the statement handle is not UPDATE, INSERT, DELETE, or if an oper ation argument of the previous SQLBulkOperations call is not SQL_ADD, SQL_UPDATE_BY_BOOKMARK, SQL_DELETE_BY_BOOKMARK or if an operation argument of the previous SQLSetPos call is not SQL_UP DATE, SQL_DELETE, then the value of *RowCountPtr is defined by the driver.

SQLSetConnectAttr

Conformance

Introduced version: ODBC 3.0
Standards compliance: ISO 92

Overview

SQLSetConnectAttr sets the attribute which controls the connection.

Syntax

```
SQLRETURN SQLSetConnectAttr(
    SQLHDBC       ConnectionHandle,
    SQLINTEGER    Attribute,
    SQLPOINTER    ValuePtr,
    SQLINTEGER    StringLength);
```

Arguments

ConnectionHandle

[Input] It is the connection handle.

Attribute

[Input] It is the attribute for setting.

ValuePtr

[Input] It is the pointer of the value related to attribute. According to the value of attribute, ValueP tr can be an unsigned integer or it points to a null-termination string. The integer type of attribute argument may not be the fixed length data type. For more information, refer to the Description sec tion.

StringLength

[Input] If attribute is the ODBC-defined attribute and ValuePtr points to the string or binary buffer, then the argument should be the length of *ValuePtr. For string data, the argument should includ e the number of bytes of the string.

If attribute is the ODBC-defined attribute and ValuePtr is an integer, then StringLength is ignored.

If attribute is the driver-defined attribute, then the application indicates the attribute characteristics set by StringLength argument to the driver manager. StringLength may have one of the following v

alues:

• If ValuePtr is a string pointer, StringLength is the string length or SQL_NTS.

• If ValuePtr is a binary buffer pointer, the application stores the result of SQL_LEN_BINARY_ATTR (length) macro which is a negative value in StringLength.

• If ValuePtr is neither string pointer nor binary buffer pointer, StringLength must have SQL_IS_POINTER.

• If ValuePtr contains the fixed length value, StringLength is SQL_IS_INTEGER or SQL_IS_UINTEGER.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE, SQL_STILL_EXECUTING

Diagnosis

SQLSTATE	Error	Description
01000	General warning	It is the driver-specific informational message. (The function returns SQL_SUCCESS_WITH_INFO.)
01S02	Option value changed	The value specified in ValuePtr is not supported by the driver and it is replaced with a similar value. (The function returns SQL_SUCCESS_WITH_INFO.)
08002	Connection name in use	Attribute argument is SQL_ATTR_ODBC_CURSORS, and the driver is already connected to the data source.
08003	Connection not open	Attribute value is assigned to request an open connection, but ConnectionHandle is not in a connected state.
08S01	Communication link failure	Before the function processing is completed, the connection between the driver and the data source is failed.
24000	Invalid cursor state	Attribute argument is SQL_ATTR_CURRENT_CATALOG, and the result set is deferred.
25000	Illegal operation while in a local transaction	The local transaction is connected while attempting to connect the distributed transaction by setting of the connection attribute SQL_ATTR_ENLIST_IN_DTC. The connection to the distributed transaction is already enlisted. The connection to the distributed transaction is already enlisted, the local transaction is started by setting SQL_ATTR_AUTOCOMMIT to SQL_AUTOCOMMIT_OFF.
3D000	Invalid catalog name	Attribute argument is SQL_ATTR_CURRENT_CATALOG and the specified catalog name is not valid.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocatio	It is a memory allocation error.

SQLSTATE	Error	Description
	n error	
HY008	Operation cancel ed	The asynchronous processing is activated for ConnectionHandle. The SQLSe tConnectAttr function is called, and before it completes execution, the SQL CancelHandle function is called on the ConnectionHandle, and then the SQ LSetConnectAttr function is called again on the ConnectionHandle. Or, SQLSetConnectAttr function is called, and before it completes execution, SQLCancelHandle is called on the ConnectionHandle from a different thread in a multithread application.
HY009	Invalid use of null pointer	Attribute argument identifies the connection attribute which requires the st ring value, and ValuePtr argument is a null pointer.
HY010	Function sequenc e error	The asynchronously executing function is called for the statementHandle rel ated to ConnectionHandle, and this function is still being executed when SQ LSetConnectAttr is called. The asynchronously executing function is called for the ConnectionHandle, and this function is still being executed when this function is called. SQLExecute, SQLExecDirect, SQLMoreResults is called for one of Statement Handle related to ConnectionHandle, and SQL_PARAM_DATA_AVAILABLE i s returned. The function is called before data is retrieved for all streamed pa rameters. SQLExecute, SQLExecDirect, SQLBulkOperation or SQLSetPos are called for StatementHandle related to ConnectionHandle, and SQL_NEED_DATA is ret urned. This function is called before data is sent for all data-at-execution par ameters or columns. SQLBrowseConnect is called for ConnectionHandle, and SQL_NEED_DATA i s returned. The function is called before SQLBrowseConnect returns SQL_SU CCESS_WITH_INFO or SQL_SUCCESS.
HY011	Attribute cannot be set now	Attribute argument is SQL_ATTR_TXN_ISOLATION, and the transaction is op en.
HY013	Memory manage ment error	The internal memory can not be accessed or the available memory size is sm all.
HY024	Invalid attribute v alue	The specified Attribute value is assigned, and the invalid value is specified to ValuePtr. Attribute argument is SQL_ATTR_TRACEFILE or SQL_ATTR_TRANSLATE_LIB, and ValuePtr is an empty string.
HY090	Invalid string or b uffer length	ValuePtr is a string and StringLength argument is smaller than 0 but it is not SQL_NTS.
	Driver does not s upport connectio	The application tries to activate asynchronous execution with SQL_ATTR_AS

SQLSTATE	Error	Description
HY114	n-level asynchron ous function exe cution	YNC_DBC_FUNCTIONS_ENABLE when the driver does not support asynchro nous connection.
HY117	Connection is sus pended due to u nknown transacti on state. Only dis connect and read -only functions ar e allowed.	For more information about the suspended state, refer to **SQLEndTran**.
HY121	Cursor Library an d Driver-Aware P ooling cannot be enabled at the sa me time	It is not supported by the driver.
HYC00	Optional feature not implemented	The value specified for Attribute argument is valid for the ODBC connection or the statement attribute for the version supported by the driver, but it is n ot supported by the driver.
HYT01	Connection time out expired	The connection is expired before the data source responds to the request. T he timeout can be set via SQL_ATTR_CONNECTION_TIMEOUT of SQLSetStm tAttr.
IM001	Driver does not s upport this functi on	The driver which is related to ConnectionHandle does not support the functi on.
IM009	Unable to load tr anslation DLL	The driver can not load the transaction DDL specified for the connection. Th e error can be returned only when attribute is SQL_ATTR_TRANSLATE_LIB.
IM017	Polling is disable d in asynchronou s notification mo de	Whenever using the notification model, polling can not be used.
IM018	SQLCompleteAsy nc has not been called to complet e the previous as ynchronous oper ation on this han dle.	If the previous function call for the handle returns SQL_STILL_EXECUTING a nd if the notification mode is activated, then SQLCompleteAsync should be called for the handle to do post processing and complete the operation.
S1118	Driver does not s upport asynchron ous notification	SQL_ATTR_ASYNC_DBC_EVENT is set but the asynchronous notification is n ot supported by the driver.

Description

The application can call SQLSetConnectAttr at any time between the connection is assigned and released. All connections and the statement attributes set by the application for connection are maintained until SQLFreeHandle is called. For example, if an application calls SQLSetConnectAttr before connecting to a data source, the attribute is maintained even when SQLSetConnectAttr fails in the driver when the application connects to the data source. If the application sets the driver-specific attribute then the attribute is maintained even when the application connects to a different driver.

The feature to set the statement attribute at the connection level by SQLSetConnectAttr call is not used in ODBC 3.x. ODBC 3.x application should not set the statement attribute at the connection level. ODBC 3.x application can not set the statement attributes except for SQL_ATTR_METADATA_ID and SQL_ATTR_ASYNC_ENABLE attributes at the connection level. The two attributes are both the connection attribute and statement attribute and they can be set at the connection level or statement level. If ODBC 3.x driver operates with ODBC 2.x application which sets the ODBC 2.x statement option at the connection level, the ODBC 3.x driver is required to support it.

Some connection attributes can be set only before the connection is made, and some other attributes can be set only after the connection is made. The following table describes these connection attributes.

Attribute	Whether it is set before or after connection
SQL_ATTR_ACCESS_MODE	Either[1]
SQL_ATTR_ASYNC_DBC_EVENT	Either
SQL_ATTR_ASYNC_DBC_FUNCTIONS_ENABLE	Either[4]
SQL_ATTR_ASYNC_DBC_PCALLBACK	Either
SQL_ATTR_ASYNC_DBC_PCONTEXT	Either
SQL_ATTR_ASYNC_ENABLE	Either[2]
SQL_ATTR_AUTO_IPD	Either
SQL_ATTR_AUTOCOMMIT	Either[5]
SQL_ATTR_CONNECTION_DEAD	After
SQL_ATTR_CONNECTION_TIMEOUT	Either
SQL_ATTR_CURRENT_CATALOG	Either[1]
SQL_ATTR_DBC_INFO_TOKEN	After
SQL_ATTR_ENLIST_IN_DTC	After
SQL_ATTR_LOGIN_TIMEOUT	Before
SQL_ATTR_METADATA_ID	Either
SQL_ATTR_OLDPWD	Before
SQL_ATTR_ODBC_CURSORS	Before
SQL_ATTR_PACKET_SIZE	Before

Attribute	Whether it is set before or after connection
SQL_ATTR_QUIET_MODE	Either
SQL_ATTR_TRACE	Either
SQL_ATTR_TRACEFILE	Either
SQL_ATTR_TRANSLATE_LIB	After
SQL_ATTR_TRANSLATE_OPTION	After
SQL_ATTR_TXN_ISOLATION	Either[3]

[1]SQL_ATTR_ACCESS_MODE and SQL_ATTR_CURRENT_CATALOG can be set before and after the connection according to the driver. But the application which uses multiple drivers should be set before the connection because these changes are not supported after the connection depending on the driver.

[2]SQL_ATTR_ASYNC_ENABLE should be set before the statement is activated.

[3]SQL_ATTR_TXN_ISOLATION can be set only when there is not any open transaction on the connection. Some connection attributes is supported by replacing with the similar value if the data source does not support the value specified by *ValuePtr. In this case, the driver returns SQL_SUCCESS_WITH_INFO and SQLSTATE 01S02 (Option value changed). For example, if Attribute is SQL_ATTR_PACKET_SIZE and *ValuePtr exceeds the maximum packet size, the driver replaces it with the maximum packet size, the application calls SQLSetConnectAttr to check the replaced value.

[4] If SQL_ATTR_ASYNC_DBC_FUNCTIONS_ENABLE is set before the connection is open, the driver manager sets the driver attribute when the driver is loaded while SQLBrowseConnect, SQLConnect, or SQLDriverConnect is called. Before SQLBrowseConnect, SQLConnect, or SQLDriverConnect is called, the driver manager does not know which driver is connected and does not know whether the driver supports the asynchronous operation for connection. Therefore, the driver manager returns SQL_SUCCESS. However if the driver does not support the asynchronous operation for connection, then calling SQLBrowseConnect, SQLConnect, or SQLDriverConnect will fail.

[5] If SQL_ATTR_AUTOCOMMIT is set to FALSE and API returns SQL_ERROR, the application should call SQLEndTran (SQL_ROLLBACK) to guarantee the transaction consistency.

Information type in *ValuePtr buffer depends on the specified attribute. SQLSetConnectAttr accepts one of the null-termination character or integer value as attribute information. The character string indicated to by the ValuePtr argument of SQLSetConnectAttr has StringLength bytes length.

If the length is defined in the attribute, StringLength argument is ignored as is the case of all attributes introduced in ODBC 2.x or the previous version.

Attribute	ValuePtr contents
	It is an SQLUINTEGER value. SQL_MODE_READ_WRITE is the default value. SQL_MODE_READ_ONLY is used in the driver or data source as an indicator of which the conn

Attribute	ValuePtr contents
SQL_ATTR_ACCESS_MODE (ODBC 1.0)	ection is not requested to support SQL statement which causes update to occur. The mode can be used to optimize the proper lock strategy, transaction management or other area for the driver or the data source. The driver is not required to prevent such information which is sent to the data source. The behavior of the driver and data source when asked to process SQL statements which are not read-only during a read-only connection is implementation-defined.
SQL_ATTR_ASYNC_DBC_EVENT (ODBC 3.8)	It is not supported by the driver.
SQL_ATTR_ASYNC_DBC_FUNCTIONS_ENABLE (ODBC 3.8)	It is an SQLUINTERGER value which activates or deactivates asynchronous execution of selected functions on the connection handle. • SQL_ASYNC_DBC_ENABLE_ON: It activates an asynchronous operation for the function related to the specified connection. • SQL_ASYNC_DBC_ENABLE_OFF: It deactivates an asynchronous operation for the function related to the specified connection. SQL_ATTR_ASYNC_DBC_FUNCTIONS_ENABLE setting is always synchronous. (SQL_STILL_EXECUTING is not returned.) Asynchronous execution of the statement is activated by SQL_ATTR_ASYNC_ENABLE.
SQL_ATTR_ASYNC_DBC_PCALLBACK (ODBC 3.8)	It is not supported by the driver.
SQL_ATTR_ASYNC_DBC_PCONTEXT (ODBC 3.8)	It is not supported by the driver..
SQL_ATTR_ASYNC_ENABLE (ODBC 3.0)	It is not supported by the driver.
SQL_ATTR_AUTO_IPD (ODBC 3.0)	It is not supported by the driver.
SQL_ATTR_AUTOCOMMIT (ODBC 1.0)	It is an SQLUINTEGER value which specifies whether to use the auto-commit or manual commit mode. • SQL_AUTOCOMMIT_OFF: The driver uses the manual commit mode, and the application should explicitly commit or roll back the transaction with SQLEndTran. • SQL_AUTOCOMMIT_ON: The driver uses the auto-commit mode. Each statement is immediately executed and committed. All open transactions in the connection are committed when SQL_ATTR_AUTOCOMMIT is set to SQL_AUTOCOMMIT_ON. Some data sources remove the access plan at the time of when the statement is committed to the connection and close the cursor. It may occur in auto commit mode after each non-query statements are executed or after the cursor is closed for the query. For more information, refer to SQL_CURSOR_COMMIT_BEHAVIOR and SQL_CURSOR_ROLLBACK_BEHAVIOR information type of **SQLGetInfo**.

Attribute	ValuePtr contents
	The batch can be executed in auto commit mode in two ways. The entire batch can be treated as the automatic processing unit, or each statement in a batch can be treated as the automatic processing unit. A particular data source may support both of these ways or it may select one and support it. The driver defines the entire batch or each statement in the batch processing can be treated as the automatic processing unit.
SQL_ATTR_CONNECTION_DEAD(ODBC 3.5)	It is not supported by the driver.
SQL_ATTR_CONNECTION_TIMEOUT (ODBC 3.0)	It is not supported by the driver.
SQL_ATTR_CURRENT_CATALOG (ODBC 2.0)	It is not supported by the driver.
SQL_ATTR_DBC_INFO_TOKEN (ODBC 3.8	It is not supported by the driver.
SQL_ATTR_ENLIST_IN_DTC (ODBC 3.0)	It is not supported by the driver.
SQL_ATTR_LOGIN_TIMEOUT (ODBC 1.0)	It is SQLUINTERGER value corresponding to a wait time, in seconds, until the login request is completed before returning to the application. The default value depends on the driver. If ValuePtr is 0, the time limit is deactivated and the connection attempt will indefinitely wait. If the specified login request timeout exceeds the maximum login time limit, then the driver replaces the value and returns SQLSTATE 01S02 (Option value changed).
SQL_ATTR_METADATA_ID (ODBC 3.0)	It is SQLUINTEGER value which determines how to handle the string argument of the catalog function. The default value is SQL_FALSE. If it is SQL_TRUE, string argument of the catalog function is regarded as an identifier and it is not case-sensitive. For a non-delimited string, the driver removes all trailing spaces and changes the string to uppercase. For a delimited string, the driver removes leading or trailing spaces and still has the character between delimiters. If one of these arguments is a null pointer, the function returns SQL_ERROR and SQLSTATE HY009 (Invalid use of null pointer). If it is SQL_FALSE, string argument of the catalog function is not regarded as an identifier and it is case-sensitive. So it may be processed as a string pattern according to the argument, or not. TableType argument of SQLTables which has the value list is not affected by the attribute. SQL_ATTR_METADATA_ID can be set in the statement step. (It is the unique connection attribute in the statement attribute.)

Attribute	ValuePtr contents
	For more information, refer to **Arguments of Catalog Function**.
SQL_ATTR_OLDPWD	It is SQLPOINTER for the previous encrypted string. The value is write only, and it should be set prior to connecting to the server.
SQL_ATTR_ODBC_CURSORS (ODBC 2.0)	It is not supported by the driver.
SQL_ATTR_PACKET_SIZE (ODBC 2.0)	It is not supported by the driver.
SQL_ATTR_QUIET_MODE (ODBC 2.0)	It is not supported by the driver.
SQL_ATTR_TRACE (ODBC 1.0)	It is not supported by the driver.
SQL_ATTR_TRACEFILE (ODBC 1.0)	It is not supported by the driver.
SQL_ATTR_TRANSLATE_LIB (ODBC 1.0)	It is not supported by the driver.
SQL_ATTR_TRANSLATE_OPTION (ODBC 1.0)	It is not supported by the driver.
SQL_ATTR_TXN_ISOLATION (ODBC 1.0)	It is 32-bit bit mask for setting the transaction isolation level for the current connection. The application should call SQLEndTran by using this option before calling SQLSetConnectAttr to commit or roll back all open transactions on connection. The valid value for ValuePtr can be determined by calling SQLGetInfo with InfoType which is equal to SQL_TXN_ISOLATION_OPTIONS. For more information about the transaction isolation level, refer to SQL_DEFAULT_TXN_ISOLATION information type of **SQLGetInfo**.

[1] The function can be asynchronously called only when the descriptor is the implementation descriptor, but not the application descriptor.

SQLSetConnectOption

Conformance

Introduced version: ODBC 1.0
Standards compliance: It is not available.

Overview

ODBC 2.0 function SQLSetConnectOption is replaced with SQLSetConnectAttr In ODBC 3.x. For more inf ormation, refer to **SQLSetConnectAttr**.

SQLSetCursorName

Conformance

Introduced version: ODBC 1.0
Standards compliance: ISO 92

Overview

SQLSetCursorName connects the cursor name to the activated statement. If the application does not call SQLSetCursorName, then the driver generates the cursor name which is necessary to the SQL statement p rocessing.

Syntax

```
SQLRETURN SQLSetCursorName(
    SQLHSTMT      StatementHandle,
    SQLCHAR *     CursorName,
    SQLSMALLINT   NameLength);
```

Arguments

StatementHandle

[Input] It is the statement handle.

CursorName

[Input] It is the cursor name. A cursor name should not contain leading or trailing spaces for efficie nt processing, and if the cursor name includes a limited identifier, then the delimiter should be posi tioned on the first letter of the cursor name.

NameLength

[Input] It is the string length of *CursorName.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01000	General warning	It is the driver-specific informational message. (The function returns SQL_SUCCESS_WITH_INFO.)
01004	String data, right truncated	The cursor name length exceeds the maximum, and only the string of the maximum allowable number is used.
24000	Invalid cursor state	The statement corresponding to StatementHandle is already running or it is in use as the to a positioning cursor.
34000	Invalid cursor name	The cursor name specified in *CursorName exceeds the maximum value of the driver, or it is not valid because it is started with SQLCUR or SQL_CUR.
3C000	Duplicate cursor name	The name specified in *CursorName already exists.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY009	Invalid use of null pointer	CursorName argument is a NULL pointer.
HY010	Function sequence error	The asynchronously executing function is called for the connection handle related to StatementHandle, and this function is still being executed when SQLSetCursorName is called. SQLSetCursorName function is called and the asynchronously executing function is called for StatementHandle. SQLExecute, SQLExecDirect, SQLBulkOperation or SQLSetPos are called for StatementHandle, and SQL_NEED_DATA is returned. This function is called before data is sent for all data-at-execution parameters or columns.
HY013	Memory management error	The internal memory can not be accessed or the available memory size is small.
HY090	Invalid string or buffer length	NameLength argument is smaller than 0. (It is not SQL_NTS.)
HY117	Connection is suspended due to unknown tracsaction state. Only disconnect and read-only functions are allowed.	For more information about the suspended state, refer to **SQLEndTran**.
HYT01	Connection timeout expired	The connection timeout period is expired before the data source responds to the request. The timeout can be set through SQL_ATTR_CONNECTION_TIMEOUT of SQLSetConnectAttr.
IM001	Driver does not support this function	The driver does not support the function.

Description

The cursor name is used for positioning update statements and positioning delete statements. (e.g. UPD ATE table-name ... WHERE CURRENT OF cursor-name). If the application does not define the cursor nam e by calling SQLSetCursorName in the execution of query statement, then the driver generates the name which starts with SQL_CUR and does not exceed a length of 18 characters.

All cursor names should be unique within a connection. The maximum length of the cursor name is defin ed in the driver. For maximum interoperability, it is recommended that the application limit the cursor na me of more than 18 characters. In ODBC 3.x, if the cursor name is enclosed in double quotes ("), it is con sidered to be case sensitive and it is not allowed by SQL syntax or it can include the specially treated chara cters like space or reserved word. The case-sensitive cursor name should be enclosed with double quotes (") identifier.

The cursor name remains until the related statement is deleted by using SQLFreeHandle. SQLSetCursorNa me can be called to rename a cursor on a statement when the cursor is in an allocated or prepared state.

SQLSetDescField

Conformance

Introduced version: ODBC 3.0
Standards compliance: ISO 92

Overview

SQLSetDescField sets the value of a single field of the descriptor record.

Syntax

```
SQLRETURN SQLSetDescField(
    SQLHDESC      DescriptorHandle,
    SQLSMALLINT   RecNumber,
    SQLSMALLINT   FieldIdentifier,
    SQLPOINTER    ValuePtr,
    SQLINTEGER    BufferLength);
```

Arguments

DescriptorHandle

[Input] It is the descriptor handle.

RecNumber

[Input] It points to the descriptor record including a field which the application seeks to set. The de scriptor records are numbered from 0 and the record number 0 is the bookmark record. RecNumb er argument is ignored for the header field.

FieldIdentifier

[Input] It indicates the descriptor field to be set.

ValuePtr

[Input] It describes a buffer which contains the descriptor information or integer value. The data ty pe depends on the value of FieldIdentifier. If ValuePtr is an integer value, it can be considered to be 8 bytes (SQLLEN), 4 bytes (SQLINTEGER), 2 bytes (SQLSMALLINT) according to the value of FieldIde ntifier argument.

BufferLength

[Input] If FieldIdentifier is the ODBC defined field and ValuePtr points to the string or binary buffer, the argument should be the length of *ValuePTr. The argument should contain the number of byt es of the string for the string data.

If FieldIdentifier is the ODBC defined field and ValuePtr is an integer, BufferLength is ignored.

If FieldIdentifier is the driver defined field, the application describes the attribute features to the dri ver manager by setting BufferLength argument. BufferLength can have the following values.

- If ValuePtr is a string buffer pointer, BufferLength is the string length or SQL_NTS.
- If ValuePtr is a binary buffer pointer, the application stores the result of SQL_LEN_BINARY_ATTR (length) macro in BufferLengfth. BufferLength stores the negative value.
- If ValuePtr includes the fixed length value, BufferLength is one of SQL_IS_INTEGER, SQL_IS_UIN TEGER, SQL_IS_SMALLINT or SQL_IS_USMALLINT.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01000	Genaral warning	It is the driver-specific informational message. (The function returns S QL_SUCCESS_WITH_INFO.)
01S02	Option value changed	The driver does not support the value specified in *ValuePtr (if the val ue of ValuPtr is an integer.), or *ValuPtr is not valid on the implemen tation conditions, so the driver replaces it with a similar value. (The fu nction returns SQL_SUCCESS_WITH_INFO.)
07009	Invalid descriptor index	When FieldIdentifier argument is a record field, and RecNumber argu ment is 0, then DescriptorHandle argument refers to IPD handle. RecNumber argument is smaller than 0, and DescriptorHandle argum ent refers to ARD or APD. RecNumber argument is bigger than the maximum of the column or parameter supported by the data source, and DescriptorHandle argu ment refers to APD or ARD. FieldIdentifier argument is SQL_DESC_COUNT, and *ValuePtr argum ent is smaller than 0. RecNumber argument is equal to 0, and DescriptorHandle argument refers to APD which is implicitly assigned. (This error does not occur f

SQLSTATE	Error	Description
		or the explicitly assigned application descriptor because it is unknown the application descriptor is APD or ARD until the execution time.)
08S01	Communication link failure	Before completing the function processing, the connection between the driver and the data source is failed.
22001	String data, right truncated	FieldIdentifier argument is SQL_DESC_NAME, and BufferLength argument has the value which is bigger than SQL_MAX_IDENTIFIER_LEN.
HY000	General error	It is an error without any specific SQLSTATE.
HY001	Memory allocation error	It is a memory allocation error.
HY010	Function sequence error	The asynchronously executing function is called for DescriptorHandle related to StatementHandle, and this function is still being executed when SQLSetDescField is called. SQLExecute, SQLExecDirect, SQLBulkOperations or SQLSetPos is called on StatementHandle related to DescriptorHandle, and SQL_NEED_DATA is returned. The function is called before all data of the column or parameter are sent at the execution time. The asynchronously executing function is called for connection handle related to DescriptorHandle, and this function is still being executed when SQLSetDescField is called. SQLExecute, SQLExecDirect or SQLMoreResults is called for one of the statement handles related to DescriptorHandle, and SQL_PARAM_DATA_AVAILABLE is returned. The function is called before retrieving data for all streamed parameters.
HY013	Memory management error	The internal memory can not be accessed or the available memory size is small.
HY016	Cannot modify an implementation row descriptor	DescriptorHandle argument is related to IRD, and FieldIdentifier argument is not SQL_DESC_ARRAY_STATUS_PTR nor SQL_DESC_ROWS_PROCESSED_PTR.
HY021	Inconsistent descriptor information	SQL_DESC_TYPE and SQL_DESC_DATETIME_INTERVAL_CODE fields are not a valid ODBC SQL type, valid driver-specific SQL type or a valid ODBC C type. Descriptor information is not consistent when checking the integrity.
HY090	Invalid string or buffer length	*ValuePtr is a string and BufferLength is smaller than 0. (It is not SQL_NTS.) When the driver is the ODBC 2.x driver and the descriptor is ARD and ColumnNumber argument is set to 0, then the value specified in BufferLength is not 4.
		The value specified in FieldIdentifier argument is not the value of OD

SQLSTATE	Error	Description
HY091	Invalid descriptor field identifier	BC defined field or the implementation defined value. FieldIdentifier argument is invalid for DescriptorHandle argument. FieldIdentifier argument is the ODBC defined field and it is read-only.
HY092	Invalid attribute/option identifier	The *ValuePtr value is invalid for FieldIdentifier argument. FieldIdentifier argument is SQL_DESC_UNNAMED,and ValuePtr is SQL_NAMED.
HY105	Invalid parameter type	The value specified in SQL_DESC_PARAMETER_TYPE field is invalid. (Refer to **InputOutputType Argument** in SQLBindParameter.)
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	For more information about the suspended state, refer to **SQLEndTran**.
HYT01	Connection timeout expired	The connection timeout period expired before the response to the data request. The connection timeout period be set via SQL_ATTR_CONNECTION_TIMEOUT of SQLSetConnectAttr.
IM001	Driver does not support this function	The driver does not support the function.

Description

The application can call SQLSetDescField to set any descriptor field one at a time. One call sets a single field in a single descriptor. The function can be called for setting any field in any descriptor type if the field can be set.

If calling SQLSetDescField fails, the descriptor record contents identified by RecNumber argument is undefined.

Other functions can be called to set multiple descriptor fields with a single call. SQLSetDescRec function can set various fields which affect the data type and the binding column or the parameter buffer. (SQL_DESC_TYPE, SQL_DESC_DATETIME_INTERVAL_CODE, SQL_DESC_OCTET_LENGTH, SQL_DESC_PRECISION, SQL_DESC_SCALE, SQL_DESC_DATA_PTR, SQL_DESC_OCTET_LENGTH_PTR, SQL_DESC_INDICATOR_PTR)

SQLBindCol or SQLBindParameter can be used to completely set the columns and parameters. These functions can set a group of descriptor field with a single function call.

SQLSetDescField can be called to change the binding buffer by adding the offset to the binding points. (SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, SQL_DESC_OCTET_LENGTH_PTR) It allows the application change the binding buffers without calling SQLBindCol or SQLBindParameter like as SQL_DESC_DATA_TYPE changes SQL_DESC_DATA_PTR without changing other fields.

If the application calls SQLSetDescField to set any field other than SQL_DESC_COUNT or deferred fields (SQL_DESC_DATA_PTR, SQL_DESC_OCTET_LENGTH_PTR, SQL_DESC_INDICAITOR_PTR), then the record binding can be released.

The descriptor header fields is set by calling SQLSetDescField with appropriate FieldIdentifier. Many header fields are the statement attributes, so they can be set by calling SQLSetStmtAttr. It allows the application to set the descriptor field first without obtaining the descriptor handle. When the header field is set by calling SQLSetDescField, RecNumber argument is ignored.

RecNumber which is 0 is used to set the bookmark field.

The statement attribute SQL_ATTR_USE_BOOKMARKS should be set before calling SQLSetDescField to set the bookmark fields. It is not necessary but is strongly recommended.

Order of Setting Descriptor Fields

When setting the descriptor fields by calling SQLSetDescField, the application should follow the specified order.

1. The application should preferentially set SQL_DESC_TYPE, SQL_DESC_CONCISE_TYPE or SQL_DESC_DATETIME_INTERVAL_CODE field.

2. After one of these fields is set, the application can set the attributes of the data types and the driver can set the data type setting fields to appropriate default values for the data types. Automatic default setting of the type attribute fields ensures that the descriptor is always ready to use after the application specifies the data types. When the application explicitly sets the data type attribute, the default attribute will be overwritten.

3. After setting one of the fields in step 1 and setting the data type attribute, the application can set SQL_DESC_DATA_PTR. It prompts the consistency check of the descriptor fields. If the application sets SQL_DESC_DATA_PTR field after changing the data type or attribute, the driver sets SQL_DESC_DATA_PTR to a NULL pointer and releases the record binding. It forces the application to sequentially complete the appropriate procedures before the descriptor record is able to be used.

Initializing Descriptor Field

When a descriptor is allocated, the descriptor fields can be initialized to the default value or it can be initialized to the value which does not have default or is not defined in the descriptor type. The following table describes the initialization of each field of each descriptor type. D refers to the initialization field with th

e default value. ND refers to the initialization field without default. The number refers that the default val ue of field is a number. The table also indicates whether the field is read/write or read-only.

The field of IRD can have the default value after the statement is prepared or executed and IRD is generat ed, and it is not when the statement handle or descriptor is allocated. Any access attempt to the IRD field returns an error until IRD is generated.

Some descriptor fields are defined for one or more, but not for all descriptor types (ARD, IRD, APD, IPD). I f a field is not defined in the descriptor type, it is not required for the function which uses the descriptor.

The accessible fields by SQLGetDescField are not necessarily possible to be set with SQLSetDescField. The fields which can be set by SQLSetDescField are listed in the following table.

The following table describes the initialization of the header fields.

Header field name	Format	R/W	Default value
SQL_DESC_ALLOC_TYPE	SQLSMALLINT	ARD: R APD: R IRD: R IPD: R	ARD: SQL_DESC_ALLOC_AUTO for implicit or SQL_DESC_ALLOC_USER for explicit APD: SQL_DESC_ALLOC_AUTO for implicit or SQL_DESC_ALLOC_USER for explicit IRD: SQL_DESC_ALLOC_AUTO IPD: SQL_DESC_ALLOC_AUTO
SQL_DESC_ARRAY_SIZE	SQLULEN	ARD: R/W APD: R/W IRD: Unused IPD: Unused	ARD:[1] APD:[1] IRD: Unused IPD: Unused
SQL_DESC_ARRAY_STATUS_PTR	SQLUSMALLINT*	ARD: R/W APD: R/W IRD: R/W IPD: R/W	ARD: Null ptr APD: Null ptr IRD: Null ptr IPD: Null ptr
SQL_DESC_BIND_OFFSET_PTR	SQLLEN*	ARD: R/W APD: R/W IRD: Unused IPD: Unused	ARD: Null ptr APD: Null ptr IRD: Unused IPD: Unused
SQL_DESC_BIND_TYPE	SQLINTEGER	ARD: R/W APD: R/W IRD: Unused IPD: Unused	ARD: SQL_BIND_BY_COLUMN APD: SQL_BIND_BY_COLUMN IRD: Unused IPD: Unused
SQL_DESC_COUNT	SQLSMALLINT	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: 0 APD: 0 IRD: D IPD: 0
SQL_DESC_ROWS_P	SQLULEN	ARD: Unused APD: Unused I	ARD: Unused APD: Unused

Header field name	Format	R/W	Default value
ROCESSED_PTR	*	RD: R/W IPD: R/W	IRD: Null ptr IPD: Null ptr

[1]Fields are defined only when IPD is automatically generated by the driver. Otherwise, they are not defined. When the application tries to set the fields, SQLSTATE HY091(Invalid descriptor field identifier) will be returned.

The following table describes the initialization of the record fields.

Record field name	Format	R/W	Default value
SQL_DESC_AUTO_UNIQUE_VALUE	SQLINTEGER	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_BASE_COLUMN_NAME	SQLCHAR *	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_BASE_TABLE_NAME	SQLCHAR *	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_CASE_SENSITIVE	SQLINTEGER	ARD: Unused APD: Unused IRD: R IPD: R	ARD: Unused APD: Unused IRD: D IPD: D[1]
SQL_DESC_CATALOG_NAME	SQLCHAR *	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_CHAR_LENGTH_UNITS	SQLSMALLINT	ARD: Unused APD: Unused IRD: Unused IPD: W	ARD: Unused APD: Unused IRD: ND IPD: Unused
SQL_DESC_CONCISE_TYPE	SQLSMALLINT	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: SQL_C_ DEFAULT APD: SQL_C_ DEFAULT IRD: D IPD: ND
SQL_DESC_DATA_PTR	SQLPOINTER	ARD: R/W APD: R/W IRD: Unused IPD: Unused	ARD: Null ptr APD: Null ptr IRD: Unused IPD: Unused[2]

Record field name	Format	R/W	Default value
SQL_DESC_DATETIME_INTERVAL_CODE	SQLSMALLINT	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_DATETIME_INTERVAL_PRECISION	SQLINTEGER	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_DISPLAY_SIZE	SQLLEN	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_FIXED_PREC_SCALE	SQLSMALLINT	ARD: Unused APD: Unused IRD: R IPD: R	ARD: Unused APD: Unused IRD: D IPD: D[1]
SQL_DESC_INDICATOR_PTR	SQLLEN *	ARD: R/W APD: R/W IRD: Unused IPD: Unused	ARD: Null ptr APD: Null ptr IRD: Unused IPD: Unused
SQL_DESC_LABEL	SQLCHAR *	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_LENGTH	SQLULEN	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_LITERAL_PREFIX	SQLCHAR *	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_LITERAL_SUFFIX	SQLCHAR *	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_LOCAL_TYPE_NAME	SQLCHAR *	ARD: Unused APD: Unused IRD: R IPD: R	ARD: Unused APD: Unused IRD: D IPD: D[1]
SQL_DESC_NAME	SQLCHAR *	ARD: Unused APD: Unused IRD: R	ARD: Unused APD: Unused IRD: D

Record field name	Format	R/W	Default value
		IPD: R/W	IPD: D[1]
SQL_DESC_NULLABLE	SQLSMALLINT	ARD: Unused APD: Unused IRD:R IPD: R	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_NUM_PREC_RADIX	SQLINTEGER	ARD: R/W APD: R/W IRD:R IPD: R/W	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_OCTET_LENGTH	SQLLEN	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_OCTET_LENGTH_PTR	SQLLEN *	ARD: R/W APD: R/W IRD: Unused IPD: Unused	ARD: Null ptr APD: Null ptr IRD: Unused IPD: Unused
SQL_DESC_PARAMETER_TYPE	SQLSMALLINT	ARD: Unused APD: Unused IRD: Unused IPD: R/W	ARD: Unused APD: Unused IRD: Unused IPD: D=SQL_PARAM_INPUT
SQL_DESC_PRECISION	SQLSMALLINT	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_ROWVER	SQLSMALLINT	ARD: Unused APD: Unused IRD: R IPD: R	ARD: Unused APD: Unused IRD: ND IPD: ND
SQL_DESC_SCALE	SQLSMALLINT	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_SCHEMA_NAME	SQLCHAR *	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
SQL_DESC_SEARCHABLE	SQLSMALLINT	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused
		ARD: Unused	ARD: Unused

Record field name	Format	R/W	Default value
SQL_DESC_TABLE_NAME	SQLCHAR *	APD: Unused IRD: R IPD: Unused	APD: Unused IRD: D IPD: Unused
SQL_DESC_TYPE	SQLSMALLINT	ARD: R/W APD: R/W IRD: R IPD: R/W	ARD: SQL_C_DEFAULT APD: SQL_C_DEFAULT IRD: D IPD: ND
SQL_DESC_TYPE_NAME	SQLCHAR *	ARD: Unused APD: Unused IRD: R IPD: R	ARD: Unused APD: Unused IRD: D IPD: D[1]
SQL_DESC_UNNAMED	SQLSMALLINT	ARD: Unused APD: Unused IRD: R IPD: R/W	ARD: ND APD: ND IRD: D IPD: ND
SQL_DESC_UNSIGNED	SQLSMALLINT	ARD: Unused APD: Unused IRD: R IPD: R	ARD: Unused APD: Unused IRD: D IPD: D[1]
SQL_DESC_UPDATABLE	SQLSMALLINT	ARD: Unused APD: Unused IRD: R IPD: Unused	ARD: Unused APD: Unused IRD: D IPD: Unused

[1]Fields are defined only when IPD is automatically generated by the driver. Otherwise, they are not defin ed. When the application tries to set the fields, SQLSTATE HY091 (Invalid descriptor field identifier) will b e returned.

[2]SQL_DESC_DATA_PTR field of IPD is set to force check the consistency. In the subsequent SQLGetDesc Field or SQLGetDescRec call, the driver does not need to return the value which SQL_DESC_DATA_PTR is set to.

FieldIdentifier Argument

FieldIdentifier argument indicates the descriptor field to be set. Descriptor consists of a descriptor header and a header field described in the following header field. Descriptor records consists of the record fields described in the following header field.

Header Field

Each descriptor consists of the following fields.

SQL_DESC_ALLOC_TYPE[All] (read-only)

This read-only SQLSMALLINT header field specifies whether the descriptor is automatically allocate d by the driver or explicitly allocated by the application. The application can obtain this field but ca n not update it. The field is set to SQL_DESC_ALLOC_AUTO by the driver when the descriptor is au tomatically allocated by the driver.

SQL_DESC_ARRAY_SIZE[Application descriptors]

This SQLLEN header field in ARD specifies the number of rows in a row set. This is the number of r ows to be returned by calling SQLFetch, SQLFetchScroll, or to be operated by calling SQLBulkOpera tions or SQLSetPos.

The header field in APD, SQLULEN, specifies the number of parameters.

The default value of the field is 1. If SQL_DESC_ARRAY_SIZE is bigger than 1, SQL_DESC_DATA_PT R, SQL_DESC_INDICATOR_PTR and SQL_DESC_OCTET_LENGTH_PTR of APD or ARD points to an a rray. The constant of each array is equal to the field value.

The field in ARD can be set by calling SQLSetStmtAttr with SQL_ATTR_ROW_ARRAY_SIZE attribute. The field in APD can be set by calling SQLSetStmtAttr with SQL_ATTR_PARAMSET_SIZE attribute.

SQL_DESC_ARRAY_STATUS_PTR[All]

SQLSMALLINT* header field for each descriptor type points to an array of SQLUSMALLINT value. Th e arrays are named such as row status array (IRD), parameter status array (IPD), row operation arra y (ARD), parameter operation array (APD).

The header field in IRD points to the row status array which includes the status value after calling S QLBulkOperations, SQLFetch, SQLFetchScroll, or SQLSetPos. The application allocates the SQLUSM ALLINT array and makes the field to point to the array. The field is a NULL pointer by default. The dr iver will create an array if SQL_DESC_ARRAY_STATUS_PTR field is not set to a NULL pointer.

If the application sets the elements of the row status array pointed by SQL_DESC_ARRAY_STAT US_PTR of IRD, the driver operation is not defined.

The array is initially populated by calling SQLBulkOperations, SQLFetch, SQLFetchScroll or SQLSetPos. If th e call does not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, the content of array pointed by the fi eld is undefined. Elements of the array may contain the following values.

- SQL_ROW_SUCCESS: The row is successfully fetched, and it is not changed since the last fetch.
- SQL_ROW_SUCCESS_WITH_INFO: The row is successfully fetched, and it is not changed since the last fetch, but the warning for the row is returned.
- SQL_ROW_ERROR: An error occurs while the row is fetched.
- SQL_ROW_UPDATED: The row is successfully fetched, and it is updated since the last fetch. If the ro w is fetched again, the state is SQL_ROW_SUCCESS.
- SQL_ROW_DELETED: The row is deleted since the last fetch.
- SQL_ROW_ADDED: The row is inserted by SQLBulkOperations. If the row is fetched again, the state is

SQL_ROW_SUCCESS.
- SQL_ROW_NOROW: The row set is overlapped with the end of the result set. And any row is not retu rned corresponding to elements of the row status array.

This field of IRD can be set by calling SQLSetStmtAttr with SQL_ATTR_ROW_STATUS_PTR attribute.

SQL_DESC_ARRAY_STATUS_PTR field of IRD is valid after SQL_SUCCESS or SQL_SUCCESS_WITH_INFO is returned. If the return code is not one of these, anything pointed by SQL_DESC_ROWS_PROCESSED_PTR is not defined.

This header field in IPD indicates the parameter status array which includes status information of each par ameter after calling SQLExecute or SQLExecDirect. If SQLExecute or SQLExecDirect is called and SQL_SUC CESS or SQL_SUCCESS_WITH_INFO is not returned, then the content of the array which is pointed by the field is not defined. The application allocates SQLUSMALLINT array, the field should point to the array. Th e driver will create an array if SQL_DESC_ARRAY_STATUS_PTR field is not set to a NULL pointer. Elements in the array contain the following values.

- SQL_PARAM_SUCCESS: The SQL statement is successfully executed for the parameter set.
- SQL_PARAM_SUCCESS_WITH_INFO: The SQL statement is successfully executed for the parameter se t, but there is a warning information available to the diagnostic data structure.
- SQL_PARAM_ERROR: An error occurs to process the parameter set. The additional error information i s in the diagnostic data structure.
- SQL_PARAM_UNUSED: This parameter is not used because some previous parameter set causes an e rror which aborts further processing, or SQL_PARAM_IGNORE is set to the parameter set in the array specified by SQL_DESC_ARRAY_STATUS_PTR field of APD.
- SQL_PARAM_DIAG_UNAVAILABLE: The diagnostic information can not be used. For example, the dri ver does not generate the level of error information by treating the array of the parameter as single o ne.

This field in IPD can be set by calling SQLSetStmtAttr with SQL_ATTR_PARAM_STATUS_PTR attribute.

In ARD, this field points to the row operation array for the value set by the application to determine whet her the row is ignored in SQLSetPos operation. Elements of the array may contain the following values.

- SQL_ROW_PROCEED: The row is included in bulk operation using SQLSetPos. (This setting does not guarantee that the operation occurs on the row. If the row has SQL_ROW_ERROR state of IRD row st atus array, the driver will not be able to perform the operation for the row.)
- SQL_ROW_IGNORE: The row is excluded from the bulk operation using SQLSetPos.

If an element in the array is not set, all rows are included in the bulk operation. If the value of SQL_DESC_ ARRAY_STATUS_PTR field of ARD is a NULL pointer, all rows are included in the bulk operation. It means that the pointer points to a valid array and all elements of the array are SQL_ROW_PROCEED. If all eleme nts in an array are set to SQL_ROW_IGNORE, the values in the row status array are not updated for the ig nored row.

This field in ARD also can be set by calling SQLSetStmtAttr with SQL_ATTR_ROW_OPERATION_PTR attrib ute.

This header field in ARD indicates the parameter operation array of the values which can be set by the ap plication to determine whether the parameter set is ignored when SQLExecute or SQLExecDirect is called. Elements of the array may contain the following values.

- SQL_PARAM_PROCEED: The parameter set is included in SQLExecute or SQLExecDirect call.
- SQL_PARAM_IGNORE: The parameter set is not included in SQLExecute or SQLExecDirect call.

If an element in the array is not set, all parameter sets of the array are used to call SQLExecute or SQLExec Direct. If the value of SQL_DESC_ARRAY_STATUS_PTR field of APD is a NULL pointer, all parameter sets a re used. It is interpreted as if the pointer points to the valid array and all elements of the array are SQL_PA RAM_PROCEED.

This field in APD can be set by calling SQLSetStmtAttr with SQL_ATTR_PARAM_OPERATION_PTR attribut e.

SQL_DESC_BIND_OFFSET_PTR[Application descriptors]

This SQLLEN* header field indicates the offset of the binding. It is set as a NULL pointer by default. If the field is not a NULL pointer, the driver dereferences the pointer and adds the dereferenced val ue to each of the deferred fields which has a non-null value in the descriptor record(SQL_DESC_DA TA_PTR_, SQL_DESC_INDICATOR_PTR and SQL_DESC_OCTET_LENGTH_PTR), and uses a new poin ter value when binding.

The binding offset is always directly added to SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR and SQL_DESC_OCTET_LENGTH_PTR fields. If the offset is changed to the other value, the new val ue is continuously and directly added as each descriptor field value.

The field is a deferred field. The field is not used at the time to be set and it is used later by the driv er when the addresses of the data buffers are needed to be checked.

This field in ARD can be set by calling SQLsetStmtAttr with SQL_ATTR_ROW_BIND_OFFSET_PTR att ribute. This field in ARD can be set by calling SQLsetStmtAttr with SQL_ATTR_PARAM_BIND_OFFS ET_PTR attribute.

For more information, refer to **SQLFetchScroll** or **SQLBindParameter**.

SQL_DESC_BIND_TYPE[Application descriptors]

This SQLUINTEGER header field is used to set the binding direction.

This field in ARD specifies the binding direction when SQLFetchScroll or SQLFetch is called in the rel ated statement handle.

This field is set to SQL_BIND_BY_COLUMN(default) to select the column-wise binding for the colu mn.

This field in ARD can be set by calling SQLSetStmtAttr with SQL_ATTR_ROW_BIND_TYPE attribute.

This field specifies the binding direction to be used in the Dynamic parameters.

This field is set to SQL_BIND_BY_COLUMN(default) to select the column-wise binding for the para meter.

This field can be set by calling SQLSetStmtAttr with SQL_ATTR_PARAM_BIND_TYPE attribute.

SQL_DESC_COUNT[All]

This header field of SQLSMALLINT specifies a 1-based index of the best record including the data. When the driver sets the data structure to the descriptor, SQL_DESC_COUNT should be set to displ ay how many of the important records. The application does not need to specify the amount of sp ace reserved for the records when allocating instances of the data structure. As like the application specifies the contents of the records, the driver performs the requested operation to ensure that th e descriptor handle indicates the data structure of sufficient size.

SQL_DESC_COUNT is not the number of all bound data columns or the number of parameters but is the number of highest-numbered records. If the binding of the highest-numbered column or par ameter is released, SQL_DESC_COUNT is changed to the number of the next highest-numbered co lumns or parameters. If the binding of column or parameter which is smaller than the number of hi ghest-numbered columns or parameters is released(when TargetValuePtr argument is set to a NUL L pointer and SQLBindCol is called, or when ParameterValuePtr argument is set to a NULL pointer a nd SQLBindParameter is called.) SQL_DESC_COUNT is not changed. If columns or parameters to be added are bound with the number which is bigger than the highest-numbered record including th e data, the driver automatically increases the value of SQL_DESC_COUNT field. If the binding for all columns are released by calling SQLFreeStmt with SQL_UNBIND option, SQL_DESC_COUNT field in ARD or IRD is set to 0. If SQLFreeStmt is called as SQL_RESET_PARAMS option, SQL_DESC_COUNT fields in APD and IPD are set to 0.

SQL_DESC_COUNT value can be explicitly set by calling SQLSetDescField in the application. If SQL_ DESC_COUNT value is clearly reduced, all records having a bigger value than the new SQL_DESC_C OUNT value are effectively deleted. If the value of SQL_DESC_COUNT field of ARD is clearly set to 0, all buffers are released except for the bound bookmark column. The number of records in the field of ARD do not include the bound bookmark column. The only way to release the bound bookmark column is set SQL_DESC_DATA_PTR to a NULL pointer.

SQL_DESC_ROWS_PROCESSED_PTR[Implementation descriptors]

This SQLULEN* header field in IRD points to the buffers which include the number of rows fetched after SQLFetch or SQLFetchScroll is called, or the number of rows and errors affected in the bulk o peration performed by calling SQLBulkOperations or SQLSetPos.

This SQLUINTEGER* header field of IPD points to the buffers which include the number of paramet er set and errors processed. If it is a NULL pointer, it will not return the number.

SQL_DESC_ROWS_PROCESSED_PTR is valid only after SQL_SUCCESS or SQL_SUCCESS_WITH_INF O has been returned after calling SQLFetch or SQLFetchScroll (for an IRD field) or SQLExecute, SQL ExecDirect, or SQLParamData (for an IPD field). If the functions does not return SQL_SUCCESS or S

QL_SUCCESS_WITH_INFO, the buffer content is not defined and the buffer value is set to 0 until S QL_NO_DATA is returned.

This field in ARD can be set by calling SQLSetStmtAttr with SQL_ATTR_ROWS_FETCHED_PTR attrib ute. This field in APD can be set by calling SQLSetStmtAttr with SQL_ATTR_PARAMS_PROCESSED_ PTR attribute.

The buffer which is pointed by this field is allocated by the application. It is the deferred output buf fer set by the driver. A NULL pointer is set by default.

Record Field

Each descriptor includes at least one record consisting of the fields which defines one of the columns of d ata or dynamic parameters depending on the descriptor type. Each record is a complete specification of a single column or parameter.

SQL_DESC_AUTO_UNIQUE_VALUE[IRDs] (read-only)

This read-only SQLINTEGER record field has SQL_TRUE if it is auto-increment column. Otherwise, it has the SQL_FALSE. The field is read-only but it does not necessarily need to be read-only, if it is au to-increment column.

SQL_DESC_BASE_COLUMN_NAME[IRDs] (read-only)

This read-only SQLCHAR* record field includes the base column name of the result set column. If t he base column name does not exist, the field includes an empty string.

SQL_DESC_TABLE_NAME[IRDs] (read-only)

This read-only SQLCHAR* record field includes the base table name of the result set column. If the base table name can not be defined or not available, the field includes an empty string.

SQL_DESC_CASE_SENSITIVE[Implementation descriptors] (read-only)

This read-only SQLINTEGER record field is SQL_TRUE if columns or parameters is case-sensitive whe n sorting and comparing. It is SQL_FALSE if it is case-insensitive or the column with non-character.

SQL_DESC_CATALOG_NAME[IRDs] (read-only)

This read-only SQLCHAR* record field includes the catalog of the base table including the column. If the column is a part of a expression or view, the return value depends on the driver. If the data s ource does not support or the catalog can not be determined, the field includes an empty string.

SQL_DESC_CHAR_LENGTH_UNITS[IPD]

This SQLSMALLINT record field describes the unit of SQL_DESC_LENGTH. (SQL_CLU_NONE/ SQL_C LU_CHARACTERS/ SQL_CLU_OCTETS)

SQL_DESC_CONCISE_TYPE[All]

This SQLSMALLINT header field specifies a simplified form for all data types including datetime and interval data types.

The values of SQL_DESC_CONCISE_TYPE, SQL_DESC_TYPE and SQL_DESC_DATETIME_INTERVAL_ CODE fields are interdependent. If time is set in a field, it should be set in others fields as well. SQL

_DESC_CONCISE_TYPE can be set by calling SQLBindCol, SQLBindParameter or SQLSetDescField. SQL_DESC_TYPE can be set by calling SQLSetDescField or SQLSetDescRec.

If SQL_DESC_CONCISE_TYPE is set to a concise data type except for the interval or datetime data type, SQL_DESC_TYPE field is set to the same value and SQL_DESC_DATETIME_INTERVAL_CODE field is set to 0.

If SQL_DESC_CONCISE_TYPE is set to a concise interval or datetime data type, SQL_DESC_TYPE field is set to a detailed data type(SQL_DATETIME or SQL_INTERVAL), and SQL_DESC_DATETIME_INTERVAL_CODE field is set to the appropriate sub code.

SQL_DESC_DATETIME_INTERVAL_CODE[All]

This SQLSMALLINT record field includes the sub code for specifying datetime or interval data type if SQL_DESC_TYPE field is SQL_DATETIME or SQL_INTERVAL. It is same for both SQL and C. The code includes the data type name with CODE which is replaced with TYPE or C_TYPE of datetime types, INTERVAL or C_INTERVAL of interval types.

If SQL_DESC_TYPE and SQL_DESC_CONCISE_TYPE of the application descriptor are set to SQL_C_DEFAULT and the descriptor is not related to the statement handle, the content of SQL_DESC_DATETIME_INTERVAL_CODE is not defined.

This field can set the datetime data types which is listed in the following table.

Datetime type	DATETIME_INTERVAL_CODE
SQL_TYPE_DATE/ SQL_C_TYPE_DATE	SQL_CODE_DATE
SQL_TYPE_TIME/ SQL_C_TYPE_TIME	SQL_CODE_TIME
SQL_TYPE_TIME_WITH_TIMEZONE/ SQL_C_TYPE_TIME_WITH_TIMEZONE	SQL_CODE_TIME_WITH_TIMEZONE
SQL_TYPE_TIMESTAMP/ SQL_C_TYPE_TIMESTAMP	SQL_CODE_TIMESTAMP
SQL_TYPE_TIMESTAMP_WITH_TIMEZONE/ SQL_C_TYPE_TIMESTAMP_WITH_TIMEZONE	SQL_CODE_TIMESTAMP_WITH_TIMEZONE

This field can set the interval data types which is listed in the following table.

Interval type	DATETIME_INTERVAL_CODE
SQL_INTERVAL_DAY/ SQL_C_INTERVAL_DAY	SQL_CODE_DAY
SQL_INTERVAL_DAY_TO_HOUR/ SQL_C_INTERVAL_DAY_TO_HOUR	SQL_CODE_DAY_TO_HOUR
SQL_INTERVAL_DAY_TO_MINUTE/ SQL_C_INTERVAL_DAY_TO_MINUTE	SQL_CODE_DAY_TO_MINUTE
SQL_INTERVAL_DAY_TO_SECOND/ SQL_C_INTERVAL_DAY_TO_SECOND	SQL_CODE_DAY_TO_SECOND

Interval type	DATETIME_INTERVAL_CODE
SQL_INTERVAL_HOUR/ SQL_C_INTERVAL_HOUR	SQL_CODE_HOUR
SQL_INTERVAL_HOUR_TO_MINUTE/ SQL_C_INTERVAL_HOUR_TO_MINUTE	SQL_CODE_HOUR_TO_MINUTE
SQL_INTERVAL_HOUR_TO_SECOND/ SQL_C_INTERVAL_HOUR_TO_SECOND	SQL_CODE_HOUR_TO_SECOND
SQL_INTERVAL_MINUTE/ SQL_C_INTERVAL_MINUTE	SQL_CODE_MINUTE
SQL_INTERVAL_MINUTE_TO_SECOND/ SQL_C_INTERVAL_MINUTE_TO_SECOND	SQL_CODE_MONUTE_TO_SECOND
SQL_INTERVAL_MONTH/ SQL_C_INTERVAL_MONTH	SQL_CODE_MONTH
SQL_INTERVAL_SECOND/ SQL_C_INTERVAL_SECOND	SQL_CODE_SECOND
SQL_INTERVAL_YEAR/ SQL_C_INTERVAL_YEAR	SQL_CODE_YEAR
SQL_INTERVAL_YEAR_TO_MONTH/ SQL_C_INTERVAL_YEAR_TO_MONTH	SQL_CODE_YEAR_TO_MONTH

SQL_DESC_DATETIME_INTERVAL_PRECISION[All]

This SQLINTEGER record field includes interval leading precision if SQL_DESC_TYPE field is SQL_INTERVAL. When SQL_DESC_DATETIME_INTERVAL_CODE field is set to interval data type, the field is set to the default interval leading precision.

SQL_DESC_DISPLAY_SIZE[IRDs] (read-only)

This read-only SQLINTEGER record field includes the maximum number of characters required to display the data from the column.

SQL_DESC_FIXED_PREC_SCALE[Implementation descriptors] (read-only)

This read-only SQLSMALLINT record field is set to SQL_TRUE if the column is a exact numeric column and it has the non-zero scale and the fixed precision. Otherwise, it is set to SQL_FALSE.

SQL_DESC_INDICATOR_PTR[Application descriptors]

This SQLLEN* record field in ARD indicates the indicator variable. The variable contains SQL_NULL_DATE if the column value is NULL. In APD, the indicator variable is set to SQL_NULL_DATA to specify a dynamic argument to NULL. Otherwise, the variable is 0.

If SQL_DESC_INDICATOR_PTR field of ARD is a NULL pointer, the driver is prevened from returning the information about whether the column is NULL. If the column is NULL and SQL_DESC_INDICATOR_PTR is a NULL pointer, SQLSTATE 22002(Indicator variable required but not supplied) is returned when the driver tries to create a buffer after calling SQLFetch or SQLFetchScroll. If SQLFetch or SQLFetchScroll call does not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, the buffer content is not defined.

SQL_DESC_INDICATOR_PTR field defines whether to set the field indicated by SQL_DESC_OCTET_LENGTH_PTR. If the column data value is NULL, the driver sets the indicator variable to SQL_NULL_DATA. The field indicated by SQL_DESC_OCTET_LENGTH_PTR is not set at that time. If the NULL value is not encountered during the fetch, the buffer indicated by SQL_DESC_INDICATOR_PTR is set to 0, and the buffer indicated by SQL_DESC_OCTET_LENGTH_PTR is set to the data length.

If SQL_DESC_INDICATOR_PTR field of APD is a NULL pointer, the application can not use the descriptor record to specify the arguments as NULL.

It is a deferred field. The field is not used when it is set but it is used when the driver describes the possibility of NULL (for ARD) or determines the possibility of NULL (for APD).

SQL_DESC_LABEL[IRDs] (read-only)

This read-only SQLCHAR* record field includes the column label or cover. If the column does not have its label, the variable includes the column name. If the column is not named or can not use its label, the variable includes an empty string.

SQL_DESC_LENGTH[All]

This SQLULEN record field is a maximum length or actual length of the string, or binary data type in bytes. It is the actual length of fixed length data types or the maximum length of variable-length data types. The value always excludes NULL termination character at the end of the string. If the data type of the value is SQL_TYPE_DATE, SQL_TYPE_TIME, SQL_TYPE_TIMESTAMP or SQL interval data type, the field has the length of characters when the datetime or interval value is represented by string again.

This field value may be different from the value of the length defined in ODBC 2.x.

SQL_DESC_LITERAL_PREFIX[IRDs] (read-only)

This read-only SQLCHAR* record field includes the characters or character which the driver recognizes as a suffix. The variable includes an empty string for the data type to which the character suffix is not applicable.

SQL_DESC_LOCAL_TYPE_NAME[Implementation descriptors] (read-only)

This read-only SQLCHAR* record field includes a localized name for the data type which may be different from the regular name of the data type. If the localized name does not exist, then an empty string is returned. The field is only for the display purpose.

SQL_DESC_NAME[Implementation descriptor]

This SQLCHAR* record field includes the alias of the field in the row descriptor. If the alias of a column is not applied, the column name is returned. In either case, the driver sets SQL_DESC_UNNAMED field to SQL_NAMED when setting SQL_DESC_NAME. If column name or its alias does not exist, then the driver returns an empty string of SQL_DESC_NAME field and sets SQL_DESC_UNNAMED field to SQL_UNNAMED.

The application can set SQL_DESC_NAME field of IPD to a parameter name or alias to specify stored procedure parameters by name. SQL_DESC_NAME field of IRD is a read-only field. If the application tries to set the field, SQLSTATE HY091 (Invalid descriptor field identifier) is returned.

For IPD, if the driver does not support the named parameters, the field is not defined. If the driver s upports the named parameters and it can explain the parameters, the parameter name is returned in this field.

SQL_DESC_NULLABLE[Implementation descriptors] (read-only)

For IRD, if the column of this read-only SQLSMALLINT record field is allowed to be NULL, it is SQL_ NULLABLE. If NULL is not allowed, it is SQL_NO_NULLS. If the column is unknown whether or not t o allow NULL, it is SQL_NULLABLE_UNKNOWN. The field exists particularly for a column in the resu lt set.

For IPD, this field is always set to SQL_NULLABLE because the dynamic parameters are always allow ed to have NULL and can not be set by the application.

SQL_DESC_NUM_PREC_RADIX[All]

This SQLINTEGER record field has the value of 2 when SQL_DESC_TYPE field is the approximate nu meric data type. Because SQL_DESC_PRECISION field includes the number of the bits. The field has the value of 10 when SQL_DESC_TYPE field has the exact numeric data type because SQL_DESC_P RECISION field includes the number of decimal places. The field sets 0 for a non-numeric data type.

SQL_DESC_OCTET_LENGTH[All]

This SQLLEN record field includes the length in bytes of a string or binary data type. For the fixed le ngth character or binary data type, it is the actual length (in bytes). For the variable length characte r or binary data type, it is the maximum length (in bytes). The value does not include a space of nul l-termination character for the implementation descriptor but it includes a space of null-terminatio n character for the application descriptor. For the application data, the field contains the buffer siz e. For APD, the field is defined only for the output or input/output parameters.

SQL_DESC_OCTET_LENGTH_PTR[Application descriptors]

This SQLLEN* record field points to a variable which includes the total length in bytes of the dynam ic argument (for parameter descriptor) or the bound column value (for row descriptors).

For APD, the value is ignored for all argument except for the string and binary. If the field is SQL_N TS, the dynamic argument should be null-terminated. The application sets the variable which inclu des the result of SQL_DATA_AT_EXEC or SQL_LEN_DATA_AT_EXEC macro at execution time in th e record field of APD to indicate that it is a bound parameter to be data-at-execution parameter. If one or more of such fields exist, SQL_DESC_DATA_PTR can be set to a value identifying the param eter to help the application determine which parameter is being requested.

If OCTET_LENGTH_PTR field of ARD is a NULL pointer, the driver does not return the column lengt h information. If SQL_DESC_OCTET_LENGTH_PTR of APD is a NULL pointer, the driver assumes tha t the string and binary values are null-terminated. (Binary values should not be null-terminated, but a length should be given to avoid data interruption.)

If SQLFetch or SQLFetchScroll which fills the buffer pointed by the field does not return SQL_SUCC ESS or SQL_SUCCESS_WITH_INFO, the buffer content is not defined. It is the deferred field. The fie ld is not immediately used, but it is used later when the driver indicates or determines the octet len gth of the data.

SQL_DESC_PARAMETER_TYPE[IRDs]

This SQLSMALLINT record field is set to SQL_PARAM_INPUT for the input parameters, to SQL_PARAM_INPUT_OUTPUT for the input/output parameters, to SQL_PARAM_OUTPUT for the output parameters, to SQL_PARAM_INPUT_OUTPUT_STREAM for the stream input/output parameters or to SQL_PARAM_OUTPUT_STREAM for the stream output parameters. The default is SQL_PARAM_INPUT.

SQL_DESC_PRECISION[All]

This SQLSMALLINT record field includes the number of effective integer for the exact numeric type, and the number of bits of mantissa (binary precision) for the approximate numeric type. Or, it includes the number of digits in fractional seconds part of SQL_TYPE_TIME, SQL_TYPE_TIMESTAMP, or SQL_INTERVAL_SECOND data type. The field is undefined for all other data types.

The field value may be different from the value of the precision defined in ODBC 2.x.

SQL_DESC_ROWVER[Implementation descriptors] (read-only)

This SQLSMALLINT record field indicates whether the column is automatically updated by DBMS when the row is updated (e.g. timestamp of SQL server). The value of record field is set to SQL_TRUE if the column is the versioning column. Otherwise, it is set to SQL_FALSE. The column attribute is similar to calling SQLSpecialColumns with IdentifierType of SQL_ROWVER to determine whether to automatically update the column.

SQL_DESC_SCALE[All]

This SQLSMALLINT record field includes the number of decimal places defined in the decimal and numeric data types. The field is not defined for all other data types.

The field value may be different from the value of the scale defined in ODBC 2.x.

SQL_DESC_SCHEMA_NAME[IRDs] (read-only)

This read-only SQLCHAR* record field includes the schema name of the base table including the column. The return value is driver-dependent if the column is a part of a expression or view. If the data source does not support the schema or can not identify the schema name, then this variable contains an empty string.

SQL_DESC_SEARCHABLE[IRDs] (read-only)

This read-only SQLSMAINT record field sets one of the following values.

• It is set to SQL_PRED_NONE if the column can not be used in WHERE clause. (It is as same as SQL_UNSEARCHABLE of ODBC 2.X.)

• It is set to SQL_PRED_CHAR if the column can be used in a WHERE clause but only with the LIKE predicate.

• It is set to SQL_PRED_BASIC if the column can be used with all comparison operators except for LIKE in WHERE clause. (It is as same as SQL_EXCEPT_LIKE of ODBC 2.x .)

• It is set to SQL_PRED_SEARCHABLE if the column can be used with all comparison operators in WHERE clause.

SQL_DESC_TABLE_NAME[IRDs] (read-only)

This read-only SQLCHAR* record field includes the base table name containing this column. The return value is driver-dependent if the column is an expression or a part of view.

SQL_DESC_TYPE[All]

This SQLSMALLINT record field includes an abbreviated SQL or C data types for all data types except for the datetime or interval data type. The field specifies SQL_DATETIME or SQL_INTERVAL for datetime or interval data type.

Whenever the field includes SQL_DATETIME or SQL_INTERVAL, SQL_DESC_DATETIME_INTERVAL_CODE field should include the appropriate server code of the concise form. SQL_DESC_TYPE includes SQL_DATETIME for the datetime data type, and SQL_DESC_DATETIME_INTERVAL_CODE field includes the sub code specifying the datetime data type. SQL_DESC_TYPE includes SQL_INTERVAL for the interval data type and SQL_DESC_DATETIME_INTERVAL_CODE field includes the sub code specifying the interval data type.

The values of SQL_DESC_TYPE and SQL_DESC_CONCISE_TYPE fields are interdependent. The other field should be set when one of the fields is set. SQL_DESC_TYPE can be set by calling SQLSetDescField or SQLSetDescRec. SQL_DESC_CONCISE_TYPE can be set by calling SQLBindCol, SQLBindParameter or SQLSetDescField.

If SQL_DESC_TYPE is set to the concise data type which is not interval nor datetime data type, SQL_DESC_CONCISE_TYPE field is set to the same value and SQL_DESC_DATETIME_INTERVAL_CODE field is set to 0.

If SQL_DESC_TYPE is set to the verbose data type of the datetime or interval data type(SQL_DATETIME or SQL_INTERVAL) and SQL_DESC_DATETIME_INTERVAL_CODE field is set to the appropriate sub code, SQL_DESC_CONCISE_TYPE field is set to the value corresponding to the concise data type. When trying to set SQL_DESC_TYPE to one of the concise datetime or interval data types, SQL STATE HY021(Inconsistent descriptor information) will be returned.

If SQL_DESC_TYPE field is set by calling SQLBindCol, SQLBindParameter or SQLSetDescField, the following fields are set to the default values described in the following table. The remaining field values of the same record are undefined.

SQL_DESC_TYPE value	Other fields implicitly set
SQL_CHAR, SQL_VARCHAR, SQL_C_CHAR, SQL_C_VARCHAR	SQL_DESC_LENGTH is set to 1. SQL_DESC_PRECISION is set to 0.
SQL_DATETIME	When SQL_DESC_DATETIME_INTERVAL_CODE is set to SQL_CODE_DATE or SQL_CODE_TIME, SQL_DESC_PRECISION is set to 0. When it is set to SQL_DESC_TIMESTAMP, SQL_DESC_PRECISION is set to 6.
SQL_DECIMAL, SQL_NUMERIC,	SQL_DESC_SCALE is set to 0. SQL_DESC_PRECISION is set to the implementation-defined precision for each data type.

SQL_DESC_TYPE value	Other fields implicitly set
SQL_C_NUMERIC	
SQL_FLOAT, SQL_C_FLOAT	SQL_DESC_PRECISION is set to the implementation-defined precision for SQL_FLOAT.
SQL_INTERVAL	If SQL_DESC_DATETIME_INTERVAL_CODE is set to interval data type, SQL_DESC_DATETIME_INTERVAL_PRECISION is set to 2 (default interval leading precision). When interval has the part of seconds, SQL_DESC_PRECISION is set to 6 (default interval seconds precision).

When the application sets the field of the descriptor by calling SQLSetDescField not SQLSetDescRec, the application should firstly define the data type. When doing like this, the other fields described in the table above are implicitly set. If any value is not allowed to set implicitly, the application can call SQLSetDescField or SQLSetDescRec to set the value explicitly.

SQL_DESC_TYPE_NAME[Implementation descriptors] (read-only)

This read-only SQLCHAR* record field includes the data source dependent type name (CHAR, VARCHAR etc.). If the data type name is unknown, the variable contains an empty string.

SQL_DESC_UNNAMED[Implementation descriptors]

If SQL_DESC_NAME field is set, the SQLSMALLINT record field in the row descriptor is set to one of SQL_NAME or SQL_UNNAMED by the driver. If SQL_DESC_NAME field includes the column alias or the column alias is not used, the driver sets SQL_DESC_UNNAMED field to SQL_NAMED. If the application sets SQL_DESC_NAME field of IPD to the parameter name or alias, the driver sets SQL_DESC_UNNAMED of IPD to SQL_NAMED. If column name or alias does not exist, the driver sets SQL_DESC_UNNAMED field to SQL_UNNAMED.

The application sets SQL_DESC_UNNAMED field of IPD to SQL_UNNAMED. If the application tries to set SQL_DESC_UNANMED field of IPD to SQL_NAMED, the driver returns SQLSTATE HY091 (Invalid descriptor field identifier). If the application tries to set SQL_DESC_UNNAMED field of IRD (read-only), SQLSTATE HY091 (Invalid descriptor field identifier) will be returned.

SQL_DESC_UNSIGNED[Implementation descriptors] (read-only)

The read-only SQLSMALLINT record field is set to SQL_TRUE if the column type is the unsigned or non-numeric, or it is set to SQL_FALSE if the column type is signed.

SQL_DESC_UPDATABLE[IRDs] (read-only)

This read-only SQLSMALLINT record field is set to one of the following values.

- It is set to SQL_ATTR_READ_ONLY if the result set column is read-only.
- It is set to SQL_ATTR_WRITE if the result set column is read and write.
- It is set to SQL_ATTR_READWRITE_UNKNOWN if the updatability of the result set column is unknown.

SQL_DESC_UPDATABLE describes the updatability of the column (It is not the column in the base t

able.) in the result set. The updatability of the column in the base table whose result set columns ar e based on can be different from the value in the field. The updatability of the column is based on t he definition of the data type, user privilege and result set itself. If the updatability of the column is unsure, SQL_ATTR_READWITE_UNKNOWN should be returned.

SQL_DESC_CHAR_LENGTH_UNITS[Alls] (read-only)

This SQLSMALLINT record field indicates the length unit of the column whose SQL type is SQL_CH AR, SQL_VARCHAR and SQL_LONGVARCHAR.

• SQL_CLU_CHARACTERS: The length unit is CHARACTER. For example, data of "문자열" refers to the length of 3, if the encoding type is UHC(Unified Hangul Code).

• SQL_CLU_OCTETS: The length unit is OCTETS. For example, data of "문자열" refers to the length of 6, if the encoding type is UHC(Unified Hangul Code).

• SQL_CLU_NONE: The length unit is undefined. It is the value returned for SQL type except for th e SQL types listed above.

Consistency Check

Concurrency check is automatically implemented by the driver each time the application passes the value to SQL_DESC_DATA_PTR of ARD, APD, or IPD. If anything of the fields is inconsistent with other fields, S QLSetDescField will return SQLSTATE HY021(Inconsistent descriptor information). For more information, refer to **Consistency Check** of **SQLSetDescRec**.

SQLSetDescRec

Conformance

Introduced version: ODBC 3.0
Standards compliance: ISO 92

Overview

SQLSetDescRec function sets the multiple descriptor fields affecting the buffer bound to a data type or co lumn or parameter data.

Syntax

```
SQLRETURN SQLSetDescRec(
    SQLHDESC      DescriptorHandle,
    SQLSMALLINT   RecNumber,
    SQLSMALLINT   Type,
    SQLSMALLINT   SubType,
    SQLLEN        Length,
    SQLSMALLINT   Precision,
    SQLSMALLINT   Scale,
    SQLPOINTER    DataPtr,
    SQLLEN *      StringLengthPtr,
    SQLLEN *      IndicatorPtr);
```

Arguments

DescriptorHandle

[Input] It is the descriptor handle. IRD handle can not be set.

RecNumber

[Input] It indicates the descriptor record including the fields to be set. The descriptor record starts f rom 0, and the 0 record is a bookmark record. The argument should be equal to or bigger than 0. I f RecNumber is bigger than SQL_DESC_COUNT, then SQL_DESC_COUNT is changed to RecNumbe r value.

Type

[Input] It is the value which sets SQL_DESC_TYPE field for the descriptor record.

SubType

[Input] It is the value which sets SQL_DESC_DATETIME_INTERVAL_CODE field for a record type of SQL_DATETIME or SQL_INTERVAL.

Length

[Input] It is the value which sets SQL_DESC_OCTET_LENGTH field for the descriptor record.

Precision

[Input] It is the value which sets SQL_DESC_PRECISION field for the descriptor record.

Scale

[Input] It is the value which sets SQL_DESC_SCALE field for the descriptor record.

DataPtr

[Deferred Input or Output] It is the value which sets SQL_DESC_DATA_PTR field for the descriptor record. DataPtr can be set to a NULL pointer.

DataPtr argument can be set to a NULL pointer to set SQL_DESC_DATA_PTR field to a NULL pointer. If the handle in DescriptorHandle argument is related to ARD, it unbinds the column.

StringLengthPtr

[Deferred Input or Output] It is the value which sets SQL_DESC_OCTET_LENGTH_PTR field for the descriptor record. StringLengthPtr can be set to a NULL pointer to set SQL_DESC_OCTET_LENGTH_PTR field to a NULL pointer.

IndicatorPtr

[Deferred Input or Output] It is the value which sets SQL_DESC_INDICATOR_PTR field of the descriptor record. IndicatorPtr can be set to a NULL pointer to set SQL_DESC_INDICATOR_PTR to a NULL pointer.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01000	General warning	It is the driver-specific informational message. (The function returns SQL_SUCCESS_WITH_INFO.)
07009	Invalid descriptor index	RecNumber argument is set to 0, and DescriptorHandle refers to the IPD handle. RecNumber argument is bigger than the maximum number of columns or parameters supported by the data source, and DescriptorHandle argument is APD, IPD or ARD.

SQLSTATE	Error	Description
		RecNumber argument is 0, and DescriptorHandle argument refers to the implicitly allocated APD. (The error does not occur for the explicitly assigned application descriptor. This is because it is unknown whether the explicitly allocated application descriptor is APD or ARD until the execution time.)
08S01	Communication link failure	Before the function processing is completed, the connection between the driver and the data source is failed.
HY000	General error	It is an error without specific SQLSTATE, and the error message returned by SQLGetDiagRec in *MessageText buffer describes error message and its cause.
HY001	Memory allocation error	The driver can not allocate the memory required for execution or completion of the function.
HY010	Function sequence error	The asynchronously executing function is called for StatementHandle related to DescriptorHandle, and this function is still being executed when SQLSetDescRec is called.
HY013	Memory management error	The function is not executed because the underlying memory object can not be accessed because of the low memory condition.
HY016	Cannot modify an implementation row descriptor	DescriptorHandle argument is related to IRD.
HY021	Inconsistent descriptor information	The Type field, or other field related to SQL_DESC_TYPE field of the descriptor is not valid or consistent.
HY090	Invalid string or buffer length	The driver is set to ODBC 2.x driver, and the descriptor is set to ARD, and ColumnNumber argument is set to 0 and the value specified in BufferLength argument is not 4.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed	For more information about the suspended state, refer to **SQLEndTran**.
HYT01	Connection timeout expired	The connection timeout period is expired before the data source response to the request. The connection timeout interval can be set via SQL_ATTR_CONNECTION_TIMEOUT of SQLSetConnectAttr.
IM001	Driver does not support this function	The driver does not support the function.

Description

The application can call SQLSetDescRec to set the following fields for a column or parameter.

- SQL_DESC_TYPE
- SQL_DESC_DATETIME_INTERVAL_CODE (For the record of SQL_DATETIME or SQL_INTERVAL type)
- SQL_DESC_OCTET_LENGTH
- SQL_DESC_PRECISION
- SQL_DESC_SCALE
- SQL_DESC_DATA_PTR
- SQL_DESC_OCTET_LENGTH_PTR
- SQL_DESC_INDICATOR_PTR

If calling SQLSetDescRec fails, the content of the descriptor record identified by RecNumber argument is not defined.

When binding a column or parameter, SQLSetDescRec allows to change the multiple fields which affect the binding without calling SQLBindCol, SQLBindParameter or calling SQLSetDescField multiple times. SQLSetDescRec can set the descriptor fields which are not related to the current statement. SQLBindParameter can set more fields than SQLSetDescRec, and it can set the fields of APD and IPD with a single call and it does not require a descriptor handle.

The SQL_ATTR_USE_BOOKMARKS statement attribute should be set before calling SQLSetDescRec with RecNumber argument of 0 to set bookmark fields. It is not necessary but is strongly recommended.

Consistency Check

The consistency check is automatically implemented by the driver whenever the application sets SQL_DESC_DATA_PTR field of APD, ARD, or IPD. If any of the fields is not consistent with other fields, SQLSetDescRec will return SQLSTATE HY021. (Inconsistent descriptor information)

Whenever the application sets SQL_DESC_DATA_PTR field of APD, ARD or IPD, the driver checks the value of SQL_DESC_TYPE field, and checks if the values applicable to that SQL_DESC_TYPE field is valid and consistent. The check is implemented when SQLBindParameter or SQLBindCol is called or whenever SQLSetDescRec is called to APD, ARD or IPD. The consistency check includes the following descriptor fields.

- SQL_DESC_TYPE field should be one of the valid ODBC C, SQL type or the driver-specific SQL type. SQL_DESC_CONCISE_TYPE field should be one of the valid ODBC C, SQL Type or the driver-specific SQL type, and it includes the simplified datetime and interval types.
- If SQL_DESC_TYPE record field is SQL_DATETIME or SQL_INTERVAL, SQL_DESC_DATETIME_INTERVA

L_CODE field should be one of the valid datetime or interval codes. (For more information refer to th e description of SQL_DESC_DATETIME_INTERVAL_CODE field in **SQLSetDescField**.)

- If SQL_DESC_TYPE field is a NUMERIC type, SQL_DESC_PRECISION and SQL_DESC_SCALE fields are c hecked whether they are valid.
- If SQL_DESC_CONCISE_TYPE field is the time or timestamp data type an interval type with a seconds component, or one of the interval data types with a time component, SQL_DESC_PRECISION field is c hecked whether it is a valid second precision.
- If SQL_DESC_CONCISE_TYPE is an interval data type, SQL_DESC_DATETIME_INTERVAL_PRECISION fi eld is checked to be a valid interval leading precision value.

SQL_DESC_DATA_PTR field of IPD is not generally set. The application can do so to force the consistency check for the IPD fields. Consistency check can not be executed in IRD. The value of SQL_DESC_DATA_PT R field of IPD is not actually stored and it can not be retrieved via SQLGetDescField or SQLGetDescRec. Th e setting is made only to force the consistency check.

SQLSetEnvAttr

Conformance

Introduced version: ODBC 3.0
Standards compliance: ISO 92

Overview

SQLSetEnvAttr sets the environment management attributes.

Syntax

```
SQLRETURN SQLSetEnvAttr(
    SQLHENV      EnvironmentHandle,
    SQLINTEGER   Attribute,
    SQLPOINTER   ValuePtr,
    SQLINTEGER   StringLength);
```

Arguments

EnvironmentHandle

[Input] It is the environment handle.

Attribute

[Input] They are listed in **Description**.

ValuePtr

[Input] It is a pointer to the value which is related to an attribute. Depending on the value of Attribute, ValuePtr points to a 32 bit integer value or null-terminated character.

StringLength

[Input] If ValuePtr points to the string or binary buffer, the argument should be the length of *ValuePtr. The argument should include the number of bytes in string for the character string data.

If ValuePtr is an integer, StringLength is ignored.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01000	General warning	It is the driver-specific informational message. (The function returns SQL_SUCCESS_WITH_INFO.)
01S02	Option value changed	The driver does not support the value specified in ValuePtr and replaces it with a similar value. (The function returns SQL_SUCCESS_WITH_INFO.)
HY000	General error	It is an error without specific SQLSTATE, and the error message returned by SQLGetDiagRec in *MessageText buffer describes error message and its cause.
HY001	Memory allocation error	The driver can not allocate the memory required for execution or completion of the function.
HY009	Invalid use of null pointer	The attribute argument identifies an environment attribute which requires a string value, and ValuePtr argument is a NULL pointer.
HY010	Function sequence error	The connection handle is allocated to EnvironmentHandle. SQL_ATTR_ODBC_VERSION is not set to SQLSetEnvAttr, Attribute is different from SQL_ATTR_ODBC_VERSION. If SQLAllocHandleStd is used, SQL_ATTR_ODBC_VERSION does not need to be explicitly specified.
HY013	Memory management error	The function is not executed because the underlying memory object can not be accessed because of the low memory condition.
HY024	Invalid attribute value	Considering the specified attribute value, the value of ValuePtr is not valid.
HY090	Invalid string or buffer length	StringLength argument is smaller than 0. (It is not SQL_NTS.)
HY092	Invalid attribute/option identifier	The specified value of attribute argument is not valid in ODBC version supported by the driver.
HY117	Connection is suspended due to unknown trasaction state. Only disconnect and read-only functions ard allowed	For more information about the suspended state, refer to **SQLEndTran**.
HYC00	Optional feature not implemented	The specified value of attribute argument is valid fot the ODBC environment attribute supported by the driver but it is not supported by the driver. An attribute argument is SQL_ATTR_OUTPUT_NTS, and ValuePtr is SQL_FALSE.

Description

The application can call SQLSetEnvAttr only when connection handle allocated to the environment handl e does not exist. The environment attributes which is set by the application persist until SQLFreeHandle is called to the environment handle. It is recommended to allocate and use only one environment handle.

The information format set through ValuePtr is dependent on the specified attribute. SQLSetEnvAttr acce pts one attribute information of two different formats which are the null-termination string or 32 bits int eger value. Each attribute format is described in the description of the attribute.

The driver-specific environment attributes does not exist.

The connection attribute can not be set with SQLSetEnvAttr. If it is tried, SQLSTATE HY092 (Invalid attrib ute/option identifier) will be returned.

Attribute	ValuPtr contents
SQL_ATTR_CONNECTION_POOLING (ODBC 3.8)	It is not supported by the driver.
SQL_ATTR_CP_MATCH (ODBC 3.0)	It is not supported by the driver.
SQL_ATTR_ODBC_VERSION (ODBC 3.0)	32-bit integer which indicates whether a particular feature is operated as ODBC 2.x or ODBC 3.x. The following values are used to set the attribute. • SQL_OV_ODBC3_80: The driver manager and driver exhibit the following ODBC 3.8 behaviors. ◦ The driver expects and returns ODBC 3.x code value for DATE, TIME, TIMESTAMP. ◦ The driver returns ODBC 3.x SQLSTATE codes when SQLError, SQLGetDiagField or SQLGetDiagRec is called. ◦ CatalogName argument of SQLTables allows the pattern search. • SQL_OV_ODBC3: The driver manager and driver exhibit the following ODBC 3.x behaviors. ◦ The driver expects and returns ODBC 3.x code value for DATE, TIME, TIMESTAMP. ◦ The driver returns ODBC 3.x SQLSTATE codes when SQLError, SQLGetDiagField or SQLGetDiagRec is called. ◦ CatalogName argument of SQLTables allows the pattern search. ◦ The driver manager does not support the extensibility of C data types. • SQL_OV_ODBC2: The driver manager and driver exhibit the following ODBC 2.x behaviors. It is very useful when the ODBC 2.x application operates in ODBC 3.x. ◦ The driver expects and returns ODBC 2.x code value for DATE, TIME, TIMESTAMP. ◦ The driver returns ODBC 2.x SQLSTATE codes when SQLError, SQLGetDiagFi

Attribute	ValuPtr contents
	eld or SQLGetDiagRec is called. ◦ CatalogName argument of SQLTables does not allow the pattern search. ◦ The driver manager does not support the extensibility of C data types. The application should set the environment attribute value before calling any function which has SQLHENV argument, or it returns SQLSTATE HY010 (Function sequence err or). It is driver-specific whether additional behavior exists for these environmental flag s.
SQL_ATTR_OUTPUT_NTS (ODBC 3.0)	It is not supported by the driver.

SQLSetParam

Conformance

Introduced version: ODBC 1.0
Standards compliance: It is not available.

Overview

ODBC 1.0 function SQLSetParam is replaced with ODBC 2.0 function SQLBindParameter. For more information, refer to **SQLBindParameter**.

SQLSetPos

Conformance

Introduced version: ODBC 1.0
Standards compliance: ODBC

Overview

SQLSetPos sets the cursor position in the row set and allows the application to update the data in the ro w set or to update or delete the data in the result set.

Syntax

```
SQLRETURN SQLSetPos(
    SQLHSTMT        StatementHandle,
    SQLSETPOSIROW   RowNumber,
    SQLUSMALLINT    Operation,
    SQLUSMALLINT    LockType);
```

Arguments

StatementHandle

[Input] It is the statement handle.

RowNumber

[Input] It is the row position in the row set which is specified with Operation argument to impleme nt the operation. If RowNumgber is 0, the operation is applied to all rows in the row set.

For more information, refer to **Description** section.

Operation

[Input] The operations to perform are SQL_POSITION, SQL_REFRESH, SQL_UPDATE, SQL_DELETE.

SQL_ADD value of Operation argument is not used in ODBC 3.x any more. The ODBC 2.x driver wil l be required to support SQL_ADD for the backward compatibility. This feature is replaced by callin g SQLBulkOperaions with operation of SQL_ADD. When ODBC 3.x application is performed with O DBC 2.x driver, the driver manager maps a call of SQLBulkOperations with an operation of SQL_AD D to a call of SQLSetPos with an operation of SQL_ADD.

For more information, refer to **Description** section.

LockType

[Input] The way to lock the row is specified after implementing the operation specified in Operation argument.

SQL_LOCK_NO_CHANGE, SQL_LOCK_EXCLUSIVE, SQL_LOCK_UNLOCK

For more information, refer to **Description** section.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_NEED_DATA, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01000	General warning	It is the driver-specific informational message. (The function returns SQL_SUCCESS_WITH_INFO.)
01001	Cursor operation conflict	Operation argument is SQL_DELETE or SQL_UPDATE. One or more rows are deleted/ updated, or any row is not deleted/ updated. Operation argument is SQL_DELETE or SQL_UPDATE, and optimistic concurrency causes the operation failure. (The function returns SQL_SUCCESS_WITH_INFO.)
01004	String data, right truncation	Operation argument is SQL_REFRESH, and string or binary data returned for a column(s) with a data type of SQL_C_CHAR or SQL_C_BINARY truncates non white space character or non-NULL binary data.
01S01	Error in row	RowNumber argument is 0, and an error occurred in one or more rows while the operation which is specified in operation argument is executed. (If an error occurs on one or more, but not all, rows of a multirow operation, SQL_SUCCESS_WITH_INFO is returned. If an error occurs on a single row operation, SQL_ERROR is returned) (If the driver is ODBC 2.x and the cursor library is not used, then this SQLSTATE occurs only when SQLSetPos is called after SQLExtendedFetch.)
01S07	Fractional truncation	If an operation argument is SQL_REFRESH and the buffer type of the application is not SQL_C_CHAR or SQL_C_BINARY, and the data returned to the application buffer for one or more columns are truncated. For the numeric data types, the fractional parts of the number are truncated. For time, timestamp, and interval data types including a time component, the fractional parts of the time are truncated.
07006	Restricted data type attribute viola	The column data value of the result set can not be converted to the data type specified in TargetType when SQLBindCol is called.

SQLSTATE	Error	Description
	tion	
07009	Invalid descriptor index	Operation argument is SQL_REFRESH or SQL_UPDATE, and the column value which is bigger than the number of columns in the result set is bound.
21S02	Degree of derived table does not match column list	Operation argument is SQL_UPDATE, and updatable column does not exist because all bound column are released, read-only or the value in bound length/indicator buffer is SQL_COLUMN_IGNORE.
22001	String data, right truncation	Operation argument is SQL_UPDATE, and the assignment of a character or binary value to a column causes the truncation of a non empty string (for character), or non-null string (for binary) or bytes.
22003	Numeric value out of range	Operation argument is SQL_UPDATE, and the assignment of a numeric value to a column in the result set causes the truncation of whole part of the number. Operation argument is SQL_REFRESH, and returning the numeric value for one or more bound column causes the loss of significant digits.
22007	Invalid datetime format	Operation argument is SQL_UPDATE, and the assignment of date or timestamp value to a column in the result set causes the year, month, day field to be out of range. Operation argument is SQL_REFRESH, and returning date or timestamp value for one or more bound column causes the year, month, day field to be out of range.
22008	Date/time field overflow	Operation argument is SQL_UPDATE, and the performance of datetime arithmetic on data being sent to a column in the result set causes a datetime field (the year, month, day, hour, minute, or second field) of the result being outside the permissible range of values for the field, or being invalid based on the Gregorian calendar's natural rules for datetimes. Operation argument is SQL_REFRESH, and the performance of datetime arithmetic on data being retrieved from the result set causes a datetime field (the year, month, day, hour, minute, or second field) of the result being outside the permissible range of values for the field, or being invalid based on the Gregorian calendar's natural rules for datetimes.
22015	Interval field overflow	The Operation argument is SQL_UPDATE, and assigning an exact numeric or interval C type to an interval SQL data type causes a loss of significant digits. The Operation argument is SQL_UPDATE, and when assigning to an interval SQL type, representation of the value of the C type does not exist in the interval SQL type. The Operation argument is SQL_REFRESH, and assigning from an exact nu

SQLSTATE	Error	Description
		meric or interval SQL type to an interval C type causes a loss of significant di gits in the leading field. The Operation argument is SQL_ REFRESH, and when assigning to an interv al C type, representation of the value of the SQL type does not exist in the in terval C type.
22018	Invalid character value for cast spe cification	Operation argument is SQL_REFRESH, and C type is the exact or approximat e numeric, datetime or interval data types. The SQL type of the column is a c haracter data type. And the column value is not a valid character for the bo und C type. Operation argument is SQL_UPDATE, and the SQL type is the exact or appro ximate numeric, datetime or interval data type. The C type of the column is SQL_C_CHAR. And the column value is not a valid character for the bound SQL type.
23000	Intergrity constrai nt violation	Operation argument is SQL_DELETE or SQL_UPDATE, and it violates the inte grity constraints.
24000	Invalid cursor stat e	StatementHandle is in the executed state but the result is not related to Stat ementHandle. The cursor is open to StatementHandle, but SQLFetch or SQLFetchScroll is n ot called. The cursor is open to StatementHandle and SQLFetch or SQLFetchScroll is ca lled. But the cursor is positioned before the starting point of the result set or after the end of the result set. Operation argument is SQL_DELETE, SQL_REFRESH or SQL_UPDATE.
40001	Serialization failu re	The transaction is rolled back due to a resource deadlock with another trans actions.
40003	Statement compl etion unknown	The connection is failed during the function execution, and the status of tra nsaction can not be determined.
42000	Syntax error or ac cess violation	The driver can not lock the rows as needed to implement the operation req uested in the operation argument. The driver can not lock the rows as requested in the LockType argument.
44000	WITH CHECK OP TION violation	Operation argument is SQL_UPDATE, and update on a viewed table or the t able derived from the viewed table which was created by specifying WITH_C HECK OPTION is performed to prevent the existence of one or more rows af fected by the update in the viewed table.
HY000	General error	It is an error without specific SQLSTATE, and the error message returned by SQLGetDiagRec in *MessageText buffer describes error message and its cau se.

SQLSTATE	Error	Description
HY001	Memory allocation error	The driver can not allocate the memory required for execution or completion of the function.
HY008	Operation canceled	The asynchronous processing is activated for StatementHandle. SQLCancel or SQLCancelHandle is called on StatementHandle before the function is called and completed, and the function is called again on StatementHandle. The function is called, and before it completed execution, SQLCancel or SQLCancelHandle is called on the StatementHandle from a different thread in a multithread application.
HY010	Function sequence error	The asynchronously executing function is called for the StatementHandle when SQLSetPos was executed, and this function is still being executed when SQLSetPos is called. SQLExecute, SQLExecDirect or SQLMoreResults is called for StatementHandle, and SQL_PARAM_DATA_AVAILABLE is returned, and the function is called before the data was retrieved from the streamed parameters. The asynchronously executing function is called for the StatementHandle, and this function is still being executed when this function is called. SQLExecute, SQLExecDirect or SQLMoreResults is executed in StatementHandle, and SQL_NEED_DATA is returned, and the function is executed before all data is sent. The driver is ODBC 2.x, and SQLSetPos is called in StatementHandle after SQFetch is called.
HY011	Attribute cannot be set now	The driver version is ODBC 2.x, SQL_ATTR_ROW_STATUS_PTR statement attribute is set, and SQLSetPos is called before SQLFetch, SQLFetchScroll or SQLExtendedFetch is called at the time.
HY013	Memory management error	The function is not executed because the underlying memory object can not be accessed because of the low memory condition.
HY090	Invalid string or buffer length	Operation argument is SQL_UPDATE.
HY092	Invalid attribute identifier	The value which is specified in Operation argument is not valid. The value which is specified in LockType argument is not valid. Operation argument is SQL_UPDATE or SQL_DELETE, and SQL_ATTR_CONCURRENCY statement attribute is SQL_ATTR_CONCUR_READ_ONLY.
HY107	Row value out of range	The value which is specified in RowNumber argument is bigger than the number of rows in the row set.
		The cursor can not be positioned in the row set because the cursor which is

SQLSTATE	Error	Description
HY109	Invalid cursor position	related to StatementHandle is set to forward-only. For more information, refer to the description of SQL_ATTR_CURSOR_TYPE attribute of SQLSetStmtAttr. Operation argument is SQL_UPDATE, SQL_DELETE or SQL_REFRESH, and the row which is identified by RowNumber argument is deleted or returned. RowNumber argument is 0, and operation argument is SQL_POSITION. SQLSetPos is called after SQLBulkOperations is called and before SQLFetchScroll or SQLFetch is called.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	For more information about the suspended state, refer to **SQLEndTran**.
HYC00	Optional feature not implemented	The driver or data source does not support the operation requested in operation argument or LockType argument.
HYT00	Timeout expired	The query timeout period is expired before returning the entire result set from the data source. The timeout can be set via SQL_ATTR_QUERY_TIMEOUT of SQLSetStmtAttr.
HYT01	Connection timeout expired	The connection is expired before the data source responds to the request. The timeout can be set via SQL_ATTR_CONNECTION_TIMEOUT of SQLSetStmtAttr.
IM001	Driver does not support this function	The driver which is related to StatementHandle does not support the function.
IM017	Polling is disabled in asynchronous notification mode	Whenever using the notification model, polling can not be used.
IM018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	If the previous function call for the handle returns SQL_STILL_EXECUTING and if the notification mode is activated, SQLCompleteAsync should be called for the handle to do post processing and complete the operation.

Description

RowNumber Argument

RowNumber argument specifies the number of rows in the row set affected by the operation implementation specified in Operation argument. If RowNumber is 0, it is applied to all rows in the row set. RowNumber should be from 0 up to the number of rows in the row set.

The array is 0-based 0 in C language but the RowNumber is 1-based. For example, when updating the fifth row in the row set, the application updates the row set buffer of the array index 4, but RowNumber specifies to 5.

All operations position the cursor on the row specified by RowNumber. The following operation requires the cursor position.

- Positioned update or delete statements
- Calling SQLGetData
- Calling SQLSetPos with the SQL_DELETE, SQL_REFRESH or SQL_UPDATE option

Operation Argument

Operation argument supports the following operations. The application calls SQLGetInfo with the information type of SQL_DYNAMIC_CURSOR_ATTRIBUTES1, SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1, SQL_KEYSET_CURSOR_ATTRIBUTES1 or SQL_STATIC_CURSOR_ATTRIBUTES1 (depending on the cursor type.) to verify the options supported by the data source.

Operation argument	Operation
SQL_POSITION	The driver positions the cursor at the row specified in RowNumber. The content of row status array which is pointed by SQL_ATTR_ROW_OPERATION_PTR statement attribute is ignored by SQL_POSITION.
SQL_REFRESH	It is not supported by the driver.
SQL_UPDATE	It is not supported by the driver.
SQL_DELETE	It is not supported by the driver.

LockType Argument

LockType argument provides the method for the application to control the concurrency. In most case, the data source which provides the concurrency level and transaction supports SQL_BLOCK_NO_CHANGE value to LockType argument.

LockType argument specifies the lock state of rows after executing SQLSetPos. If the driver can not lock the row either to perform the requested operation or to satisfy the LockType argument, it returns SQL_ERROR and SQLSTATE 42000(Syntax error or access violation).

Although LockType argument is specified in a statement, it is applied to all statements of the same privile ges on the connection. The special lock which is acquired by a statements on the connection can be unlo cked by another statement on the same connection.

A row locked through SQLSetPos remains locked until the application calls SQLSetPos for the row with Lo ckType set to SQL_LOCK_UNLOCK, or until the application calls SQLFreeHandle for the statement or SQLF reeStmt with the SQL_CLOSE option. When the driver supports the transaction, the row lock through SQ LSetPos is unlocked when the application commits or rolls back the transaction on the connection by calli ng SQLEndTran. (If the cursor is closed when a transaction is committed or rolled back as indicated by the of SQL_CURSOR_COMMIT_BEHAVIOR and SQL_CURSOR_ROLLBACK_BEHAVIOR information types retur ned by SQLGetInfo.)

LockType argument supports the following lock types. The application calls SQLGetInfo with the SQL_DY NAMIC_CURSOR_ATTRIBUTES1, SQL_FORWARD_ONLY_CURSOR_ATTRIBUTES1, SQL_KEYSET_CURSOR _ATTRIBUTES1 or SQL_STATIC_CURSOR_ATTRIBUTES1 information type to verify the lock supported by t he data source.

LockType argument	Lock type
SQL_LOCK_NO_CHANGE	The driver or data source ensures the locking or unlocking of the same row as it was b efore SQLSetPos is called. The value of LockType does not allow explicit row-level locki ng so that any lock required by the current concurrency and transaction isolation level s can be used.
SQL_LOCK_EXCLUSIVE	The driver or data source performs the exclusive lock on a row. The statement on oth er connection or in other application can not be used to acquire any locks on the row.
SQL_LOCK_UNLOCK	The driver or data source releases the lock.

When the update and delete operations are performed in SQLSetPos, the application uses the following L ockType argument.

- An application calls SQLSetPos with operation set to SQL_REFRESH, and LockType set to SQL_LOCK_ EXCLUSIVE to guarantee that the row is not updated after it is retrieved.
- If the application sets LockType to SQL_LOCK_NO_CHANGE, the driver guarantees the update or del ete operation only when the application specifies SQL_CONCUR_LOCK for SQL_ATTR_CONCURRENC Y statement attribute.
- If the application specifies SQL_CONCUR_ROWVER or SQL_CONCUR_VALUES for SQL_ATTR_CONCU RRENCY statement attribute, the driver compares the version or value of the row and denies the oper ation when the row is changes since the application fetched the row.
- If the application specifies SQL_CONCUR_READ_ONLY for SQL_ATTR_CONCURRENCY statement attr ibute, the driver denies any update or delete operation.

For more information about SQL_ATTR_CONCURRENCY statement attribute, refer to **SQLSetStmtAttr**.

Status and Operation Array

The following status and operation arrays are used when SQLSetPos is called.

- The row status array includes the status value of each row data in the row set. The driver sets the stat us value in the array after calling SQLFetch, SQLFetchScroll, SQLBulkOperations or SQLSetPos. The arr ay is pointed to by SQL_ATTR_ROW_STATUS_PTR statement attribute.
- The row operation array includes each row value in the row set which indicates whether SQLSetPos c all for the bulk operation is ignored or performed. Each element in the array sets one of SQL_ROW_P ROCEED or SQL_ROW_IGNORE. The array is pointed to by SQL_ATTR_ROW_OPERATION_PTR statem ent attribute.

The number of elements in the status and operation array should be equal to the number of rows in the r ow set (as defined by the SQL_ATTR_ROW_ARRAY_SIZE statement attribute).

For more information about the row status array, refer to **SQLFetch**.
For more information about the row operation array, refer to **Ignoring Rows in the Bulk Operation**.

Using SQLSetPos

The application should perform the following steps before using SQLSetPos.

1. If the application calls SQLSetPos with operation set to SQL_UPDATE, call SQLBindCol (or SQLSetDes cRec) for each column to specify its data type and bind buffers for the column's data and length.
2. If the application calls SQLSetPos with operation set to SQL_DELETE or SQL_UPDATE, call SQLColAttr ibute to make sure that the columns to be deleted or updated are updatable.
3. The result set is generated by calling SQLExecDirect, SQLExecute or the catalog function.
4. The data is retrieved by calling SQLFetch or SQLFetchScroll.

Deleing Data Using SQLSetPos

An application calls SQLSetPos with RowNumber set to the number of the row to delete and calls operati on set to SQL_DELETE, to delete data by using SQLSetPos.

After the data is deleted, the driver changes the value in the implementation row status array for the app ropriate row to SQL_ROW_DELETED (or SQL_ROW_ERROR).

Updating Data Using SQLSetPos

The application can pass the data to the column through one of the bound buffer or once or more SQLPu tData calls. Columns whose data is passed with SQLPutData are known as data-at-execution columns. Us ually it may be used to transfer data to SQL_LONGVARBINARY and SQL_LONGVARCHAR and it can be m ixed with other columns.

Updating Data Using SQLSetPos in Application

1. Place value in the data and length/indicator buffer bound with SQLBindCol.

- For general columns, the application positions the new column value in *TargetValuPtr buffer and th e value length in *StrLen_or_IndPtr buffer. If the row should not be updated, the application position s SQL_ROW_IGNORE in the row element in the row operation array.
- For data-at-execution columns, the application positions an application-defined value, such as the col umn number, in the *TargetValuePtr buffer. The value can be used later to identify the column.
  The application positions the macro result of SQL_LEN_DATA_AT_EXEC(length) in *StrLen_or IndPtr buffer. If the SQL data type of the column is SQL_LONGVARBINARY, SQL_LONGVAR_CHAR or the so urce-defined long data type and the driver returns Y to SQL_NEED_LONG_DATA information type of SQLGetInfo, length is the number of bytes to be sent to the parameter. Otherwise, it is not a negative and it should be ignored.

2. Call SQLSetPos with the operation argument set to SQL_UPDATE to update the row of data.

- If data-at-execution columns do not exist, the processing is completed.
- If data-at-execution columns exist, the function returns SQL_NEED_DATA and proceeds the step 3.

3. Call SQLParamData to retrieve the address of *TargetValuePtr buffer for the first data-at-execution c olumn to be processed. SQLParamData returns SQL_NEED_DATA. The application retrieves the appli cation-defined value in *TargetValuPtr buffer.

- Though data-at-execution parameter is similar to the data-at-execution column, but each v alue which is returned by SQLParamData is different.

- Data-at-execution parameters are parameters in an SQL statement for which data will be se nt with SQLPutData when the statement is executed with SQLExecDirect or SQLExecute. Th ey are bound to SQLBindParameter or SQLSetDescRec by setting the descriptors. The value which is returned by SQLParamData is the 32 bits value and it is sent to SQLBindParameter of ParameterValuePtr argument.

- Data-at-execution columns are columns in a row set for which data will be sent with SQLPu tData when a row is updated with SQLSetPos. They are bound with SQLBindCol. The value r eturned by SQLParamData is the address of the row in the *TargetValuePtr buffer which is being processed.

4. Call SQLPutData once or more times to transfer data to the column. If all data values can not be retu rned in *TargetValuePtr buffer specified in SQLPutData, multiple calls are needed. Only when the ch aracter data of C language is sent to the column with a character, binary or a data source data type, or when the binary data of C language is sent to to the column with a character, binary or a data so

urce data type, then multiple calls of SQLPutData are allowed for the same column.

5. Call SQLParamData again to indicate all data are sent to the column.

- If more data-at-execution columns exist, then SQLParamData returns SQL_NEED_DATA and the addr ess of the TargetValuePtr buffer for the next data-at-execution column to be processed. The applicati on repeats the steps 4 and 5.
- If data-at-execution columns do not exist any more, then the processing is completed. If the stateme nt is successfully executed, SQLParamData returns SQL_SUCCESS or SQL_SUCCESS_WITH_INFO. If ex ecution is failed, SQL_ERROR is returned. At this point, SQLParamData returns any SQLSTATE which c an be returned by SQLSetPos.

If the data is updated, the driver changes the value in the implementation row status array for the approp riate row to SQL_ROW_UPDATED.

If the operation is canceled or an error occurs in SQLParamData or SQLPutData, after SQLSetPos returns S QL_NEED_DATA and before data is transferred for all data-at-execution columns, the application can call only SQLCancel, SQLGetDiagField, SQLGetDiagRec, SQLGetFunctions, SQLParamData, or SQLPutData for the statement or the connection related to the statement. At this point, if another function is called, SQL _ERROR and SQLSATATE HY010 (Function sequence error) are returned.

If the application calls SQLCancel while the driver still needs the data of data-at-execution columns, the d river cancels the operation. The application can call SQLSetPos again later. Cancellation does not affect th e cursor state or the current cursor position.

When SELECT-list of the query specification related to the cursor includes one or more references in the s ame column, the driver defines whether an error occurs or the driver ignores the duplicated reference an d implements the requested operation.

Executing the Bulk Operation

If the RowNumber argument is 0, the driver performs the operation specified in the operation argument f or every row in the rowset which has a value of SQL_ROW_PROCEED in its field in the row operation arra y pointed to by SQL_ATTR_ROW_OPERATION_PTR statement attribute. This is a valid value of the RowNu mber argument for an operation argument of SQL_DELETE, SQL_REFRESH, or SQL_UPDATE, but not SQL _POSITION. SQLSetPos with an operation of SQL_POSITION and a RowNumber equal to 0 will return SQL STATE HY109 (Invalid cursor position).

If an error occurs in the entire row set, such as SQLSTATE HYT00 (Timeout expired), the driver returns SQ L_ERROR and the appropriate SQLSTATE. The contents of the row set buffers are undefined, and the curs or position is unchanged.

If an error occurs in a single row, the driver performs the followings.

- It sets the row element in the row status array pointed by SQL_ATTR_ROW_STATUS_PTR statement a

ttribute to SQL_ROW_ERROR.
- It adds one or more SQLSTATE to an error in the error queue, and sets SQL_DIAG_ROW_NUMBER fie ld of the diagnostic data structure.

If the driver completes the remaining row operation in the row set after an error or warning is processed, SQL_SUCCESS_WITH_INFO is returned. Therefore, the error queue includes zero or more SQLSTATE for t he rows which return the errors. The driver stops the operation after processing an error or warning and r eturns SQL_ERROR.

If the driver returns a warning such as SQLSTATE 01004 (Data truncated), then the driver returns a warni ng applied to the entire row set or unknown rows in the row set before returning the error information w hich is applied to a specific row. It returns warnings for specific rows together with other error informatio n about those rows.

If RowNumber is 0 and operation is SQL_UPDATE, SQL_REFRESH or SQL_DELETE, then the number of ro ws on which SQLSetPos operates is set by SQL_ATTR_ROWS_FETCHED_PTR statement attribute.

If RowNumber is 0 and the operation is SQL_DELETE, SQL_REFRESH or SQL_UPDATE, then the current ro w after the operation is as same as the row before the operation.

Ignoring Rows in the Bulk Operation

The row operation array can be used to specify the row which should be ignored in the current row set d uring the bulk operation with SQLSetPos. The application should perform the following steps to make the driver ignore one or more rows in the driver during the bulk operation.

1. It calls SQLSetStmtAttr to set the SQL_ATTR_ROW_OPERATION_PTR statement attribute to point to an array of SQLUSMALLINTs. This field can also be set by calling SQLSetDescField to set the SQL_DES C_ARRAY_STATUS_PTR header field of the ARD, which requires that an application obtains the desc riptor handle.

2. It sets each element in the row operation array to one of the following values.

- Setting to SQL_ROW_IGNORE it indicates that the row is excluded in the bulk operation.
- Setting to SQL_ROW_PROCEED it indicates that the row is included in the bulk operation.

3. It performs the bulk operation by calling SQLSetPos.

The following rules are applied to the row operation array.

- SQL_ROW_IGNORE and SQL_ROW_PROCEED affect only bulk operations using SQLSetPos with an o peration of SQL_DELETE or SQL_UPDATE. They do not affect calling SQLSetPos with an operation of SQL_REFRESH or SQL_POSITION.
- The pointer is set to NULL by default.
- If the pointer is NULL, then all rows are updated like as all elements are set to SQL_ROW_PROCEED.
- Setting the element to SQL_ROW_PROCEED does not guarantee occurrance of the operation in a par

ticular row. If a certain row in the row set has the SQL_ROW_ERROR status, the driver can not updat e that row regardless of whether the application specified SQL_ROW_PROCEED. The application sho uld always check the row status array to see whether the operation was successful.

- SQL_ROW_PROCEED is defined as 0 in the header file. The application can initialize the row operatio n array to 0 to process all rows.
- If the n-th element of the row status array is set to SQL_ROW_IGNORE and the bulk update or delete operation is executed by calling SQLSetPos, the n-th row in the row set remains unchanged after calli ng SQLSetPos.
- The application should automatically set a read-only column to SQL_ROW_IGNORE.

Ignoring Columns in the Bulk Operation

The application can set the value in the bound length/indicator buffer to SQL_COLUMN_IGNORE so that i t can avoid unnecessary processing diagnostics caused by attempting to update one or more read-only co lumns. For more information, refer to **SQLBindCol**.

SQLSetScrollOptions

Conformance

Introduced version: ODBC 1.0
Standards compliance: It is not available.

Overview

SQLSetScrollOptions function in ODBC 2.0 is replaced with calling SQLGetInfo and SQLSetStmtAttr in ODBC 3.x.

When the driver manager maps SQLSetScrollOptions for the application which operates with ODBC 3.x driver which does not support SqlSetScrollOptions, the driver manager sets SQL_ROW_SET_SIZE statement option, not the SQL_ATTR_ROW_ARRAY_SIZE statement attribute in RowsetSize argument of SQLSetScrollOptins. As a result, SQLSetScrollOptions can not be used by an application when fetching multiple rows by calling SQLFetch or SQLFetchScroll, and it can be used only when fetching multiple rows by calling SQLExtendedFetch.

SQLSetStmtAttr

Conformance

Introduced version: ODBC 3.0
Standards compliance: ISO 92

Overview

SQLSetStmtAttr sets the attributes related to the statements.

Syntax

```
SQLRETURN SQLSetStmtAttr(
    SQLHSTMT      StatementHandle,
    SQLINTEGER    Attribute,
    SQLPOINTER    ValuePtr,
    SQLINTEGER    StringLength);
```

Arguments

StatementHandle

[Input] It is the statement handle.

Attribute

[Input] It is the option to be set. For more information, refer to the **Description** section.

ValuePtr

[Input] It is the value which is related to the attribute. ValutPtr can be one of the followings accordi ng to the attribute value.

- ODBC descriptor handle
- SQLUINTEGER value
- SQLULEN value
- One of the following pointers

° Null-termination character string

° Binary buffer

° SQLLEN, SQLULEN, SQLUSMALLINT values or the array of values

° Driver-defined value

If Attribute argument is the driver-specific value, ValuePtr may be an integer.

StringLength

[Input] If an attribute is an ODBC-defined attribute and ValuePtr points to a string or binary buffer, the argument is the length of *ValuePtr. If Attribute is a ODBC-defined attribute and ValuePtr is an integer, the argument is ignored. StringLength may have one of the following values.

If Attribute is a driver-defined attribute, the application indicates the nature of the attribute to the driver manager by setting StringLength argument.

- If ValuePtr is a string pointer, Stringlength is the string length or SQL_NTS.
- If ValuePtr is a binary buffer, the application stores SQL_LEN_BINARY_ATTR (length) macro result in StringLength. StringLength has a negative value.
- If ValuePtr is a pointer of a different value other than a string or a binary buffer, StringLength must have SQL_IS_POINTER value.
- If ValuePtr includes a fixed length value, StringLength is SQL_IS_INTEGER or SQL_IS_UINTEGER.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01000	General warning	It is the driver-specific informational message. (The function returns SQL_SUCCESS_WITH_INFO.)
01S02	Option value changed	It is temporarily replaced with a similar value because the driver does not support the value specified in ValuePtr or the value specified in ValuePtr is invalid due to the implementation working conditions. (SQLGetStmtAttr can be called to see which value is temporarily replaced.) The replaced value is valid for the StatementHandle until the cursor is closed, and it is changed to the previous value when the cursor is closed. The statement attributes which can be changed are as follows. SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ATTR_KEYSET_SIZE, SQL_ATTR_MAX_LENGTH, SQL_ATTR_MAX_ROWS, SQL_ATTR_QUERY_TIMEOUT, SQL_ATTR_ROW_ARRAY_SIZE, SQL_ATTR_SIMULATE_CURSOR. (The function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure	Before the function processing is completed, the connection between the driver and the data source is failed.
24000	Invalid cursor state	Attribute is SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ATTR_SIMULATE_CURSOR or SQL_ATTR_USE_BOOKMARKS, and the cursor is open.

SQLSTATE	Error	Description
HY000	General error	It is an error without specific SQLSTATE, and the error message returned by SQLGetDiagRec in *MessageText buffer describes error message and its cause.
HY001	Memory allocation error	The driver can not allocate the memory required for execution or completion of the function.
HY009	Invalid use of null pointer	The statement attribute which is identified by Attribute argument requires the string attribute, and ValuePtr argument is a NULL pointer.
HY010	Function sequence error	The asynchronously executing function is called for the connection handle related to StatementHandle, and this function is still being executed when SQLSetStmtAttr is called. SQLExecute, SQLExecDirect or SQLMoreResults are called for StatementHandle, and SQL_PARAM_DATA_AVAILABLE is returned. This function is called before the data is returned for the streamed parameters. The asynchronously executing function is called for StatementHandle, and is still being executed when the function is called. SQLExecute, SQLExecDirect, SQLBulkOperation or SQLSetPos are called for StatementHandle, and SQL_NEED_DATA is returned. This function is called before data is sent for all data-at-execution parameters or columns.
HY011	Attribute cannot be set now	Attribute is SQL_ATTR_CONCURRENCY, SQL_ATTR_CURSOR_TYPE, SQL_ATTR_SIMULATE_CURSOR or SQL_ATTR_USE_BOOKMARKS, and the statement is prepared.
HY013	Memory management error	The function is not executed because the underlying memory object can not be accessed because of the low memory condition.
HY017	Invalid use of an automatically allocated descriptor handle	Attribute argument is SQL_ATTR_IMP_ROW_DESC or SQL_ATTR_IMP_PARAM_DESC. Attribute argument is SQL_ATTR_APP_ROW_DESC or SQL_ATTR_APP_PARAM_DESC, and the value in ValuePtr is an implicitly allocated descriptor handle other than the handle originally allocated for the ARD or APD.
HY024	Invalid attribute value	Considering the specified Attribute value, the value in ValuePtr is invalid. (The driver manager returns this SQLSTATE only for the statement attribute and connection which allows the separate set such as SQL_ATTR_ACCESS_MODE or SQL_ATTR_ASYNC_ENABLE. The driver should verify the value in ValuePtr for other connection and statement attribute.) Attribute argument is SQL_ATTR_APP_ROW_DESC or SQL_ATTR_APP_PARAM_DESC, and ValuePtr is an explicitly allocated descriptor handle which is not on the same connection as the StatementHandle argument.
HY090	Invalid string or buffer length	*ValuePtr is a string and StringLength is smaller than 0. (It is not SQL_NTS.)

SQLSTATE	Error	Description
HY092	Invalid attribute/ option identifier	The value specified for the Attribute argument is invalid for the version of ODBC supported by the driver. The value which is specified in Attribute is the read-only attribute.
HY117	Connection is suspended due to unknown tracsaction state. Only disconnect and read-only functions are allowed	For more information about the suspended state, refer to **SQLEndTran**.
HYC00	Optional feature not implemented	The value which is specified in Attribute argument is valid in ODBC version supported by the driver but it is not supported by the driver. Attribute argument is SQL_ATTR_ASYNC_ENABLE. When SQLGetInfo is called in InforType with SQL_ASYNC_MODE, then SQL_AM_CONNECTION is returned. Attribute argument is SQL_ATTR_ENABLE_AUTO_IPD, and the value of SQL_ATTR_AUTO_IPD connection attribute is SQL_FALSE.
HYT01	Connection timeout expired	The connection timeout period is expired before the data source response to the request. The connection timeout period can be set via SQLSetConnectAttr of SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support this function	The driver does not support the function.
S1118	Driver does not support asynchronous notification	When calling SQLSetStmtAttr to set SQL_ATTR_ASYNC_STMT_EVENT the driver does not support the asynchronous notification.

Description

The statement attribute is updated by calling SQLSetStmtAttr or it remains effective until the statement is deleted by SQLFreeHandle. Calling SQLFreeStmt with SQL_CLOSE, SQL_UNBIND or SQL_RESET_PARAMS option does not reset the statement attribute.

Some statements attributes is replaces with a similar value if the driver does not support the value specified in ValuePtr. The driver returns SQL_SUCCESS_WITH_INFO and SQLSTATE 01S02 (Option value changed) for these cases. For example, if attribute is SQL_ATTR_CONCURRENCY and ValuePtr is SQL_CONCUR_ROWVER and the data source does not support it, the driver replace it with SQL_CONCUR_VALUES and returns SQL_SUCCESS_WITH_INFO. The application calls SQLGetStmtAttr to obtain the replaced value.

The format of information set with ValuePtr depends on the value specified in Attribute. SQLSetStmtAttr

accepts one of attribute information types such as a different character string or integer. Each type is described in the description of the attribute. The type is applied to the information returned for each attribute in SQLGetStmtAttr. The length of the character string pointed to by the ValuePtr argument of SQLSetStmtAttr is StringLength.

- The feature whose statement attribute is set at the connection level by calling SQLSetConnectAttr may disappear in ODBC 3.x. The ODBC 3.x application should not set the statement attribute at the connection level. The ODBC 3.x statement attribute can not be set at the connection level except for SQL_ATTR_METADATA_ID, and SQL_ATTR_ASYNC_ENABLE attribute is both the connection attribute and the statement attribute, and it can be set at the connection level and the statement level.

- ODBC 3.x driver requires the option feature which sets the driver version to ODBC 2.x at the connection level if the driver should operate with the ODBC 2.x application.

Statement Attribute to Set the Descriptor Field

Many statement attributes correspond to the header fields of the descriptor. Setting these attributes is the same result with setting the descriptor fields. Setting fields by calling SQLSetStmtAttr rather than calling SQLSetDescField has the advantage that a descriptor handle does not have to be obtained for the function call.

Calling SQLSetStmtAttr in a single statement may affect other statements. It happens when APD or ARD is explicitly assigned to the statement and it is related to other statements. SQLSetStmtAttr is applied to all statements related to this descriptor because it modifies APD or ARD. If it is not required operation, the application should separate this descriptor with other statements (by calling SQLSetStmtAttr to set the SQL_ATTR_APP_ROW_DESC or SQL_ATTR_APP_PARAM_DESC field to a different descriptor handle) before calling SQLSetStmtAttr again.

When the descriptor field is set as a result of the corresponding statement attribute being set, the field is set only for the applicable descriptors which are currently related to the statement identified by the StatementHandle argument, and the attribute setting does not affect any descriptor of the related statement in the future. When the descriptor field which is related to the statement handle is set to SQLSetDescField, the corresponding statement attribute is also set. If an explicitly allocated descriptor is dissociated from a statement, a statement attribute which corresponds to a header field will revert to the value of the field in the implicitly allocated descriptor.

When the statement is allocated, four descriptor handles are automatically allocated and is related to the statement. Explicitly allocated descriptor handles can be associated with the statement by calling SQLAllocHandle with an HandleType of SQL_HANDLE_DESC to allocate a descriptor handle and then calling SQLS

etStmtAttr to associate the descriptor handle with the statement.

The following is the statement attributes corresponding to the descriptor header field.

Statement attribute	Header field	Description
SQL_ATTR_PARAM_BIND_OFFSET_PTR	SQL_DESC_BIND_OFFSET_PTR	APD
SQL_ATTR_PARAM_BIND_TYPE	SQL_DESC_BIND_TYPE	APD
SQL_ATTR_PARAM_OPERATION_PTR	SQL_DESC_ARRAY_STATUS_PTR	APD
SQL_ATTR_PARAM_STATUS_PTR	SQL_DESC_ARRAY_STATUS_PTR	IPD
SQL_ATTR_PARAMS_PROCESSED_PTR	SQL_DESC_ROWS_PROCESSED_PTR	IPD
SQL_ATTR_PARAMSET_SIZE	SQL_DESC_ARRAY_SIZE	APD
SQL_ATTR_ROW_ARRAY_SIZE	SQL_DESC_ARRAY_SIZE	ARD
SQL_ATTR_ROW_BIND_OFFSET_PTR	SQL_DESC_BIND_OFFSET_PTR	ARD
SQL_ATTR_ROW_BIND_TYPE	SQL_DESC_BIND_TYPE	ARD
SQL_ATTR_ROW_OPERATION_PTR	SQL_DESC_ARRAY_STATUS_PTR	ARD
SQL_ATTR_ROW_STATUS_PTR	SQL_DESC_ARRAY_STATUS_PTR	IRD
SQL_ATTR_ROWS_FETCHED_PTR	SQL_DESC_ROWS_PROCESSED_PTR	IRD

Statement Attributes

The following table describes currently defined attributes and the version of ODBC in which they were int roduced.

Attribute	ValuePtr contents
SQL_ATTR_APP_PARAM_DESC (ODBC 3.0)	It is not supported by the driver.
SQL_ATTR_APP_ROW_DESC (ODBC 3.0)	It is not supported by the driver.
SQL_ATTR_ASYNC_ENABLE (ODBC 1.0)	It is not supported by the driver.
SQL_ATTR_ASYNC_STMT_EVENT (ODBC 3.8)	It is not supported by the driver.
SQL_ATTR_ASYNC_STMT_PCALLBACK (ODBC3.8)	It is not supported by the driver.
SQL_ATTR_ASYNC_STMT_PCONTEXT (ODBC 3.8)	It is not supported by the driver.
SQL_ATTR_ATOMIC_EXEC	SQLUSMALLINT: It is whether an atomic insert operation is operable.

Attribute	ValuePtr contents
UTION	• SQL_ATOMIC_EXECUTION_OFF • SQL_ATOMIC_EXECUTION_ON
SQL_ATTR_CONCURRENCY (ODBC 2.0)	The default value of SQL_ATTR_CONCURRENCY is SQL_CONCUR_READ_ONLY. • SQLULEN: It is the value which specifies the cursor concurrency. ◦ SQL_CONCUR_READ_ONLY: The cursor is read-only and the update is not allowed. ◦ SQL_CONCUR_LOCK: The cursor uses the minimum level locking enough to completely update the row. ◦ SQL_CONCUR_POWVER: The cursor uses the concurrency to control and compare the row version, such as SQLBase ROWID or Sybase TIMESTAMP. ◦ SQL_CONCUR_VALUES: The cursor uses the concurrency to control and compare the values. If SQL_ATTR_CURSOR_TYPE attribute is changed to the value which is not supported in SQL_ATTR_CONCURRENCY, the value of SQL_ATTR_CONCURRENCY may be changed at the execution time and a warning may occur when SQLExecDirect or SQLPrepare is called. While the driver supports SELECT FOR UPDATE statement and the statement is executed, an error occurs if SQL_ATTR_CONCURRENCY is changed to SQL_CONCUR_READ_ONLY. If SQL_ATTR_CONCURRENCY value is changed to SQL_ATTR_CURSOR_TYPE value which is supported by the driver, or to the value not supported by the driver, SQL_ATTR_CURSOR_TYPE value is changed at the execution time, and SQLSTATE 01S02 (Option value changed) may be returned when SQLExecDirect or SQLPrepare is executed. If the specified concurrency is not supported by the data source, the driver replaces it with other concurrency and returns SQLSTATE 01S02 (Option value changed). The driver replaces SQL_CONCUR_VALUES with SQL_CONCUR_ROWVER, or reverse. Also, SQL_CONCUR_LOCK is replaced with SQL_CONCUR_ROWVER, SQL_CONCUR_VALUES order. The validity of the replaced value is not confirmed until the execution time.
SQL_ATTTR_CURSOR_SCROLLABLE (ODBC 3.0)	• SQLULEN: It is the value specifying the required support level of the application. Setting the attribute affects the next SQLExecDirect and SQLExecute. ◦ SQL_NONSCROLLABLE: It is the default value. The scroll cursor is not required to the statement handle. If the application calls SQLFetchScroll to the handle, the only valid value of FetchOrientation is SQL_FETCH_NEXT. ◦ SQL_SCROLLABLE: The scroll cursor is required to the statement handle. When SQLFetchScroll is called, the application positions the cursor on the mode except the sequential mode and the valid value of FetchOrientation may be specified.
	• SQLULEN: The value which determines whether the cursor of the statement handle can view the change in the result set generated by the other cursor. Setting t

Attribute	ValuePtr contents
SQL_ATTR_CURSOR_SENSITIVITY (ODBC 3.0)	he field affects the next calls of SQLExecDirect and SQLExecute. The application can read again the attribute value to obtain the initial state or its latest state set by the application. ◦ SQL_UNSPECIFIED: It is the default value. It is unspecified what the cursor type is and whether cursors on the statement handle make visible the changes made to a result set by another cursor. The cursor of the statement handle can probably view none, some, or all such changes. ◦ SQL_INSENSITIVE: All cursors of the statement handle can view only the result but they can not view any changes made by the other cursor. The insensitive cursor is read-only. It corresponds to a static cursor with a read-only concurrency. ◦ SQL_SENSITIVE: It allows all cursors of the statement handle can view all of the changes in the result set generated by the other cursor.
SQL_ATTR_CURSOR_TYPE (ODBC 2.0)	• SQLULEN: It is the value which specifies the cursor type. ◦ SQL_CURSOR_FORWARD_ONLY: The cursor can only move forward. ◦ SQL_CURSOR_KEYSET_DRIVEN: The driver stores and uses the keys as many as the number of rows specified in SQL_ATTR_KEYSET_SIZE statement attribute. ◦ SQL_CURSOR_DYNAMIC: The driver stores and uses the key for only the row in the row set. The default value is SQL_CURSOR_FORWARD_ONLY. The attribute can not be specified in the prepared SQL statement. If the specified cursor type is not supported by the data source, the driver replaces it with the other type and returns SQLSTATE 01S02 (Option value changed. The driver replaces the mixed or dynamic cursor with the keyset-driven or static cursor. The driver replaces a keyset-driven cursor with a static cursor.
SQL_ATTR_ENABLE_AUTO_IPD (ODBC 3.0)	It is not supported by the driver.
SQL_ATTR_EXPLAIN_PLAN_OPTION	SQLUSMALLINT: Whether to create the plan information. • SQL_EXPLAIN_PLAN_OFF: Plan information is not generated. • SQL_EXPLAIN_PLAN_ON: The SQL statement is performed and plan information is generated. • SQL_EXPLAIN_PLAN_ONLY: The SQL statement is not performed and plan information is generated.
SQL_ATTR_EXPLAIN_PLAN_TEXT	It is the generated plan string. (read-only)
SQL_ATTR_FETCH_BOOKMARK_PTR (ODBC 3.0)	It is not supported by the driver.
SQL_ATTR_FETCH_FAILOV	SQLUSMALLINT: It is whether to use fetch failover.

Attribute	ValuePtr contents
ER	• SQL_FETCH_FAILOVER_OFF: It does not use fetch failover. • SQL_FETCH_FAILOVER_ON: It uses fetch failover.
SQL_ATTR_IMP_PARAM_DESC (ODBC 3.0)	It is not supported by the driver.
SQL_ATTR_IMP_ROW_DESC (ODBC 3.0)	It is not supported by the driver.
SQL_ATTR_KEYSET_SIZE (ODBC 2.0)	It is not supported by the driver.
SQL_ATTR_MAX_LENGTH (ODBC 1.0)	It is not supported by the driver.
SQL_ATTR_MAX_ROWS (ODBC 1.0)	• SQLULEN: It is the value corresponding to the maximum number of rows returned by SELECT statement. If *ValuePtr is equal to 0, the driver returns all rows. The purpose of this attribute is to reduce the network traffic. Notionally, it is applied when the result set is generated, and it limits the result set of the first ValuePtr rows. If the number of rows in the result set are bigger than ValuePtr, the result set is reduced. SQL_ATTR_MAX_ROWS is applied to all result set of the statement including those which are returned by the catalog functions. SQL_ATTR_MAX_ROWS sets the maximum number of cursor rows. If it is not sure if the SQL_ATTR_MAX_ROWS is to be properly implemented (If the limit of the result set size can not be implemented in the data source), the driver should not emulate the SQL_ATTR_MAX_ROWS operation for SQLFetch or SQLFetchScroll. The driver defines whether SQL_ATTR_MAX_ROWS is applied to the statements except for SELECT statement (such as the catalog functions). The attribute value is set on the cursor which is open. But it would not be immediately effective. In this case, the driver returns SQLSTATE 01S02 (Option value changed) and sets the attribute to its original value.
SQL_ATTR_METADATA_ID	• SQLULEN: It is the value which specifies the way to process the string arguments of the catalog functions. If it is SQL_TRUE, catalog functions treates a string argument as an identifier. It is not case-sensitive. The driver removes the trailing spaces of the string whose range is not determined and converts them to uppercase. The driver removes the leading and trailing spaces of the strings whose range is determined and takes the string literally between delimiters. If one of the argument is set to a NULL pointer, the function returns SQL_ERROR and SQLSTATE HY009. (Invalid use of null pointer).

Attribute	ValuePtr contents
(ODBC 3.0)	If it is SQL_FALSE, catalog functions does not treat a string argument as an identifier. I t is case-sensitive. They can either include a string search pattern or not, depending o n the argument. The default value is SQL_FALSE. It is the TableType argument of SQLTables which takes a list of values, is not affected by this attribute. SQL_ATTR_METADATA_ID can be set in the connection level. (SQL_ATTR_METADAT A_ID and SQL_ATTR_ASYNC_ENABLE are unique, and they are the statement attribut es and at the same time they are the connection attributes.) For more information, refer to **Arguments of Catalog Function**.
SQL_ATTR_NOSCAN (ODBC 1.0)	It is not supported by the driver.
SQL_ATTR_PARAM_BIND_ OFFSET_PTR (ODBC 3.0)	• SQLULEN*: It is the value which indicates the offset added to a pointer to chang e the binding of the dynamic parameters. If the field is not NULL, the driver dere ferences the pointer and adds the dereferenced value to each of the deferred fie lds in the descriptor record (SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, and SQL_DESC_OCTET_LENGTH_PTR), and it uses a new pointer value when bin ding. The default value is set to NULL. The bind offset is always directly added to SQL_DESC_DATA_PTR, SQL_DESC_INDICA TOR_PTR, SQL_DESC_OCTET_LENGTH_PTR fields. If the offset is changed to a differe nt value, the new value is still directly added to the descriptor field value. A new offset is not added to the field value plus previous offsets. Setting this statement attribute sets the SQL_DESC_BIND_OFFSET_PTR field in the AP D header.
SQL_ATTR_PARAM_BIND_ TYPE (ODBC 3.0)	• SQLULEN: It represents the binding direction used for the dynamic parameter. The field is set to SQL_PARAM_BIND_BY_COLUMN which is for the column-wise bind ing. (The default value) To select the row-wise binding, the field is set to the structure length or the buffer ins tance which is bound to the dynamic parameter set . If the length includes the space f or the padding of all the bound parameter and structure or the address of the bound parameter is increased to the specified length, the result should be buffered to point t o the beginning of the next parameter. Using sizeof operator of ANSI C guarantees th is operation. Setting this statement attribute sets the SQL_DESC_BIND_TYPE field in the APD head

Attribute	ValuePtr contents
	er.
SQL_ATTR_PARAM_OPERATION_PTR (ODBC 3.0)	• SQLUSMALLINT*: It is the value which points to an array of SQLUSMALLINT values used to ignore the parameter while executing the SQL statement. Each value is SQL_PARAM_PROCEED (to execute the parameter) or SQL_PARAM_IGNORE (to ignore the parameter). The parameter set can be ignored during the process if it sets the status value of the array pointed by SQL_DESC_ARRAY_STATUS_PTR in APD. The parameter set is processed only if the status value is SQL_PARAM_PROCEED or the the array element is not set. The statement attribute can be set to a NULL pointer. In this case, the driver does not return the parameter status value. The attribute can be set at any time, but a new value is not used until the next SQLExecDirect or SQLExecute is called. If the bound parameter does not exist, the attribute is ignored. Setting this statement attribute sets the SQL_DESC_ARRAY_STATUS_PTR field in the APD header.
SQL_ATTR_PARAM_STATUS_PTR (ODBC 3.0)	• SQLUSMALLINT*: It is the value which points to SQLUSMALLINT array value including the status information value of each of the rows of the parameter value after calling SQLExecute or SQLExecDirect. The field is required only when PARAMSET_SIZE is bigger than 1. The status value may include the following values. ◦ SQL_PARAM_SUCCESS: The SQL statement is successfully executed for the parameter set. ◦ SQL_PARAM_SUCCESS_WITH_INFO: The SQL statement is successfully executed for the parameter set, but warning information is in the diagnostic data structure. ◦ SQL_PARAM_ERROR: An error occurs when processing the parameter set. Additional error information is in the diagnostic data structure. ◦ SQL_PARAM_UNUSED: The parameter set is not used because some previous parameter set caused an error which aborted further processing, or because SQL_PARAM_IGNORE is set for the parameter set in the array specified by the SQL_ATTR_PARAM_OPERATION_PTR. ◦ SQL_PARAM_DIAG_UNAVAILABLE: The driver treats the parameter arrays in a monolithic unit because it does not generate the error information level. The statement attribute can be set to a NULL pointer, and the driver does not return the status value of the parameter. The attribute can be set at any time. But a new value is not used until the next SQLExecDirect or SQLExecute is called. Setting the attribute can affect the operation that outputs the parameter to the driver. Setting this statement attribute sets the SQL_DESC_ARRAY_STATUS_PTR field in the IPD header.

Attribute	ValuePtr contents
SQL_ATTR_PARAMS_PROCESSED PTR (ODBC 3.0)	• SQLULEN*: It is the record field which points to the buffer returning the number of parameter sets processed and it includes the error set. If it is a NULL pointer, it is not returned. Setting this statement attribute sets SQL_DESC_ROWS_PROCESSED_PTR field of IPD header. If SQLExecDirect or SQLExecute that fills the buffer specified in the attribute does not return SQL_SUCCESS or SQL_SUCCESS_WITH_INFO, the buffer content is not defined.
SQL_ATTR_PARAMSET_SIZE (ODBC 3.0)	• SQLULEN: It is the value which specifies the number of each of the parameter values. If SQL_ATTR_PARAMSET_SIZE is bigger than 1, SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, SQL_DESC_OCTET_LENGTH_PTR of APD point to the array. Each array constant is equal to the value of the field. If the bound parameter does not exist, the attribute is ignored. Setting this statement attribute sets SQL_DESC_ARRAY_SIZE field of APD header.
SQL_ATTR_QUERY_TIMEOUT (ODBC 1.0)	• SQLULEN: The value of seconds that waits before the SQL statement is executed and returned to the application. If ValuePtr is 0 (the default value), timeout does not occur. If the specified time out value exceeds the maximum value of the data source or smaller than the minimum value of the data source, SQLSetStmtAttr replaces the value and returns SQLSTATE 01S02 (Option value changed). Even if SELECT statement is timeout the application does not need to call SQLCloseCursor to reuse the statement. Setting the query timeout of the statement attribute is valid for both synchronous and asynchronous modes.
SQL_ATTR_RETRIEVE_DATA (ODBC 2.0)	It is not supported by the driver.
SQL_ATTR_ROW_ARRAY_SIZE (ODBC 3.0)	• SQLULEN: It is the value which specifies the number of rows returned by calling SQLFetch or SQLFetchScroll. It is also the number of rows of the bookmark array which is used in the bulk bookmark operation of SQLBulkOperations. The default value is 1. If the specified row set size exceeds the maximum row set size supported by the data source, the driver replaces the value and returns SQLSTATE 01S02 (Option value changed). Setting this statement attribute sets SQL_DESC_ARRAY_SIZE field of ARD header.

Attribute	ValuePtr contents
SQL_ATTR_ROW_BIND_OFFSET_PTR (ODBC 3.0)	• SQLULEN: It is the value which points the added offset value to indicate the change of the column data binding. If the field is not NULL, the driver dereferences the pointer and adds the dereference value to each field in the descriptor record (SQL_DESC_DATA_PTR, SQL_DESC_INDICATOR_PTR, SQL_DESC_OCTET_LENGTH_PTR) and uses a new pointer value when it is bound. The default value is NULL. Setting this statement attribute sets SQL_DESC_BIND_OFFSET_PTR field of ARD header.
SQL_ATTR_ROW_BIND_TYPE (ODBC 1.0)	• SQLULEN: It is the value which sets the binding direction when SQLFetch or SQLFetchScroll is called in the related statement. If SQL_BIND_BY_COLUMN value is set, the column-wise binding is selected. If the value to length of a structure or an instance of a buffer into which result columns will be bound is set, the row-wise binding is selected. If a length is specified, they should include space for all bound columns and any padding of the structure or buffer to ensure that when the address of a bound column is increased as the specified length, the result will point to the beginning of the same column in the next row. Using sizeof operator with the structure or union of ANSI C guarantees this operation. The column-wise binding is the default binding direction of SQLFetch and SQLFetchScroll. Setting this statement attribute sets SQL_DESC_BIND_TYPE field of ARD header.
SQL_ATTR_ROW_NUMBER (ODBC 2.0)	It is not supported by the driver.
SQL_ATTR_ROW_OPERATION_PTR (ODBC 3.0)	It is not supported by the driver.
SQL_ATTR_ROW_STATUS_PTR (ODBC 3.0)	• SQLUSMALLINT*: It is the value which points to the SQLUSMALLINT array including the row status values after calling SQLFetch or SQLFetchScroll. The array has the elements as many as the number of rows included in the row set. The statement attribute can be set to a NULL pointer, and the driver does not return the row status values in this case. The attribute can be set at any time, but a new value is not used until the next call of SQLBulkOperations, SQLFetch, SQLFetchScroll or SQLSetPos. Setting this statement attribute sets SQL_DESC_ARRAY_STATUS_PTR of IRD header. The attribute is mapped to rgbRowStatus array of SQLExtendedFetch in ODBC 2.x driver.

Attribute	ValuePtr contents
SQL_ATTR_ROWS_FETCHED_PTR (ODBC 3.0)	• SQLULEN*: It points to the buffer which returns the number of rows fetched after calling SQLFetch or SQLFetchScroll. It is the number of rows affected by a bulk operation performed by a call to SQLSetPos with an operation argument of SQL_REFRESH, or it is the number of rows affected by a bulk operation performed by SQLBulkOperations. The number of rows includes the error rows. Setting this statement attribute sets SQL_DESC_ROWS_PROCESSED_PTR field of the IRD header. When SQLFetch or SQLFetchScroll is called to fill the buffer pointed by the attribute, the contents of the buffer are not defined if SQL_SUCCESS or SQL_SUCCESS_WITH_INFO is not returned.
SQL_ATTR_SIMULATE_CURSOR (ODBC 2.0)	It is not supported by the driver.
SQL_ATTR_USE_BOOKMARKS (ODBC 2.0)	It is not supported by the driver.

SQLSetStmtOption

Conformance

Introduced version: ODBC 1.0
Standards compliance: It is not available.

Overview

SQLSetStmtOption function in ODBC 2.0 is replaced with SQLSetStmtAttr in ODBC 3.x.
For more information, refer to **SQLSetStmtAttr**.

SQLSpecialColumns

Conformance

Introduced version: ODBC 1.0
Standards compliance: It is open group.

Overview

SQLSpecialColumns function retrieves the following information for a column in the specified table.

- The optimal column set that uniquely identifies a row in the table
- The columns which are automatically updated when a row value is updated by the transaction

Syntax

```
SQLRETURN SQLSpecialColumns(
    SQLHSTMT      StatementHandle,
    SQLSMALLINT   IdentifierType,
    SQLCHAR *     CatalogName,
    SQLSMALLINT   NameLength1,
    SQLCHAR *     SchemaName,
    SQLSMALLINT   NameLength2,
    SQLCHAR *     TableName,
    SQLSMALLINT   NameLength3,
    SQLSMALLINT   Scope,
    SQLSMALLINT   Nullable);
```

Arguments

StatementHandle

[Input] It is the statement handle.

IdentifierType

[Input] It is the column type to return. It should be one of the following values.

• SQL_BEST_ROWID: It returns the optimal column or set of columns that, by retrieving values fro m the column(s), allows any row in the specified table to be uniquely identified. The column may b e a pseudo column for the special purpose (ROWID of ORACLE, TID of INGRES), a general column or a column of a unique index on the table.

• SQL_ROWVER: It returns the column or columns in the specified table, if any, which are automatically updated by the data source when any value in the row is updated by any transaction.(such as in SQLBase ROWID or Sybase TIMESTAMP).

CatalogName

[Input] It is the table catalog. If the driver does not support the catalog, an empty string ("") is returned and the tables do not have a catalog. The catalog name can not include a string search pattern.

If SQL_ATTR_METADATA_ID is set to SQL_TRUE, CatalogName is treated as an identifier and it is not case-sensitive. If it is set to SQL_FALSE, CatalogName is generally treated as an string argument and it is case-sensitive. For more information, refer to **Arguments of Catalog Function**.

NameLength1

[Input] It is the length of *CatalogName string.

SchemaName

[Input] It is name of the table schema. It is the string search pattern for the schema name. If the driver does not support the schema, it returns an empty string ("") and the tables do not have a schema. SchemaName can not include a string search pattern .

If SQL_ATTR_METADATA_ID is set to SQL_TRUE, SchemaName is treated as an identifier and it is not case-sensitive. If it is set to SQL_FALSE, SchemaName is an ordinary argument, it is treated literally and it is case-sensitive.

NameLength2

[Input] It is the length of *SchemaName string.

TableName

[Input] It is name of the table. The argument can not be a NULL pointer. The table name can not include a string search pattern.

If SQL_ATTR_METADATA_ID is set to SQL_TRUE, TableName is treated as an identifier and it is not case-sensitive. If it is set to SQL_FALSE, TableName is an ordinary argument, it is treated literally, and it is case-sensitive.

NameLength3

[Input] It is the length of *TableName string.

Scope

[Input] It is minimum required scope of the rowid. The returned ROWID scope may be of a bigger scope. The value should be one of the followings.

• SQL_SCOPE_CURROW: The rowid is guaranteed to be valid only while it is positioned on that row. If it is updated or deleted by another transaction, the used ROWID is not retrieved.

• SQL_SCOPE_TRANSACTION: The rowid is guaranteed to be valid while the current transaction is maintained.

• SQL_SCOPE_SESSION: The rowid is guaranteed to be valid while the session is maintained. (It across the transaction boundaries.)

Nullable

[Input] It checks that the particular column can have NULL value. It should be one of the followings.

• SQL_NO_NULLS: A particular column can not have NULL value. Some drivers do not support SQL_NO_NULLS and they will return an empty result set if SQL_NO_NULLS is specified. The application should be prepared this case and request SQL_NO_NULLS only if it is required.

• SQL_NULLABLE: A particular column can have NULL value.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01000	General warning	It is the driver-specific informational message. (The function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure	Before the function processing is completed, the connection between the driver and the data source is failed.
24000	Invalid cursor state	A cursor is open on the StatementHandle, and SQLFetch or SQLFetchScroll had been called. This error is returned by the Driver Manager if SQLFetch or SQLFetchScroll does not return SQL_NO_DATA, and is returned by the driver if SQLFetch or SQLFetchScroll returns SQL_NO_DATA. A cursor is open on the StatementHandle, but SQLFetch or SQLFetchScroll is not called.
40001	Serialization failure	The transaction is rolled back due to a resource deadlock of other transactions.
40003	Statement completion unknown	The related connection is failed while executing this function, and the state of the transaction cannot be determined.
HY000	General error	It is an error without specific SQLSTATE, and the error message returned by SQLGetDiagRec in *MessageText buffer describes error message and its cause.
HY001	Memory allocation error	The driver can not allocate the required memory for execution or completion of the function.
HY008	Operation canceled	Asynchronous processing for StatementHandle is available and SQLCancel or SQLCancelHandle is called on StatementHandle before the function is called and completed. Then this function is called again on StatementHandle. SQLCancel or SQLCancelHandle is called on StatementHandle from oth

SQLSTATE	Error	Description
		er thread in the multithreaded application before the function is called and completed.
HY009	Invalid use of null poin ter	The TableName argument is a null pointer. The SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, the CatalogName argument is a null pointer, and the SQL_CATALOG_ NAME InfoType returns that catalog names are supported. The SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, and the SchemaName argument is a null pointer.
HY010	Function sequence err or	The asynchronously executing function is called for the connection han dle related to StatementHandle, and this function is still being execute d when SQLSpecialColumns is called. SQLExecute, SQLExecDirect, SQLMoreResults is called for StatementHa ndle, and SQL_PARAM_DATA_AVAILABLE is returned. The function is called, before the data is retrieved for all streamed parameters. The asynchronously executing function is called for StatementHandle, a nd is still being executed when this function is called. SQLExecute, SQLExecDirect, SQLBulkOperation or SQLSetPos are called for StatementHandle, and SQL_NEED_DATA is returned. This function i s called before data is sent for all data-at-execution parameters or colu mns.
HY013	Memory management error	The function is not executed because the underlying memory object ca n not be accessed because of the low memory condition.
HY090	Invalid string or buffer length	The value of a length arguments is smaller than 0 but it is not SQL_NTS. The value of a length arguments exceeds the maximum length value f or the corresponding name.
HY097	Column type out of ra nge	An invalid IdentifierType value is specified.
HY098	Scope type out of ran ge	An invalid scope value is specified.
HY099	Nullable type out of ra nge	An invalid nullable value is specified.
HY117	Connection is suspend ed due to unknown tr ansaction state. Only disconnect and read-o nly functions are allow ed.	For more information about the suspended state, refer to **SQLEndTran**.

SQLSTATE	Error	Description
HYC00	Optional feature not implemented	A catalog is specified, and the driver or data source does not support catalogs. A schema is specified, and the driver or data source does not support schemas. A string search pattern is specified, and all or one of the catalog name, table schema, table name, data source does not support it. The combination of the current settings of the SQL_ATTR_CONCURRENCY and SQL_ATTR_CURSOR_TYPE statement attributes is not supported by the driver or data source. The SQL_ATTR_USE_BOOKMARKS statement attribute is set to SQL_UB_VARIABLE, and the SQL_ATTR_CURSOR_TYPE statement attribute is set to a cursor type for which the driver does not support bookmarks.
HYT00	Timeout expired	The query timeout period is expired before the data source returns the requested result. The time limit can be set via SQLSetStmtAttr of SQL_ATTR_CONNECTION_TIMEOUT.
HYT01	Connection timeout expired	The connection timeout period is expired before the data source responds to the request. The connection time limit can be set via SQLSetConnectAttr of SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support this function	The driver does not support the function.
IM017	Polling is disabled in asynchronous notification mode	Whenever using the notification model, polling can not be used.
IM018	SQLCompleteAsync had not been called to complete the previous asynchronous on this handle.	If the previous function call for the handle returns SQL_STILL_EXECUTING and if the notification mode is activated, then SQLCompleteAsync should be called for the handle to do post processing and complete the operation.

Description

If IdentifierType argument is SQL_BEST_ROWID, SQLSpecialColumns returns a column or the column which uniquely identifies each row of the table. The column can be used in select-list or WHERE clause. SQLColumns returns various information of the table column, but it is not necessary to return the column which uniquely identifies each row nor does return a column automatically updated when any value in a row is updated by the transaction. For example, SQLColumns does not return ROWID which is the pseudo column of Oracle. It is the reason that SQLSpecialColumns is used to return information of particular column. For more information, refer to **Using Catalog Data**.

For more information about the general use, arguments, and returned data of ODBC catalog fu nctions, refer to **Catalog Function**.

If column to uniquely identify each row in the table does not exist, SQLSpecialColumns does not return a ny column. Then, SQL_NO_DATA is returned when SQLFetch or SQLFetchScroll is called in the statement.

The characteristics which are specified in IdentifierType, scope, nullable arguments are not supported by t he data source, SQLSpecialColumn returns an empty result set.

If SQL_ATTR_METADATA_ID attribute is set to SQL_TRUE, then CatalogName, SchemaName, TableName arguments are treated as identifiers and they can not use a NULL pointer. (For more information, refer to **Arguments of Catalog Function**.)

SQLSpecialColumns function returns a standard result set which is sorted by SCOPE.

The following columns are renamed in ODBC 3.x. Changing column name does not affect the backward compatibility because applications bind by column number.

ODBC 2.0 column	ODBC 3.x column
PRECISION	COLUMN_SIZE
LENGTH	BUFFER_LENGTH
SCALE	DECIMAL_DIGITS

SQLGetInfo function can be called with SQL_MAX_COLUMN_NAME_LEN option in the application to det ermine the actual length of COLUMN_NAME column.

The following table lists the columns in the result set. The additional columns beyond the column 8(PSEU DO_COLUMN) can be defined by the driver. The application should count from the end of the result set r ather than specifying an explicit position to access the columns defined by the driver. For more informatio n, refer to **Data Returning of Catalog Function**.

Column name	Column number	Data type	Description
SCOPE (ODBC 1.0)	1	Smallint	It is the actual scope of ROWID. It includes one of SQL_SCOPE_CURRO W, SQL_SCOPE_TRANSACTION, or SQL_SCOPE_SESSION. If IdentifierType is SQL_ROWVER, NULL is returned. For more information about each value, refer to scope in **Syntax**.
COLUMN_N AME (ODBC 1.0)	2	Varchar not NULL	It is the column name. If the column does not have a name, the driver r eturns an empty string.
DATA_TYPE (ODBC 1.0)	3	Smallint not NULL	It is SQL data type. It may be the ODBC SQL data type or the SQL data t ype specified in the driver. All ODBC SQL data types are valid. For more information about SQL data types specified in the driver, refer to the d ocumentation of the driver.

Column name	Column number	Data type	Description
TYPE_NAME (ODBC 1.0)	4	Varchar not NULL	It is name of the data source-dependent data type. e.g. CHAR, VARCHAR, MONEY, LONG VARBINARY, CHAR() FOR BIT DATA, etc.
COLUMN_SIZE (ODBC 1.0)	5	Integer	It is the column size in the data source.
BUFFER_LENGTH (ODBC 1.0)	6	Integer	It is the length in bytes of data transferred on an SQLGetData or SQLFetch operation if SQL_C_DEFAULT is specified. The numeric data may be different from the size of the data stored in the data source. The value is as same as COLUMN_SIZE for a string or binary data.
DECIMAL_DIGITS (ODBC 1.0)	7	Smallint	It is the number of decimal places of the column in the data source. NULL is returned if the number of decimal places can not be applied.
PSEUDO_COLUMN (ODBC 2.0)	8	Smallint	It displays whether the column is the pseudo-column such as Oracle ROWID. • SQL_PC_UNKNOWN • SQL_PC_NOT_PSEUDO • SQL_PC_PSEUDO The pseudo-column is not allowed to quote an identifier in quotation marks returned in SQLFetInfo for maximum interoperability.

After the application retrieves the SQL_BEST_ROWID value. The application can use the retrieved values to reselect the row within the defined scope. It guarantees that SELECT statement returns either *no rows* or *one row*.

If the application does not find the row when the row is re-queried based on ROWID or the column, it is assumed that the row is deleted or the ROWID column is changed. Although ROWID is not changed, the other column of the row may be updated.

The columns which are returned for the column type SQL_BEST_ROWID are very useful when the application need to move forward or backward within a result set to retrieve the most recent data among the row set. The column(s) of ROWID are not updated while it is positioned on that row.

The ROWID columns remain valid even when the cursor is not positioned on the row. The application can determine it by checking the SCOPE column in the result set.

SQLStatistics

Conformance

Introduced version: ODBC 1.0
Standards compliance: ISO 92

Overview

SQLStatistics retrieves a list of statistics about a single table and the indexes related to the table. The drive r returns the information as a result set.

Syntax

```
SQLRETURN SQLStatistics(
    SQLHSTMT        StatementHandle,
    SQLCHAR *       CatalogName,
    SQLSMALLINT     NameLength1,
    SQLCHAR *       SchemaName,
    SQLSMALLINT     NameLength2,
    SQLCHAR *       TableName,
    SQLSMALLINT     NameLength3,
    SQLUSMALLINT    Unique,
    SQLUSMALLINT    Reserved);
```

Arguments

StatementHandle

[Input] It is the statement handle.

CatalogName

[Input] It is the catalog name. If the driver does not support the catalog, an empty string ("") is retu rned and the tables do not have a catalog. The catalog name can not include a string search patter n .

If SQL_ATTR_METADATA_ID is set to SQL_TRUE, CatalogName is treated as an identifier and it is n ot case-sensitive. If it is set to SQL_FALSE, CatalogName is treated as an ordinary argument, it is tre ated literally, and it is case-sensitive. For more information, refer to **Arguments of Catalog Function**.

NameLength1

[Input] It is the length of *CatalogName string.

SchemaName

[Input] It is the schema name. If the driver does not support the schema, an empty string ("") is ret urned and the tables do not have a schema.

If SQL_ATTR_METADATA_ID is set to SQL_TRUE, SchemaName is treated as an identifier and it is n ot case-sensitive. If it is set to SQL_FALSE, SchemaNameis treated as an ordinary argument, it is tre ated literally, and it is case-sensitive.

NameLength2

[Input] It is the length of *SchemaName string.

TableName

[Input] It is the table name. The argument can not be a NULL pointer. TableName does not include a string search pattern .

If SQL_ATTR_METADATA_ID is set to SQL_TRUE, SchemaName is treated as an identifier and it is n ot case-sensitive. If it is set to SQL_FALSE, SchemaName is treated as an ordinary argument, it is tre ated literally, and it is case-sensitive.

NameLength3

[Input] It is the length of *TableName string.

Unique

[Input] It is the index type. It can be SQL_INDEX_UNIQUE or SQL_INDEX_ALL.

Reserved

[Input] The importance of CARDINALITY and PAGES columns is displayed in the result set. The foll owing options affect only the results of CARDINALITY and PAGES columns. The index information is returned even when CARDINALITY and PAGES are not returned.

• SQL_ENSURE: It requests the driver to unconditionally search for statistics. (The driver which co mplies with Open Group standards, but does not support the ODBC extension can not support SQ L_ENSURE.)

• SQL_QUICK: If it is readily available from the server, it requests the driver to unconditionally sear ch for CARDINALITY and PAGES. The driver does not guarantee that the retrieved value is the curr ent value. (The application which is registered in Open Group will always get SQL_QUICK from OD BC 3.x-compliant drivers.)

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01000	Genera warning	It is the driver-specific informational message. (The function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure	Before the function processing is completed, the connection between the driver and the data source is failed.
24000	Invalid cursor state	The cursor is open in StatementHandle, and SQLFetch or SQLFetchScroll is called. If SQLFetch or SQLFetchScroll does not return SQL_NO_DATA, the driver manager returns this error. If SQLFetch or SQLFetchScroll returns SQL_NO_DATA, the driver returns this error. The cursor is open in StatementHandle, but SQLFetch or SQLFetchScroll is not called.
40001	Serialization failure	The transaction is rolled back due to a resource deadlock of other transactions.
40003	Statement completion unknown	The related connection fails during the function execution and the status of the transaction is not able to be checked.
HY000	General error	It is an error without specific SQLSTATE, and the error message returned by SQLGetDiagRec in *MessageText buffer describes error message and its cause.
HY001	Memory allocation error	The driver can not allocate the memory required for execution or completion of the function.
HY008	Operation canceled	Asynchronous processing for StatementHandle is available and SQLCancel or SQLCancelHandle is called on StatementHandle before the function is called and completed. Then this function is called again on StatementHandle. SQLCancel or SQLCancelHandle is called on StatementHandle from other thread in the multithreaded application before the function is called and completed.
HY009	Invalid use of null pointer	TableName argument is the null pointer. The SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, the CatalogName argument is a null pointer, and the SQL_CATALOG_NAME InfoType returns that catalog names are supported. The SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, and the SchemaName argument is a null pointer.
		The asynchronously executing function is called for the connection handle related to StatementHandle, and this function is still being executed when SQLStatistics is called.

SQLSTATE	Error	Description
HY010	Function sequence error	SQLExecute, SQLExecDirect, SQLMoreResults is called for StatementHandle, and SQL_PARAM_DATA_AVAILABLE is returned. The function is called, before the data retrieves all streamed parameters. The asynchronously executing function is called for StatementHandle, and is still being executed when the function is called. SQLExecute, SQLExecDirect, SQLBulkOperation or SQLSetPos are called for StatementHandle, and SQL_NEED_DATA is returned. This function is called before data is sent for all data-at-execution parameters or columns.
HY013	Memory management error	The function is not executed because the underlying memory object can not be accessed because of the low memory condition.
HY090	Invalid string or buffer length	A name length argument value is smaller than 0, but it is not SQL_NTS. A single value of the name length arguments exceeds the maximum length.
HY100	Uniqueness option type out of range	The invalid unique value is specified.
HY101	Accuracy option type out of range	The invalid reserved value is specified.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed.	For more information about the suspended state, refer to **SQLEndTran**.
HYC00	Optional feature not implemented	A catalog is specified, and the driver or data source does not support catalogs. A schema is specified, and the driver or data source does not support schemas. A string search pattern is specified, and the catalog name, table schema, table name, data source does not support it. The combination of the current settings of the SQL_ATTR_CONCURRENCY and SQL_ATTR_CURSOR_TYPE statement attributes is not supported by the driver or data source. The SQL_ATTR_USE_BOOKMARKS statement attribute is set to SQL_UB_VARIABLE, and the SQL_ATTR_CURSOR_TYPE statement attribute is

SQLSTATE	Error	Description
		set to a cursor type for which the driver does not support bookmarks.
HYT00	Timeout expired	The query timeout period is expired before the data source returns the requested result. The time limit can be set via SQLSetStmtAttr of SQL_ATTR_CONNECTION_TIMEOUT.
HYT01	Connection timeout expired	The connection timeout period is expired before the data source responds to the request. The connection time limit can be set via SQLSetConnectAttr of SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support this function	The driver does not support the function.
IM017	Polling is disabled in asynchronous notification mode	Whenever using the notification model, polling can not be used.
IM018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	If the previous function call for the handle returns SQL_STILL_EXECUTING and if the notification mode is activated, then SQLCompleteAsync should be called for the handle to do post processing and complete the operation.

Description

SQLStatistics returns information of a table as a standard result set sorted by NON_UNIQUE, TYPE, INDEX_QUALIFIER, INDEX_NAME, ORDINAL_POSITION. The result set combines the table statistics information (CARDINALITY and PAGES row of the result set) along with information for each index. For more information, refer to **Using Catalog Data**.

SQLGetInfo is called with SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_SCHEMA_NAME_LEN, SQL_MAX_TABLE_NAME_LEN, SQL_MAX_COLUMN_NAME_LEN to determine the actual length of TABLE_CAT, TABLE_SCHEM, TABLE_NAME, COLUMN_NAME rows in the application.

For more information about the general use, arguments, and returned data of ODBC catalog functions, refer to **Catalog Function**.

The following columns are renamed in ODBC 3.x. Changing column name does not affect the backward compatibility because applications bind by column number.

ODBC 2.0 column	ODBC 3.x column
TABLE_QUALIFIER	TABLE_CAT
TABLE_OWNER	TABLE_SCHEM
SEQ_IN_INDEX	ORDINAL_POSITION
COLLATION	ASC_OR_DESC

The following table lists the columns in the result set. The additional columns beyond the column 13 (FILTER_CONDITION) can be defined by the driver. The application should count from the end of the result set rather than specifying an explicit position to access the columns defined by the driver. For more information, refer to **Data Returning of Catalog Function**.

Column name	Column number	Data type	Description
TABLE_CAT (ODBC 1.0)	1	Varchar	It is the catalog name. If the data source can not be converted, it is NULL. If the driver does not support the catalog, an empty string ("") is returned and the tables do not have the catalog.
TABLE_SCHEM (ODBC 1.0)	2	Varchar	It is the schema name. If it is not applicable to the data source, it is NULL. If the driver does not support the schema, an empty string ("") is returned and the tables do not have the schema.
TABLE_NAME (ODBC 1.0)	3	Varchar not NULL	It is the table name to which the statistics or index is applied.
NON_UNIQUE (ODBC 1.0)	4	Smallint	It indicates whether the duplication of the index value is allowed. • SQL_TRUE: The index values may not be unique. • SQL_FALSE: The index values should be unique. If TYPE is SQL_TABLE_STAT, NULL is returned.
INDEX_QUALIFIER (ODBC 1.0)	5	Varchar	It is an identifier which executes DROP INDEX and specifies the index name. If it is the data source which does not support the index rules or TYPE is SQL_TABLE_STAT, then NULL is returned. If a non-NULL value is returned in the row, the value is used to define the index identified in DROP INDEX. Otherwise, the TABLE_SCHEM should be used to qualify the index name.
INDEX_NAME (ODBC 1.0)	6	Varchar	It is the index name. If TYPE is SQL_TABLE_STAT, NULL is returned.
TYPE (ODBC 1.0)	7	Smallint not NULL	It is the information type to be returned. • SQL_TABLE_STAT: It indicates statistics for the table. (in CARDINALITY or PAGES row) • SQL_INDEX_BTREE: It indicates B-Tree index. • SQL_INDEX_CLUSTERED: It indicates the cluster index. • SQL_INDEX_CONTENT: It indicates the content of the index. • SQL_INDEX_HASHED: It indicates the hash index. • SQL_INDEX_OTHER: It indicates other types of index.
ORDINAL_POSITION (ODBC 1.0)	8	Smallint	It is the row order on the index (starting from 1). If TYPE is SQL_TABLE_STAT, NULL is returned.
COLUMN_N			It is the column name. If the row is an expression such as SALARY + BE

Column name	Column number	Data type	Description
AME (ODBC 1.0)	9	Varchar	NEFITS, the expression is returned. If the expression is unknown, an empty string is returned. If TYPE is SQL_TABLE_STAT, NULL is returned.
ASC_OR_DESC (ODBC 1.0)	10	Char(1)	It is the column sorting sequence. A is ascending, and D is descending. If the row sorting sequence is not supported in the data source or TYPE is SQL_TABLE_STAT, then NULL is returned.
CARDINALITY (ODBC 1.0)	11	Integer	The cardinality of the table or index. If TYPE is SQL_TABLE_STAT, it is the number of rows of the table. If TYPE is not SQL_TABLE_STAT, it is the number of unique values of the index. If the value can not be used from the data source, NULL is returned.
PAGES (ODBC 1.0)	12	Integer	It is the number of pages stored in the index or table. If TYPE is SQL_TABLE_STAT, it is the number of pages of the table. If TYPE is not SQL_TABLE_STAT, it is the number of pages of the index. If the value of the data source can not be used or converted, then NULL is returned.
FILTER_CONDITION (ODBC 2.0)	13	Varchar	If the index is the filtered index such as SALARY> 30000, it is the filter condition. If the filter condition can not be determined, it is an empty string. If it is not an index, the filtered index can not be determined or TYPE is SQL_TABLE_STAT.

If the row in the result set corresponds to the table, the driver sets TYPE to SQL_TABLE_STAT, and sets NON_UNIQUE, INDEX_QUALIFIER, INDEX_NAME, ORDINAL_POSITION, COLUMN and ASC_OR_DESC to NULL. If CARDINALITY or PAGES can not be used in the data source, the driver sets them to NULL.

SQLTablePrivileges

Conformance

Introduced version: ODBC 1.0
Standards compliance: ODBC

Overview

SQLTablePrivileges returns the table list and the privileges related to each table. The driver returns the inf ormation in the form of the result set in the specified statement.

Syntax

```
SQLRETURN SQLTablePrivileges(
    SQLHSTMT      StatementHandle,
    SQLCHAR *     CatalogName,
    SQLSMALLINT   NameLength1,
    SQLCHAR *     SchemaName,
    SQLSMALLINT   NameLength2,
    SQLCHAR *     TableName,
    SQLSMALLINT   NameLength3);
```

Arguments

StatementHandle

[Input] It is the statement handle for the search results.

CatalogName

[Input] It is the table catalog. If the driver does not support the catalog, an empty string ("") is retur ned and the tables do not have a catalog. The catalog name can not include a string search patter n .

If SQL_ATTR_METADATA_ID is set to SQL_TRUE, CatalogName is treated as an identifier and it is n ot case-sensitive. If it is set to SQL_FALSE, CatalogNameis treated as an ordinary argument, it is tre ated literally, and it is case-sensitive. For more information, refer to **Arguments of Catalog Function**.

NameLength1

[Input] It is the string length of *CatalogName.

SchemaName

[Input] It is a string search pattern for the schema name. If the driver does not support the schema, an empty string ("") is returned and the tables do not have a schema.

If SQL_ATTR_METADATA_ID is set to SQL_TRUE, SchemaName is treated as an identifier and it is not case-sensitive. If it is set to SQL_FALSE, SchemaName is treated as a pattern value string argument and it is case-sensitive.

NameLength2

[Input] It is the string length of *SchemaName.

TableName

[Input] It is table name string search pattern .

If SQL_ATTR_METADATA_ID is set to SQL_TRUE, TableName is treated as an identifier and it is not case-sensitive. If it is set to SQL_FALSE, TableName is treated as a pattern value string argument and it is case-sensitive.

NameLength3

[Input] It is the string length of *TableName.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01000	General warning	It is the driver-specific informational message. (The function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure	Before the function processing is completed, the connection between the driver and the data source is failed.
24000	Invalid curosr state	A cursor is open on the StatementHandle, and SQLFetch or SQLFetchScroll had been called. This error is returned by the driver manager if SQLFetch or SQLFetchScroll does not return SQL_NO_DATA, and is returned by the driver if SQLFetch or SQLFetchScroll returns SQL_NO_DATA. A cursor is open on the StatementHandle, but SQLFetch or SQLFetchScroll is not called.
40001	Serialization failure	The transaction is rolled back due to a resource deadlock of other transactions.
40003	Statement completion unknown	The related connection fails during the function execution and the status of the transaction is not able to be checked.

SQLSTATE	Error	Description
HY000	General error	It is an error without specific SQLSTATE, and the error message returned by SQLGetDiagRec in *MessageText buffer describes error message and its cause.
HY001	Memory allocation error	The driver can not allocate the memory required for execution or completion of the function.
HY008	Operation canceled	Asynchronous processing for StatementHandle is available and SQLCancel or SQLCancelHandle is called on StatementHandle before the function is called and completed. Then this function is called again on StatementHandle. SQLCancel or SQLCancelHandle is called on StatementHandle from other thread in the multithreaded application before the function is called and completed.
HY009	Invalid use of null pointer	SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, and CatalogName argument is a null pointer. SQL_CATALOG_NAME InfoType returns that catalog names are supported. SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, and SchemaName or TableName argument is a null pointer.
HY010	Function sequence error	The asynchronously executing function is called for the connection handle related to StatementHandle, and this function is still being executed when SQLTablePrevileges is called. SQLExecute, SQLExecDirect or SQLMoreResults are called for StatementHandle, and SQL_PARAM_DATA_AVAILABLE is returned. This function is called before data is retrieved for all streamed parameters. The asynchronously executing function is called for StatementHandle, and is still being executed when the function is called. SQLExecute, SQLExecDirect, SQLBulkOperation or SQLSetPos are called for StatementHandle, and SQL_NEED_DATA is returned. This function is called before data is sent for all data-at-execution parameters or columns.
HY013	Memory management error	The function is not executed because the underlying memory object can not be accessed because of the low memory condition.
HY090	Invalid string or buffer length	The value of a length arguments is smaller than 0 but it is not SQL_NTS. The value of a length arguments exceeds the maximum length value for the corresponding name.
HY117	Connection is suspended due to unknown transaction state. Only	For more information about the suspended state, refer to **SQLEndTran**.

SQLSTATE	Error	Description
	disconnect and read-only functions are allowed.	
HYC00	Optional feature not implemented	A catalog is specified, and the driver or data source does not support catalogs. A schema is specified, and the driver or data source does not support schemas. A string search pattern is specified, and the catalog name, table schema, table name, data source does not support it. The combination of the current settings of the SQL_ATTR_CONCURRENCY and SQL_ATTR_CURSOR_TYPE statement attributes is not supported by the driver or data source. The SQL_ATTR_USE_BOOKMARKS statement attribute is set to SQL_UB_VARIABLE, and the SQL_ATTR_CURSOR_TYPE statement attribute is set to a cursor type for which the driver does not support bookmarks.
HYT00	Timeout expired	The query timeout period is expired before the data source returns the requested result. The time limit can be set via SQLSetStmtAttr of SQL_ATTR_CONNECTION_TIMEOUT.
HYT01	Connection timeout expired	The connection timeout period is expired before the data source responds to the request. The connection time limit can be set via SQLSetConnectAttr of SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support this function	The driver does not support the function.
IM017	Polling is disabled in asynchronous notification mode	Whenever using the notification model, polling can not be used.
IM018	SQLCompleteAsync has not been called to complete the previous asynchronous operation on this handle.	If the previous function call for the handle returns SQL_STILL_EXECUTING and if the notification mode is activated, then SQLCompleteAsync should be called for the handle to do post processing and complete the operation.

Description

SchemaName and TableName arguments accept search patterns. For more information about valid search patterns, refer to **Pattern Value Argument**.

SQLTablePrivileges returns the result as a standard result set sorted by TABLE_CAT, TABLE_SCHEM, TABLE_NAME, PRIVILEGE, GRANTEE.

SQLGetInfo function is called with SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_SCHEMA_NAME_LEN, SQL_MAX_TABLE_NANE_LEN to determine the actual length of TABLE_CAT, TABLE_SCHEM, TABLE_NAME columns in the application.

For more information about the general use, arguments, and returned data of ODBC catalog functions, refer to **Catalog Function**.

The following columns are renamed in ODBC 3.x. Changing column name does not affect the backward compatibility because applications bind by column number.

ODBC 2.0 column	ODBC 3.x column
TABLE_QUALIFIER	TABLE_CAT
TABLE_OWNER	TABLE_SCHEM

The following table lists the columns in the result set. The additional columns beyond the column 7 (IS_GRANTABLE) can be defined by the driver. The application should count from the end of the result set rather than specifying an explicit position to access the columns defined by the driver. For more information, refer to **Data Returning of Catalog Function**.

Column name	Column number	Data type	Description
TABLE_CAT (ODBC 1.0)	1	Varchar	It is the catalog name. If the data source can not be converted, it is NULL. If the driver does not support the catalog, an empty string ("") is returned and the tables do not have the catalog.
TABLE_SCHE (ODBC 1.0)	2	Varchar	It is the schema name. If it is not applicable to the data source, it is NULL. If the driver does not support the schema, an empty string ("") is returned and the tables do not have the schema.
TABLE_NAME (ODBC 1.0)	3	Varchar not NULL	It is the table name.
GRANTOR (ODBC 1.0)	4	Varchar	It is the user name who grants the privilege. If it is not applicable to the data source, it is NULL. For all rows in which the value in the GRANTEE column is the owner of the object, the GRANTOR column is _SYSTEM.
GRANTEE (ODBC 1.0)	5	Varchar not NULL	It is the user name to whom the privilege is granted.
			The table privilege. It is one of the followings or the a data source-specific privilege. • SELECT: Grantee is allowed to retrieve one or more columns in a table.

Column name	Column number	Data type	Description
PRIVILEGE (ODBC 1.0)	6	Varchar not NULL	• INSERT: Grantee is allowed to insert new rows containing data for one or more columns into a table. • UPDATE: Grantee is allowed to update one or more columns in a table. • DELETE: Grantee is allowed to delete the data of a table. • REFERENCES: Grantee is allowed to refer to one or more columns in a table within constraints. (e.g. unique, referential, Table constraints check) The scope of action allowed to grantee who is given the table privilege depends on the data source. For example, the UPDATE privilege permits the grantee to update all columns in a table on one data source and only the columns for which the grantor has the UPDATE privilege on another data source.
IS_GRANTABLE (ODBC 1.0)	7	Varchar	It indicates whether GRANTEE can give other user the privilege and it is YES or NO. If it is not applicable to the data source or unknown, it is NULL. The privilege is grantable or not grantable, and it can not be both of them. The result set that SQLColumnPrivileges returns does not include the two rows which all columns except for IS_GRANTABLE have the same value.

SQLTables

Conformance

Introduced version: ODBC 1.0
Standards compliance: Open group

Overview

SQLTables returns the table list, catalog, or schema name, table type stored in the specified data source. The driver returns the information as a result set.

Syntax

```
SQLRETURN SQLTables(
    SQLHSTMT       StatementHandle,
    SQLCHAR *      CatalogName,
    SQLSMALLINT    NameLength1,
    SQLCHAR *      SchemaName,
    SQLSMALLINT    NameLength2,
    SQLCHAR *      TableName,
    SQLSMALLINT    NameLength3,
    SQLCHAR *      TableType,
    SQLSMALLINT    NameLength4);
```

Arguments

StatementHandle

[Input] It is the statement handle for the search results.

CatalogName

[Input] It is the catalog name. If SQL_ODBC_VERSION environment attribute is SQL_OV_ODBC3, CatalogName argument accepts a search pattern. If the driver supports catalogs only for a few tables, for example, the driver retrieves the data from another DBMS, an empty string ("") indicates that the table does not have a catalog.

If SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, CatalogName is treated as an identifier and it is not case-sensitive. If it is set to SQL_FALSE, CatalogName is a pattern value argument, it is treated literally and it is case-sensitive. For more information, refer to **Arguments of Catalog Function** .

NameLength1

[Input] It is the string length of *CatalogName.

SchemaName

[Input] It is the string search pattern for the schema name. An empty string ("") such as CatalogName indicates that the table does not have schema. The argument processing depends on SQL_ATTR_METADATA_ID statement attribute such as CatalogName.

NameLength2

[Input] It is the character length of *SchemaName.

TableName

[Input] It is the string search pattern for the table name. Also, the argument processing depends on SQL_ATTR_METADATA_ID statement attribute such as CatalogName.

NameLength3

[Input] It is the character length of *TableName.

TableType

[Input] It is the matched table type list.

SQL_ATTR_METADATA_ID statement attribute does not affect TableType argument. TableType is the value list argument, regardless of setting of SQL_ATTR_METADATA_ID.

NameLength4

[Input] It is the character length of *TableType.

Returns

SQL_SUCCESS, SQL_SUCCESS_WITH_INFO, SQL_STILL_EXECUTING, SQL_ERROR, SQL_INVALID_HANDLE

Diagnosis

SQLSTATE	Error	Description
01000	General waring	It is the driver-specific informational message. (The function returns SQL_SUCCESS_WITH_INFO.)
08S01	Communication link failure	Before the function processing is completed, the connection between the driver and the data source is failed.
24000	Invalid cursor state	A cursor is open on the StatementHandle, and SQLFetch or SQLFetchScroll had been called. This error is returned by the Driver Manager if SQLFetch or SQLFetchScroll does not return SQL_NO_DATA, and is returned by the driver if SQLFetch or SQLFetchScroll returns SQL_NO_DATA. A cursor is open on the StatementHandle, but SQLFetch or SQLFetchSc

SQLSTATE	Error	Description
		roll is not called.
40001	Serialization failure	The transaction is rolled back due to a resource deadlock of other trans actions.
40003	Statement completion unknown	The related connection is failed while executing this function, and the s tate of the transaction cannot be determined.
HY000	General error	It is an error without specific SQLSTATE, and the error message returne d by SQLGetDiagRec in *MessageText buffer describes error message a nd its cause.
HY001	Memory allocation err or	The driver can not allocate the required memory for execution or compl etion of the function.
HY008	Operation canceled	Asynchronous processing for StatementHandle is available and SQLCan cel or SQLCancelHandle is called on StatementHandle before the functi on is called and completed. Then this function is called again on State mentHandle. SQLCancel or SQLCancelHandle is called on StatementHandle from oth er thread in the multithreaded application before the function is called and completed.
HY009	Invalid use of null poi nter	SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, and CatalogName argument is a null pointer. SQL_CATALOG_NAME InfoTy pe returns that catalog names are supported. SQL_ATTR_METADATA_ID statement attribute is set to SQL_TRUE, and SchemaName or TableName argument is a null pointer.
HY010	Function sequence err or	The asynchronously executing function is called for the connection han dle related to StatementHandle, and this function is still being executed when SQLTables is called. SQLExecute, SQLExecDirect, SQLMoreResults is called for StatementHa ndle, and SQL_PARAM_DATA_AVAILABLE is returned. The function is called, before the data for all streamed parameters is retrieved. The asynchronously executing function is called for StatementHandle, a nd is still being executed when SQLTables function is called. SQLExecute, SQLExecDirect, SQLBulkOperation or SQLSetPos are called for StatementHandle, and SQL_NEED_DATA is returned. This function i s called before data is sent for all data-at-execution parameters or colu mns.
HY013	Memory management error	The function is not executed because the underlying memory object ca n not be accessed because of the low memory condition.
HY090	Invalid string or buffer length	The value of a length arguments is smaller than 0 but it is not SQL_NTS.

SQLSTATE	Error	Description
		The value of a length arguments exceeds the maximum length value for the corresponding name.
HY117	Connection is suspended due to unknown transaction state. Only disconnect and read-only functions are allowed	For more information about the suspended state, refer to **SQLEndTran**.
HYC00	Optional feature not implemented	A catalog is specified, and the driver or data source does not support catalogs. A schema is specified, and the driver or data source does not support schemas. A string search pattern is specified, and the catalog name, table schema, table name, data source does not support it. The combination of the current settings of the SQL_ATTR_CONCURRENCY and SQL_ATTR_CURSOR_TYPE statement attributes is not supported by the driver or data source. The SQL_ATTR_USE_BOOKMARKS statement attribute is set to SQL_UB_VARIABLE, and the SQL_ATTR_CURSOR_TYPE statement attribute is set to a cursor type for which the driver does not support bookmarks.
HYT00	Timeout expired	The query timeout period is expired before the data source returns the requested result. The time limit can be set via SQLSetStmtAttr of SQL_ATTR_CONNECTION_TIMEOUT.
HYT01	Connection timeout expired	The connection timeout period is expired before the data source responds to the request. The connection time limit can be set via SQLSetConnectAttr of SQL_ATTR_CONNECTION_TIMEOUT.
IM001	Driver does not support this function	The driver does not support the function.
IM017	Polling is disabled in asynchronous notification mode	Whenever using the notification model, polling can not be used.
IM018	SQLCompleteASync has not been called to complete the previous asynchronous operation on this handle.	If the previous function call for the handle returns SQL_STILL_EXECUTING and if the notification mode is activated, then SQLCompleteAsync should be called for the handle to do post processing and complete the operation.

Description

SQLTables lists all tables in the requested scope. The user may have the SELECT privilege on the tables, or not. The application checks the accessibility as follows.

- SQL_ACCESSIBLE_TABLES information type is checked by calling SQLGetInfo.
- The privilege on each table is checked by calling SQLTablePrivileges.

Otherwise, the application should deal with the situation that the user performs a select statement without SELECT privilege.

SchemaName and TableName arguments can use the search pattern. If SQL_ODBC_VERSION is SQL_OV_ODBC3, then CatalogName argument can also use the search pattern. For more information about valid search pattern, refer to **Pattern Value Argument**.

For more information about the general use, arguments, and returned data of ODBC catalog functions, refer to **Catalog Function**.

The particular meaning of CatalogName, SchemaName, TableName, TableType arguments of SQLTables is defined as follows to support the enumeration of catalog, schema, and table types.

- If CatalogName is SQL_ALL_CATALOGS and SchaemaName and TableName are the empty string, the result set includes a valid catalog for the data source. (All columns except for TABLE_CAT column include NULL.)
- If SchemaName is SQL_ALL_SCHEMAS and CatalogName and TableName are the empty string, the result set includes a valid schema for the data source. (All columns except for TABLE_SCHE column include NULL.)
- If TableType is SQL_ALL_TABLE_TYPES and CatalogName, SchemaName, and TableName are the empty string, the result set includes a valid table type for the data source. (All columns except for TABLE_TYPE column include NULL.)

If TableType is not an empty string, it should include a list of comma-separated values for the types of interest. Each value can be enclosed in single quote (') or unquoted such as 'TABLE', 'VIEW' or TABLE, VIEW. An application should always specify the table type in uppercase. The driver should convert the table type to whatever case is needed by the data source. If the data source does not support the specified table type, SQLTables does not return any data result for that type.

SQLTables returns the result as a standard result set ordered by TABLE_TYPE, TABLE_CAT, TABLE_SCHE, TABLE_NAME. For more information, refer to **Using Catalog Data**.

SQLGetInfo is called with SQL_MAX_CATALOG_NAME_LEN, SQL_MAX_SCHEMA_NAME_LEN, SQL_MAX_TABLE_NAME_LEN to determine the actual length of TABLE_CAT, TABLE_SCHEM, TABLE_NAME column in the application.

The following columns are renamed in ODBC 3.x. Changing column name does not affect the backward compatibility because applications bind by column number.

ODBC 2.0 column	ODBC 3.x column
TABLE_QUALIFIER	TABLE_CAT
TABLE_OWNER	TABLE_SCHEM

The following table lists the columns in the result set. The additional columns beyond the column 5 (REMARKS) can be defined by the driver. The application should count from the end of the result set rather than specifying an explicit position to access the columns defined by the driver.
For more information, refer to **Data Returning of Catalog Function**.

Column name	Column number	Data type	Description
TABLE_CAT (ODBC 1.0)	1	Varchar	It is the catalog name. If it is not applicable to the data source, it is NULL. If the driver does not support the catalog, an empty string ("") is returned and the tables do not have the catalog.
TABLE_SCHEM (ODBC 1.0)	2	Varchar	It is the schema name. If it is not applicable to the data source, it is NULL. If the driver does not support the schema, an empty string ("") is returned and the tables do not have the schema.
TABLE_NAME (ODBC 1.0)	3	Varchar	It is the table name.
TABLE_TYPE (ODBC 1.0)	4	Varchar	It is the table type name. It is one of "TABLE", "VIEW", "SYSTEM TABLE", "GLOBAL TEMPORARY", "LOCAL TEMPORARY", "ALIAS", "SYNONYM" or the name specified in the data source. The meaning of "ALIAS" and "SYNONYM" is driver-specific.
REMARKS (ODBC 1.0)	5	Varchar	It is the description for the table.

SQLTransact

Conformance

Introduced version: ODBC 1.0
Standards compliance: It is not available.

Overview

SQLTransact function in ODBC 2.x is replaced with SQLEndTran in ODBC 3.x.
For more information, refer to **SQLEndTran**.

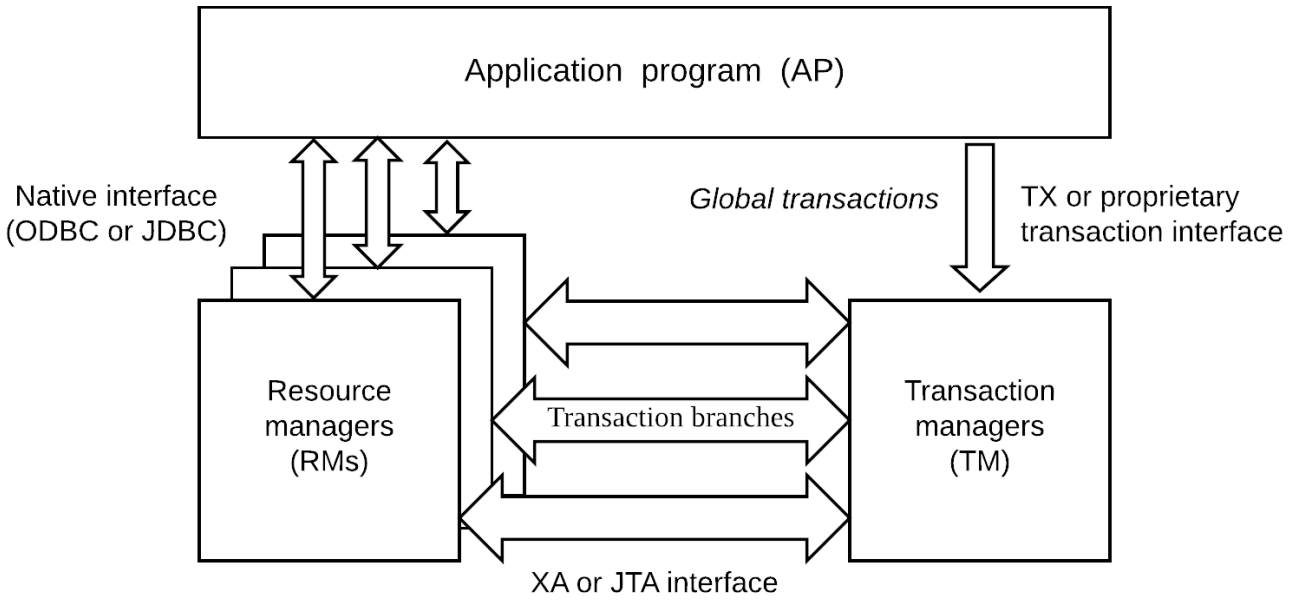
31.7 XA API References

Overview

X/Open XA is the standards for distributed transaction processing defined in X/Open. Typically, it prescribes the interface between a multi-transaction manager and a local resource manager. XA describes what is needed for the resource manager to process the transaction.

GOLDILOCKS XA is implemented based on X/Open CAE document, Distributed Transaction Processing: The XA Specification (**http://www.opengroup.org/public/catalog/c193.htm**).

X/Open DTP (Distributed Transaction Processing) model defines the transaction management between different heterogeneous computer databases.



- Application Program (AP): It defines the work consisting of transactions.
- Resource Managers (RM): It manages the shared resources accessed by a distributed transaction. It means a database management system, such as GOLDILOCKS.
- Transaction Manager (TM): It assigns ID(XID) of a distributed transaction, and manages the progress of a distributed transaction, and has the responsibility for the termination and recovery of a distributed transaction.

AP is a program developed by using the precompiler or ODBC. AP does not directly use the XA interface, and it uses the native interface of RM, or it controls the transaction by using TX interface provided by TM.

XA Interface

XA interface is the interface specification between RM and TM. GOLDILOCKS does not provide a separate library for XA interface, and it is included in the ODBC libraries provided by GOLDILOCKS.

switch_t Structure

It is a structure which includes information about the entry point for XA interface and information about RM.
xa_switch_t provided by GOLDILOCKS is goldilocks_xa_switch.

Variable name	Description
char name[RMNAMESZ]	It is RM name.
long flags	It is the option provided in RM. • MIGRATE option is not supported. (It is set to TMNOMIGRATE.)
long version	It is RM version.
int (*xa_open_entry)(char *, int, long);	It is xa_open function pointer.
int (*xa_close_entry)(char *, int, long);	It is xa_close function pointer.
int (*xa_start_entry)(XID *, int, long);	It is xa_start function pointer.
int (*xa_end_entry)(XID *, int, long);	It is xa_end function pointer.
int (*xa_rollback_entry)(XID *, int, long);	It is xa_rollback function pointer.
int (*xa_prepare_entry)(XID *, int, long);	It is xa_prepare function pointer.
int (*xa_commit_entry)(XID *, int, long);	It is xa_commit function pointer.
xint (*xa_recover_entry)(XID *, long, int, long);	It is xa_recover function pointer.
int (*xa_forget_entry)(XID *, int, long);	It is xa_forget function pointer.
int (*xa_complete_entry)(int *, int *, int, long);	It is xa_complete function pointer. • It is not provided in GOLDILOCKS.

ODBC Functions Related to XA

It describes the functions which are added to use the XA interface besides the standard ODBC.

SQLGetXaSwitch

It gets xa_switch_t provided by ODBC function.

```
xa_switch_t * SQLGetXaSwitch( void );
```

Return

It returns a pointer to sw_switch_t structure provided by ODBC. NULL can not be returned.

SQLGetXaConnectionHandle

It gets the connection handle related to the current XA session.

```
SQLHANDLE SQLGetXaConnectionHandle( void );
```

Return

If the connected XA session exist at the corresponding thread, it returns the related connection handle. O therwise, it returns NULL.

XA Functions

It describes the details for XA related functions of xa_switch_t structure.

xa_open

It connects to RM.

It is ignored if it is already connected in the thread which called xa_open.

```
int xa_open(
    char * xa_info,
    int    rmid,
    long   flags );
```

Arguments

xa_info

[Input] It is a string which includes the access information, and the maximum length is 256 bytes. For more information, refer to InConnectionString of **SQLDriverConnect**.

rmid

[Input] It is the unique ID of RM to be connected. The argument is ignored.

flags

[Input] It is connection flag. It should be set to TMNOFLAGS.

Diagnosis

Return value	Description
XA_OK	The operation is normally performed.
XAER_RMFAIL	It occurs when RM can not be used.
XAER_RMERR	RM can not perform the operation in a transaction branch due to lack of resources.
XAER_INVAL	It occurs when passing the abnormal argument.

xa_close

It terminates the connection of RM, and releases the connection handle.

```
int xa_close(
    char * xa_info,
    int    rmid,
    long   flags );
```

Arguments

xa_info

[Input] It is a string which includes the access information, and the argument is ignored.

rmid

[Input] It is the unique ID of RM to be connected. The argument is ignored.

flags

[Input] It is the termination flag. The argument is ignored.

Diagnosis

Return value	Description
XA_OK	The operation is normally performed.
XAER_RMFAIL	It occurs when RM can not be used.
XAER_RMERR	RM can not perform the operation in a transaction branch due to lack of resources.
XAER_INVAL	It occurs when passing the abnormal argument.

xa_start

It creates a new transaction branch or starts the existing transaction branch.

```
int xa_start(
    XID *  xid,
    int    rmid,
    long   flags );
```

Arguments

xid

[Input] It is the transaction ID to be started.

rmid

[Input] It is the unique ID of RM to be connected. The argument is ignored.

flags

[Input] It is the start flag.

flags can use the following values.

- TMASYNC: It starts a transaction branch in asynchronous mode. (It is not supported.)
- TMNOFLAGS: FLAG is not used and the appropriate flag should be specified if flag is not used.
- TMNOWAIT: If the specified transaction branch is used by another session, XA_RETRY error is returne d without waiting.
- TMRESUME: The previously suspended transaction branch is continued. It can not be used with TMJ OIN.
- TMJOIN: It is connected to an existing transaction branch. It can not be used with TMRESUME.

Diagnosis

Return value	Description
XA_OK	The operation is normally performed.
XAER_RMFAIL	It occurs when the session in use is abnormally terminated.
XAER_RMERR	RM can not perform the operation in a transaction branch due to lack of resources.
XAER_PROTO	It occurs when the execution order does not fit into XA protocol.
XAER_INVAL	It occurs when passing the abnormal argument.
XAER_DUPID	The transaction branch with the same XID already exists.
XAER_NOTA	The transaction branch specified as XID does not exist. It may occur when TMRESUME or TMJOIN is used.
XA_RETRY	It occurs when the specified transaction branch is already used by another session. It may occur when using TMNOWAIT.
XAER_OUTSIDE	The local transaction is already in progress in the session.

Return value	Description
XA_RBROLLBACK	The transaction branch is set to rollback-only.

xa_end

It terminates the transaction branch operation.

```
int xa_end(
    XID *  xid,
    int    rmid,
    long   flags );
```

Arguments

xid

[Input] It is the transaction ID to be terminated.

rmid

[Input] It is the unique ID of RM to be connected. The argument is ignored.

flags

[Input] It is the job termination flag.

flags can use the following values.

- TMFAIL: It indicates that the operation failed. It can not be used with TMSUSPEND or TMSUCCESS, a nd it changes the status of the transaction branch to rollback-only.
- TMMIGRATE: It is restarted in connection with another branch. (It is not supported.)
- TMSUCCESS: It indicates that the operation is successfully terminated. It can not be used with TPSUS PEND or TMFAIL.
- TMSUSPEND: The transaction branch is suspended and terminated.

Diagnosis

Return value	Description
XA_OK	The operation is normally performed.
XAER_RMFAIL	It occurs when the session in use is abnormally terminated.
XAER_RMERR	RM can not perform the operation in a transaction branch due to lack of resources.
XA_NOMIGRATE	MIGRATE flag is not supported.
XAER_PROTO	It occurs when the execution order does not fit into XA protocol.
XAER_INVAL	It occurs when passing the abnormal argument.
XAER_NOTA	The transaction branch specified as XID does not exist.

xa_prepare

It prepares to commit the transactions corresponding to XID. It is the first phase of Two-phase Commit Pr otocol (2PC).

```
int xa_prepare(
    XID *  xid,
    int    rmid,
    long   flags );
```

Arguments

xid

[Input] It is the transaction ID to be prepared.

rmid

[Input] It is the unique ID of RM to be connected. The argument is ignored.

flags

[Input] It is the prepare flag. TMNOFLAGS should be set.

Diagnosis

Return value	Description
XA_OK	The operation is normally performed.
XAER_RMFAIL	It occurs when the session in use is abnormally terminated.
XAER_RMERR	RM can not perform the operation in a transaction branch due to lack of resources.
XA_PROTO	It occurs when the execution order does not fit into XA protocol.
XA_RDONLY	The transaction branch is the read-only transaction.
XAER_NOTA	The transaction branch specified as XID does not exist.
XA_RBROLLBACK	The transaction branch is set to rollback-only.

xa_commit

It commits the transactions corresponding to XID. It is the second phase of Two-phase Commit Protocol (2PC).

```
int xa_commit(
    XID *  xid,
    int    rmid,
    long   flags );
```

Arguments

xid

[Input] It is the transaction ID to be committed.

rmid

[Input] It is the unique ID of RM to be connected. The argument is ignored.

flags

[Input] It is the commit flag.

flags can use the following values.

- TMNOFLAGS: It means that FLAG is not used, and if any flag is not used, the appropriate flag should be specified.
- TMONEPHASE: It performs One Phase Commit (1PC).

Diagnosis

Return value	Description
XA_OK	The operation is normally performed.
XAER_RMFAIL	It occurs when the session in use is abnormally terminated.
XAER_RMERR	RM can not perform the operation in a transaction branch due to lack of resources.
XA_PROTO	It occurs when the execution order does not fit into XA protocol.
XA_RDONLY	The transaction branch is the read-only transaction.
XAER_NOTA	The transaction branch specified as XID does not exist.
XA_RBROLLBACK	The transaction branch is set to rollback-only.

xa_rollback

It rolls back the transaction corresponding to XID.

```
int xa_rollback(
    XID *  xid,
    int    rmid,
    long   flags );
```

Arguments

xid

[Input] It is the transaction ID to be rolled back.

rmid

[Input] It is the unique ID of RM to be connected. The argument is ignored.

flags

[Input] It is rollback flag. TMNOFLAGS should be set.

Diagnosis

Return value	Description
XA_OK	The operation is normally performed.
XAER_RMFAIL	It occurs when the session in use is abnormally terminated.
XAER_RMERR	RM can not perform the operation in a transaction branch due to lack of resources.
XA_HEURRB	The transaction branch is already rolled back heuristically. (heuristic rollback)
XA_HEURCOM	The transaction branch is already committed heuristically. (heuristic commit)
XAER_NOTA	The transaction branch specified as XID does not exist.

xa_recover

It gets a list of transactions which are heuristically committed or rolled back.

```
int xa_recover(
    XID *  xids,
    long   count,
    int    rmid,
    long   flags );
```

Arguments

xids

[Output] It is the list of transactions which are heuristically committed or rolled back.

count

[Input] It refers to the array size of xids.

rmid

[Input] It is the unique ID of RM to be connected. The argument is ignored.

flags

[Input] It is the recover flag.

flags can use the following values.

- TMSTARTSCAN: It starts to scan, or it starts again from the beginning.
- TMENDSCAN: It terminates the scan.
- TMNOFLAGS: If it is used after TMSTARTSCAN, the following list is obtained. XA_PROTO error occurs if TMNOFLAGS is used from the beginning.

Diagnosis

Return value	Description
>= 0	It is the number of returned XIDs. (The array size that is valid in xids.)
XAER_RMFAIL	It occurs when the session in use is abnormally terminated.
XAER_RMERR	RM can not perform the operation in a transaction branch due to lack of resources.
XA_PROTO	It occurs when the execution order does not fit into XA protocol.

xa_forget

It deletes the information about transactions which are heuristically committed or rolled back.

```
int xa_forget(
    XID *  xid,
    int    rmid,
    long   flags );
```

Arguments

xid

[Input] It is the transaction ID to be deleted.

rmid

[Input] It is the unique ID of RM to be connected. The argument is ignored.

flags

[Input] It is the delete flag. TMNOFLAGS should be set.

Diagnosis

Return value	Description
XA_OK	The operation is normally performed.
XAER_RMFAIL	It occurs when the session in use is abnormally terminated.
XAER_RMERR	RM can not perform the operation in a transaction branch due to lack of resources.
XAER_PROTO	It occurs when the execution order does not fit into XA protocol.
XAER_INVAL	It occurs when passing the abnormal argument.
XAER_NOTA	The transaction branch specified by XID does not exist.

Example

The following is a simple example of connecting to GOLDILOCKS and performing Two-phase Commit aft er inserting/ retrieving/ updating/ deleting the record. The complete code for the following example is in $GOLDILOCKS_HOME/sample/ODBC/SAMPLE_XA.c.

1. SQLGetXaSwitch()
   Obtain xa switch_t structure for using XA interface.

2. SQLAllocHandle()
   Obtain ODBC environment handle.

3. xa_open()
   Connect to the server with the given connection string.
   Use the existing connection if it is already connected to the server. Otherwise, try a new connection.

4. SQLGetXaConnectionHandle()
   Obtain XA connection handle related to the corresponding thread.

5. xa_start()
   Start the transaction branch.

6. Perform the transaction by using standard ODBC functions.

7. xa_end()
   Terminate the transaction.

8. xa_prepare()
   Prepare to commit the transaction, and which is the first phase of 2PC.

9. xa_commit()
   Commit the transaction, and which is the second phase of 2PC.

10. xa_close()
    Terminate the connection and release the connection handle.
    SQLDisconnect() and SQLFreeHandle() which are the ODBC standard functions, may be used.

11. SQLFreeHandle()
    Release ODBC environment handle.

```
int main( int aArgc, char** aArgv )
{
    SQLHENV       sEnv    = NULL;
    SQLHDBC       sDbc    = NULL;
    SQLINTEGER    sState  = 0;
    xa_switch_t * sXaSwitch;
    XID           sXid;
    sXaSwitch = SQLGetXaSwitch();
```

- If a user calls SQLAllocEnv() which is included in GOLDILOCKS ODBC

```
GOLDILOCKS_SQL_TRY( SQLAllocHandle( SQL_HANDLE_ENV,
                                    NULL,
                                    &sEnv ) );
    sState = 1;
```

- SQLSetEnvAttr sets attributes which govern aspects of environments.

```
GOLDILOCKS_SQL_TRY( SQLSetEnvAttr( sEnv,
                                   SQL_ATTR_ODBC_VERSION,
                                   (SQLPOINTER)SQL_OV_ODBC3,
                                   0 ) );
    if( (sXaSwitch->xa_open_entry)(
                "DSN=GOLDILOCKS;UID=test;PWD=test",
                0,
                TMNOFLAGS ) != XA_OK )
    {
        GOLDILOCKS_SQL_THROW( GOLDILOCKS_FINISH_LABEL );
    }
    sState = 2;
    sDbc = SQLGetXaConnectionHandle();
    sXid.formatID = 0;
    sXid.gtrid_length = 2;
    sXid.bqual_length = 1;
    memcpy( sXid.data,
            "100",
            sXid.gtrid_length + sXid.bqual_length );

    if( (sXaSwitch->xa_start_entry)( &sXid, 0, TMNOFLAGS ) != XA_OK )
    {
        GOLDILOCKS_SQL_THROW( GOLDILOCKS_FINISH_LABEL );
    }
```

- If SQL_SUCCESS which is an insert function succeeds

```
GOLDILOCKS_SQL_TRY( testInsert( sDbc ) );
```

- If SQL_SUCCESS which is a select function succeeds

```
GOLDILOCKS_SQL_TRY( testSelect( sDbc ) );
```

- If SQL_SUCCESS which is an update function succeeds

```
GOLDILOCKS_SQL_TRY( testUpdate( sDbc ) );
```

- If SQL_SUCCESS which is a delete function succeeds

```
GOLDILOCKS_SQL_TRY( testDelete( sDbc ) );

    if( (sXaSwitch->xa_end_entry)( &sXid, 0, TMSUCCESS ) != XA_OK )
    {
        GOLDILOCKS_SQL_THROW( GOLDILOCKS_FINISH_LABEL );
    }

    if( (sXaSwitch->xa_prepare_entry)( &sXid, 0, TMNOFLAGS ) != XA_OK )
    {
        GOLDILOCKS_SQL_THROW( GOLDILOCKS_FINISH_LABEL );
    }

    if( (sXaSwitch->xa_commit_entry)( &sXid, 0, TMNOFLAGS ) != XA_OK )
    {
        GOLDILOCKS_SQL_THROW( GOLDILOCKS_FINISH_LABEL );
    }

    sState = 1;
    if( (sXaSwitch->xa_close_entry)( "", 0, TMNOFLAGS ) != XA_OK )
    {
        GOLDILOCKS_SQL_THROW( GOLDILOCKS_FINISH_LABEL );
    }
    sDbc = NULL;
```

- SQLFreeHandleEnv releases resources which are related to an environment.

```
sState = 0;
    GOLDILOCKS_SQL_TRY( SQLFreeHandle( SQL_HANDLE_ENV,
                                  sEnv ) );
    sEnv = NULL;
    return EXIT_SUCCESS;
    GOLDILOCKS_FINISH;
    if( sDbc != NULL)
    {
        PrintDiagnosticRecord( SQL_HANDLE_DBC, sDbc );
    }
    if( sEnv != NULL)
    {
        PrintDiagnosticRecord( SQL_HANDLE_ENV, sEnv );
    }
    switch( sState )
```

```
    {
```

- Case 2: SQLDisconnect closes the connection which is related to a specific connection handle.

```
(void)(sXaSwitch->xa_close_entry)( "", 0, TMNOFLAGS );
```

- Case 1: SQLFreeHandleEnv releases resources which are related to an environment.

```
(void)SQLFreeHandle( SQL_HANDLE_ENV, sEnv );
            sEnv = NULL;
        default:
            break;
    }
    return EXIT_FAILURE;
}
```

32.

JDBC

32.1 Overview of GOLDILOCKS JDBC Driver

Concepts of GOLDILOCKS JDBC Driver

GOLDILOCKS provides GOLDILOCKS JDBC driver (excluding some features) which complies with standard JDBC 4.0 based on TCP/IP connection. The user can use various transaction features and data query features by using GOLDILOCKS JDBC driver and connecting to GOLDILOCKS in Java program. GOLDILOCKS JDBC driver is written and built based on JDK 1.6. Therefore, it supports JDBC 4.0 features. The driver can be used by adding $GOLDILOCKS_HOME/lib/goldilocks6.jar file to the class path.

The number after goldilocks refers to the JDK version. For more information, refer to **Supporting Versions**.

GOLDILOCKS JDBC complies with most of JDBC standard specifications, and it also supports non-standard API methods and classes to provide some unique features. For more information about non-standard methods, refer to each class API of **JDBC API References** or **Using Other Data Types**.

Characteristics

- **Type-4 JDBC Driver**

GOLDILOCKS JDBC is a JDBC Type-4 type which is implemented only with pure Java. A user can use a JDBC driver only with jar file without any additional libraries. Also, it is faster and reliable than JDBC-ODBC Bridge type, and it has better portability than Type-2 using Native API.

- **JDBC Standard Compliance**

GOLDILOCKS JDBC can recycle most of other existing JDBC programs without changing because it complies with the JDBC standard. The connection, various statements and ResultSet features can be used without modifying. However, the connection URL and property name, the name of driver class to be loaded should be changed to suit GOLDILOCKS. And the non-standard features and types are usable with the separate classes and methods API.

- **Supporting Various Java Versions**

GOLDILOCKS JDBC driver supports three files such as goldilocks8.jar, goldilocks7.jar, goldilocks6.jar, so that a user can select and use the file appropriate for user's Java run-time environment. Because each jar file was built based on JDK 1.8, JDK 1.7, JDK 1.6, it complies with each JDBC 4.2, JDBC 4.1, JDBC 4.0 specifications.

- **Flexible Version Compatibility with the Server**

Refer to the protocol version for the compatibility with the server and the JDBC driver. If the protocol vers ion of server is higher than that of the JDBC driver, it can be connected.

- **API Support for Connection Pooling**

JDBC connection object is a resource which is expensive to generate. Therefore, the JDBC standard define a pooling system for it, and ConnectionPoolDataSource and PooledConnection are its interfaces. GOLDIL OCKS JDBC implements these interfaces and provides the pooling facility in third party middleware produ cts.

- **Supporting XA API**

A user can perform the global transaction work by implementing XA interface which is the global transac tion standard. And a user can perform the various XA features complying with a standard by using XARes ource which is JDBC interface.

- **GOLDILOCKS-specific Data Type**

GOLDILOCKS uses the data types which are not provided by the JDBC standard. For example, they are Int erval related type, the type with time zone information such as Timestamp with time zone, etc. The driver provides a way to obtain or insert these types to DB.

- **Server-based Powerful Cursor Scroll Feature**

Other JDBC drivers cache the row set in the driver for ResultSet scroll, so uses excessively memory of a clie nt application program. For example, if the cursor is open by scroll insensitive and the rows are patched u p to the last, all rows are cached in the driver, and the entire table resides in the client memory. It causes t he out of memory error and the excessive use of client memory resource.

However, GOLDILOCKS supports the cursor scroll feature within the server, and clients may have lightwei ght, fast and reliable performance.

- **Efficient Use of Resources**

Because GOLDILOCKS minimizes the memory which maintains the row information compared to other JD BC drivers and it does not use Java objects as possible. It can avoid the excessive garbage collection, and perform fast and reliable table scanning with less memory.

- **Accurate and Extensive Metadata**

Because DatabaseMetaData interface of JDBC standard is faithfully and accurately implemented, it is easy to be linked with various DB tools. In addition, the usability is increased by retrieving DB meta information through various system views.

- **Powerful Logging**

Various logging features are provided to monitor the usual JDBC API calls and network usage as well as t he problem. If the logging-related features are specified in the connection URL, a user can leave a the con tent log to the console or files. Four types of logging exist, which are logging for JDBC method call record ing, logging for protocol sending and receiving, logging for the SQL statement used, logging for global c onnection usage.

- **Connection Failover**

Connection failover is supported on JDBC driver level to continuously use an existing connection by auto matically reconnecting to the previously registered alternate server when the connection with the executi on GOLDILOCKS server fails or is disconnected. A user can use connection failover feature as an existing J DBC program without any exception handling in preparation for the broken connection.

- **Connectivity between server and direct attach**

Other than TCP/IP based connection, it can be connected with direct attach method interworking with a process as same as the server. Like as ODBC connection supports direct attach method and Client/ Server method, GOLDILOCKS JDBC driver also provides both methods. JDBC program connected with direct atta ch does not communicate with TCP/IP, but it can use the server features in jvm by directly interworking w ith the server process. The perfomance is doubled or more than the JDBC program connected with TCP/ I P.

Supporting Versions

GOLDILOCKS JDBC versions

GOLDILOCKS JDBC version information can be viewed when executing goldilocks6.jar file as follows.

```
shell>java -jar goldilocks6.jar
 GOLDILOCKS JDBC Driver 1.1 Procotol-2.5.2, JDBC4.0 compiled with JDK1.6
```

The examples above describe that current GOLDILOCKS JDBC driver version is 1.1, and the protocol versio n is 2.5.2, and JDBC standard version is 4.0 and it is built in JDK 1.6. The driver version is displayed apart f rom GOLDILOCKS product version, and it goes up whenever the function is strengthened. For more infor mation about JDBC driver version, refer to **getDriverMajorVersion**, **getDriverMinorVersion**, **getDriverVersi on** of DatabaseMetaData.

Protocol version determines compatibility with the server, and the driver can interwork with the server if t he version is equal to or lower than the server protocol version. The server supports all clients API of the l ower protocol version.

goldilocks6.jar complies with JDBC 4.0 standard and it was built in JDK 1.6. goldilocks7.jar complies with JDBC 4.1 standard and it was built in JDK 1.7. goldilocks8.jar complies with JDBC 4.2 standard and it was built in JDK 1.8. Therefore, if the user's java environment is higher than JDK 1.8, use goldilocks8.jar. However, when using Java JDK 1.9 or higher, goldilocks8.jar can be used but API or classes of JDBC 4.3 can not be used.

Examples

Setting Class Path

CLASSPATH should be set to use GOLDILOCKS JDBC driver.

```
export CLASSPATH=.:$GOLDILOCKS_HOME/lib/goldilocks6.jar
```

Or, add a suitable jar file for the user's Java execution environment to the path.

Loading Driver Class

The driver class can be loaded as follows.

```
Class.forName("sunje.goldilocks.jdbc.GoldilocksDriver");
```

The example above is the conventional method of using JDBC, and it is the method of dynamically loading the driver class and registering in DriverManager and getting the connection. Nowadays, the way to get the connection through DataSource is used more. For more information, refer to the corresponding class in **JDBC API References**.

Getting Connection

The following code is used to get the connection.

```
Connection con = DriverManager.getConnection(
    "jdbc:goldilocks://127.0.0.1:22581/test", "TEST", "test");
```

The connection URL should be started with "jdbc:goldilocks:" to use GOLDILOCKS JDBC. The next is the IP address and port number of the server and "/test" which is the final part of URL is the DB name. The current GOLDILOCKS does not specifically check the DB name because it does not support multi DB. A user connected URL may be obtained again through DatabaseMetaData.getURL().

It connects in Direct Attach (D/A) mode when setting IP to 0.0.0.0, and a port to 0. Or, the connecting protocol, *da*, can be used instead of ip:port. In other words, both of the following two URL enables connect

ing in DA mode.

```
"jdbc:goldilocks://0.0.0.0:0/test"
"jdbc:goldilocks:da/test"
```

Username and password should respectively use the account and password for DB.

Using Statement and ResultSet

Statement and ResultSet are used in the same way as other JDBC programs.

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT NAME, ADDRESS FROM EMP");
while (rs.next())
{
    System.out.println("name = " + rs.getString(1));
    System.out.println("address = " + rs.getString(2));
}
rs.close();
stmt.close();
```

32.2 Feature Specification

Connection

Connection Using DriverManager

The traditional way to get a connection is to use DriverManager.

```
Connection con = DriverManager.getConnection(url_string, user_name, password);
```

url_string has the following three types.

```
jdbc:goldilocks://[ip address]:[port_no]/[db_name]
```

```
jdbc:goldilocks:da/[db_name]
```

```
jdbc:goldilocks:locator//[ip address]:[port_no][, [ip address]:[port_no] ]*/[db_name]
```

The host name, IPv4 and IPv6 are available for IP address, and port_no refers to the port number set. db_name is not used for the current connection so it can have any name. However, it can not be omitted. The following is an example of URL.

```
String url_string = "jdbc:goldilocks://localhost:22581/test";
```

```
String url_string = "jdbc:goldilocks://127.0.0.1:22581/test";
```

```
String url_string = "jdbc:goldilocks://[::1]:22581/test";
```

IP "0.0.0.0" and port 0 are used as a special address for D/A connection. For more information about D/A mode, refer to **Connecting in Direct Attach Mode**.

user_name and password refer to GOLDILOCKS account.

The following method is used for various property settings except for the getConnection() method above.

```
Properties prop = new Properties();
prop.setProperty("user", user_name);
prop.setProperty("password", password);
Connection con = DriverManager.getConnection(url_string, prop);
```

A locator keyword can be added to url_string to obtain the server connection information from glocator instead of using the connection property. In this case, ip address and port_no is considered as the connec tion information of glocator.
The following is an example of using the locator keyword to the connection statement.

```
String url_string =
"jdbc:goldilocks:locator//127.0.0.1:42581,127.0.0.1:42582/test?locator_service=S1";
```

The first connection information (127.0.0.1:42581) is the connection information of glocator, and the lat ter connection information is processed as ALTERNATE_LOCATORS property such as 127.0.0.1:42582.

The following is the property setting which has the meaning as same as the url_string above.

```
String url_strng = "jdbc:goldilocks://0.0.0.0:0/test";
Properties prop = new Properties();
prop.setProperty("locator_host", "127.0.0.1");
prop.setProperty("locator_port", "42581");
prop.setProperty("alternate_locators", "127.0.0.1:42582");
prop.setProperty("locator_service", "S1");
Connection con = DriverManager.getConnection(url_string, prop);
```

The property list which can be used for the connection is known through getPropertyInfo() method of Go ldilocksDriver. For more information, refer to **Table 32-1 Connection property**.

The login timeout of the connection object which is created by DriverManager and the logger uses the va lue registered in DriverManager. The logger is used by all JDBC interfaces generated from the connection object. Each connection object is not allowed to have an individual logger.

Connection Using DataSource

The JDBC standard recommends to use DataSource rather than the previous DriverManager. It is because DataSource is a single interface to access any data source, and it can individually set the values related to the various data sources, and may remotely send a DataSource object literally.

A connection object can be obtained through DataSource as follows.

```
import sunje.goldilocks.jdbc.GoldilocksDataSource;
GoldilocksDataSource ds = new GoldilocksDataSource();
ds.setServerName("127.0.0.1");
ds.setPortNumber(22581);
ds.setDatabaseName("test");
ds.setUser("TEST");
ds.setPassword("test");
```

```
Connection con = ds.getConnection();
```

GoldilocksDataSource classes should be used to create a DataSource object.
Then, various connection information should use setter method which is not DataSource standard API.For more information, refer to **DataSource**.

When using DriverManager, the login timeout and the logger should be globally set. However, when using DataSource, the login timeout and the logger can be individually set.

```
ds.setLoginTimeout(10);
ds.setLogWriter(out);
```

Interworking with Middleware

It is necessary to know the name of the class which implements XADataSource, ConnectionPoolDataSource interface to interwork with middleware such as Weblogic, JBoss, etc. GOLDILOCKS provides Goldilocks XADataSource, GoldilocksConnectionPoolDataSource for sunje.goldilocks.jdbc package. Both classes offer the various setter methods such as GoldilocksDataSource.

```
import sunje.goldilocks.jdbc.GoldilocksXADataSource;
GoldilocksXADataSource ds = new GoldilocksXADataSource();
ds.setServerName("127.0.0.1");
ds.setPortNumber(22581);
ds.setDatabaseName("test");
ds.setUser("TEST");
ds.setPassword("test");
XAConnection con = ds.getXAConnection();
```

Connection Property

Table 32-1 Connection property

Name	Mandatory/ optional	Valid value	Description
alternate_servers	Optional	IP:PORT[,IP:PORT]+	It is the list of alternate servers for failover. It is delimited by comma (,).
alternate_locators	Optional	IP:PORT[,IP:PORT]+	It is the list of alternate of glocator.
batch_count	Optional	Any integer	It is the number of the batch jobs which can be processed in a single protocol transmission and reception. The default value is 1,000. If the value is too small, too much frequent network communication causes poor performance of the batch processi

Name	Mandatory/ optional	Valid value	Description
			ng. If the value is too large, the server memory is increased be cause the session stacks the execution results.
connection_retry_count	Optional	Any integer	It stores the number of retrials to connect. The default value is 0, then it does not retry to connect.
connection_retry_delay	Optional	Any integer	It stores the delay before the connection retrial in seconds wh en trying to connect again. The default value is 3.
date_format	Optional	Any string	It is the character format which is used to interconvert betwee n date and string inside the driver.
da_buffer_size	Optional	Any integer	It sets the size of a buffer which sends and receives data when fetching and binding while connected in direct attach mode. T he default value is 100000.
decoding_replacement	Optional	Any string	It is the character replacing a byte value which can not be dec oded when decoding a byte array into a string. The default val ue is ?.
failover_granularity	Optional	{"0", "1", "2"}	It determines whether failover is successful. non-atomic(0), at omic(1), 2 are not yet supported. When an error occurs for th e existing prepared statements in the prepare process during f ailover, if it is 0, then it proceeds the failover, and if it is 1, it d etermines that the failover fails. The default value is 0.
failover_type	Optional	{"connection", "session"}	It determines the failover types. • Connection: Failover is used only when connecting to ser ver. • Session: Failover is used when connecting to server as we ll as communicating with the server like execution. The default value is session.
format_grammar	Optional	{"db", "java"}	It determines whether the property string such as date_format is the GOLDILOCKS syntax or the syntax used in SimpleDateFo rmat of java. The default value is db.
global_connection_log	Optional	boolean	It determines whether to perform the global connection loggi ng. The default value is false.
global_logger	Optional	{"console"}	It specifies the logging target. Currently only console is available. Only the first specified one i s valid.
home_dir	Optional	Any string	It specifies a home directory of a cluster server. The default val ue is null.
include_synonyms	Optional	boolean	It sets whether to include the synonym object in DatabaseMet aData.getColumns(). The default value is false.
keep_alive	Optional	boolean	It determines whether to set the keep_alive as the socket prop erty of the connection. Using the property, the connection re mains by periodically sending and receiving ack inside TCP soc ket. LAN cable error detection can forcibly cut off the connecti

Name	Mandatory/ optional	Valid value	Description
			on. The default value is false.
locality_aware_transaction	Optional	boolean	It determines whether to use GLOBAL CONNECTION. If it is set to true, then it uses GLOBAL CONNECTION. The default value is false.
locality_group_policy	Optional	{"0", "1", "2"}	It determines how to select a group if neither of groups are available, or two or more groups are available when using GLOBAL CONNECTION. • 0: It randomly selects the group. • 1: It sequentially selects groups which exist in LOCALITY_GROUP_PATH setting. If neither of groups in LOCALITY_GROUP_PATH are not available, it randomly selects the group. • 2: It sequentially selects groups. It always selects groups in an order of they are connected to the driver.
locality_group_path	Optional	Group name list	It defines the list of selected groups when the available group is not a single one when using GLOBAL CONNECTION. Each group is distinguished with comma (,). e.g. G1,G2,G3
locality_member_policy	Optional	{"0","1","2","3","4"}	It determines how to select a member in the selected group when using GLOBAL CONNECTION. • 0: DML : MASTER / SELECT : MASTER • 1: DML : ANY / SELECT : ANY • 2: DML : MASTER / SELECT : ANY • 3: DML : MASTER / SELECT : SLAVE • 4: It sequentially selects members which exist in LOCALITY_MEMBER_PATH setting. If neither of members in LOCALITY_MEMBER_PATH are not available, it uses the MASTER in the selected group.
locality_member_path	Optional	Member name list	It defines the list of members to be used in the selected group when using GLOBAL CONNECTION. Each member is distinguished with comma (,). e.g. G1N1,G2N1,G3N1,G1N2,G2N2,G3N2
locality_on_demand	Optional	boolean	It accesses to a specific member in need when using GLOBAL CONNECTION. The default value is false. • false: It accesses to all members from the beginning. • true: It accesses to a specific member when it is required.
locator_connection_timeout	Optional	Any integer	It is the time of waiting for receiving a packet from glocator.
locator_file	Optional	Any string	It is a location file.
locator_host	Optional	IP address	It is host address of glocator.
locator_port	Optional	Port no	It is port of glocator.

Name	Mandatory/ optional	Valid value	Description
locator_service	Optional	Any string	It is the service name to obtain the server connection informati on.
login_timeout	Optional	Any integer	It sets the timeout duration (second) of the socket when conn ecting to the server. The default value is 0, and it indefinitely waits.
lzeros	Optional	Any integer	When a numeric is expressed as a string and the number of ze ros following the decimal point exceeds this value, it is express ed in exponent notation. The default value is 15.
new_password	Optional	Any string	It can change the account password by using old_password pr operty together.
old_password	Optional	Any string	It can change the account password by using new_password property together.
packet_compre ssion_threshold	Optional	Any integer	It compresses the communication data to be sent to the server when the data size is bigger than packet_compression_thresh old. The property range is 32 ~ 2113929216.
password	Mandatory	Any string	It is user account password.
prefer_ipv6	Optional	boolean	It sets whether IPv6 takes precedence over other IP addresses on the host name. The default value is false.
program	Optional	Any string	It is program description.
protocol_log	Optional	boolean	It determines whether to log sending and receiving a protocol. The default value is false.
query_log	Optional	boolean	It determines whether to log a query. The default value is false.
role	Optional	{"", "SYSDBA", "ADMIN"}	It specifies the account role. The default value is "".
session_type	Optional	{"1", "2", "3"} or {"dedicate", "sh ared", "default" }	It is one of dedicated/ shared/ default. (It is selected by DB when it is set to default.)
statement_poo l_on	Optional	boolean	It enables the statement pool. The default value is false.
statement_poo l_size	Optional	Any integer	It sets the size of the statement pool.
time_format	Optional	Any string	It is the character format which is used to interconvert betwee n time and string inside the driver.
tcp_nodelay	Optional	boolean	It sets TCP_NODELAY(Nagle's Algorithm) property on the sock et of the connection. The default value is true.
timestamp_for mat	Optional	Any string	It is the character format which is used to interconvert betwee n timestamp and string inside the driver.
time_with_tim	Optional	Any string	It is the character format which is used to interconvert betwee

Name	Mandatory/ optional	Valid value	Description
e_zone_format			n time with timezone and string inside the driver.
timestamp_with_time_zone_format	Optional	Any string	It is the character format which is used to interconvert between timestamp with timezone and string inside the driver.
trace_log	Optional	boolean	It determines whether to log the trace. The default value is "" (not). The default value is false.
tzeros	Optional	Any integer	When a numeric is expressed as a string and the number of zeros in digit goes beyond this value, it is expressed in exponent notation. The default value is 15.
user	Mandatory	Any string	It is user account name.
use_global_session	Optional	boolean	It is whether to use GLOBAL SESSION. The default value is false.
use_targettype	Optional	{"0", "1", "2"}	It is the information which is to be received together when receiving a column type through communication. • 0: none • 1: name • 2: all

- locator_file is applied prior to locator_host and locator_port. For more information about locator_file, refer to **Location File**.
- locator_service property enables the access to the server belonging to locator_service. For more information, refer to **glocator** and **gloctl**.

Data Manipulation

Data Manipulation Using Statement

Various SQL statements can be executed by using statement objects. Statement object can be obtained from the connection object as follows.

```
Connection con = DriverManager.getConnection(...);
Statement stmt = con.createStatement();
```

Various DML or DDL statements can be executed by using the execute method of the created statement object, and the executeUpdate method.

```
stmt.executeUpdate("create table emp ( id varchar(20), name varchar(30), age integer )");
stmt.executeUpdate("insert into emp values ('1234560000', 'Yuna', 24)");
int updated = stmt.executeUpdate("delete from emp where age > " + age);
```

The difference between execute and executeUpdate methods is only the return values. The execute meth od indicates whether the executed SQL statement returns ResultSet. The executeUpdate method returns the number of updated rows. The execute method can be used for both of DML and SELECT statements, but it throws SQLException if the executeUpdate method is used for SELECT statement.

Direct-execution refers to execution of SQL statement by using statement. It is the method of p erforming the prepare operation(parsing, validation, optimization) and execution for the SQL st atement at a time. On the other hand, prepare-execution refers to the method of preparing the operation, then repeating the execution. Direct-execution repeatedly performs prepare and exe cution operation every time it is performed, so prepare-execution is more efficient when repeat edly performing the SQL statement.

Data Manipulation Using PreparedStatement

The JDBC standard provides PreparedStatement interface which can use a parameter for data manipulati on, and it enables a user to run it more efficiently. In addition, PreparedStatement quickly perform the re peated operation of the SQL statement because performs the prepare operation for the SQL statement o nly once.

```
Connection con = DriverManager.getConnection(...);
PreparedStatement pstmt = con.prepareStatement(
  "insert into emp values (?, ?, ?)");
pstmt.setString(1, "55544123000");
pstmt.setString(2, "Gildong");
pstmt.setInt(3, 32);
pstmt.executeUpdate();
pstmt.setString(1, "1357924680");
pstmt.setString(2, "Dooli");
pstmt.setInt(3, 41);
pstmt.executeUpdate();
```

After preparing the SQL statement as shown in the lines 2-3, the data is bound and performed. Prepared Statement which is prepared once, can bind and execute repeatedly.

If the binding is omitted and execute() is performed, the previously bound values are used.

```
pstmt.setString(1, "333555777000");
pstmt.setString(2, "Kang");
pstmt.setInt(3, 55);
pstmt.executeUpdate();
pstmt.setString(1, "245778884440");
pstmt.executeUpdate();
```

When the value is not bound to the second, third parameters as shown in line 6 but immediately execute d, the value, "Kang" and 55, which was previously bound, is used. If the previously bound value does not exist, it throws SQLException.

All bound values are deleted by using clearParameters().

```
pstmt.setString(1, "333555777000");
pstmt.setString(2, "Kang");
pstmt.setInt(3, 55);
pstmt.executeUpdate();
pstmt.clearParameters();
pstmt.setString(1, "245778884440");
pstmt.executeUpdate();
```

After deleting parameters in line 5, if the value is bound only to the first parameter and executed (line 7), then it throws SQLException.

The main reason to use PreparedStatement is because it can bind the data of various types. It is inconveni ent to handle the data of the type such as binary or timestamp because the value can be expressed only i n the character string when using Statement. Therefore, it is more convenient to bind all types by using P reparedStatement.

```
pstmt.setTimestamp(1,
  new Timestamp(Calendar.getInstance().getTimeInMillis()));
pstmt.setCharacterStream(2, new StringReader(BIG_STRING));
pstmt.setObject(3, someObj, Types.LONGVARCHAR);
pstmt.setObject(4, otherObj);
```

In the sample above, lines 1 ~ 2 binds java.sql.Timestamp object. The binding type (It is GOLDILOCKS typ e of the data when the data is sent to the server) is TIMESTAMP. For more information about the binding type which is determined by the various setter methods, refer to the corresponding API of **PreparedState ment**.

Line 3 binds a reader object, and it is bound as LONG VARCHAR type internally.

Line 4 binds the object of Java object type, and explicitly notifies that the type is LONG VARCHAR. For mo re information about GOLDILOCKS type which is mapped to the type of Types, refer to **SQL types → GOL**

DILOCKS types.

Line 5 literally binds the Java object type, and it is bound to the corresponding GOLDILOCKS data type ac cording to the class type. For more information about mapping between class types and GOLDILOCKS da ta types, refer to **Table 32-2 Java objects → GOLDILOCKS types**.

Batch Execution Using Statement

The different SQL statements can be executed as the batch job by using addBatch() and executeBatch() o f Statement.

```
stmt.addBatch("insert into emp values ("12345", "Jake", 22)");
stmt.addBatch("insert into salary values ("12345", 5000)");
stmt.addBatch("update members set total_count=total_count+1 where age=22");
stmt.executeBatch();
```

Likewise, different SQL statements may be performed through a single executeBatch () method. However, this does not mean that the method call executes three statements in the server, gathers the results, send s the results to the JDBC driver with a single protocol. Internally, three times of protocol transmission, exe cution in the server and result transmission are performed. In other words, it does not have a big perform ance advantage.

After performing executeBatch, all registered jobs are deleted.

Even if an error occurs during execution, all registered jobs are performed until it is completed. Whether an error occurs or not can be known by the value of the returned int [] type. In other words, if int [] has E XECUTE_FAILED value, it means that the job is failed.

Batch Execution Using PreparedStatement

More powerful batch operation can be performed by using addBatch() and executeBatch() methods of Pr eparedStatement.

```
PreparedStatement pstmt = con.prepareStatement("insert into emp values (?, ?, ?)");
pstmt.setString(1, "12345000");
pstmt.setString(2, "John");
pstmt.setInt(3, 33);
pstmt.addBatch();
pstmt.setString(1, "222333555");
pstmt.setString(2, "Henry");
pstmt.setInt(3, 24);
pstmt.addBatch();
int[] inserted = pstmt.executeBatch();
```

The example above is a process of inserting two rows. The values configuring the two rows are bound thr ough addBatch (), and then if executeBatch () is called, the bound value and the protocol for execution a re sent to the server and executed. It is more efficient than the batch execution of Statement and the perf ormance is faster because communication occurs only when executeBatch() is called.

The new value should be bound to execute executeBatch() again because the bound values are deleted a fter executing executeBatch(). clearBatch() does not be to be explicitly called to delete the registered job after execution.

Even if an error occurs during execution, all registered jobs are performed until it is completed. Whether an error occurs or not can be known by the value of the returned int [] type. In other words, if int [] has E XECUTE_FAILED value, it means that the job is failed.

GOLDILOCKS JDBC provides a separate method which is executeBatchAtomic() in addition to executeBat ch(). executeBatch () performs the execution as many as the number of the registered jobs in the server, but the response time is fast because executeBatchAtomic() performs the execution at once with the regi stered job (the bound values). However, if any one fails, everything fails as it can be seen from the name which is atomic. Therefore, the return type is not int[], but int.

```
...
int inserted = ((GoldilocksPreparedStatement)pstmt).executeBatchAtomic();
```

Data Retrieval

Getting Column Values

The data can be retrieved through ResultSet object which is obtained by executeQuery () of the Statemen t or PreparedStatement.

```
...
ResultSet rs = pstmt.executeQuery();
while(rs.next())
{
    String id = rs.getString(1);
    String name = rs.getString(2);
    int age = rs.getInt(3);
    ...
}
```

The contents of the column can be obtained through various getter methods in the ResultSet after retriev ing the data in the table. The data in the table is sent to JDBC drivers in the original form of GOLDILOCKS data type, and it is converted to an appropriate Java data type according to the types of getter methods c

alled by a user and it is transmitted to the user. For more information about type conversion mapping of GOLDILOCKS data type and getter method, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .

If GOLDILOCKS data type can not be converted to the type of getter, it throws SQLException.

Closing ResultSet

The ResultSet which is used up can release the resources through close(). Even if a user does not explicitly call close(), the followings bring the result as same as when calling close() of ResultSet.

- When superordinate statement is closed
- When superordinate connection is closed
- When executeQuery() of Statement is called again
- When an error occurs during fetching

The two ResultSets which are created from a single statement can not be remained simultaneously in the third case above. Only the ResultSet object which is created by the last executeQuery() is valid.

Fetch Size

JDBC can specify the number of rows fetched at once from the server through setFetchSize(int rows) of S tatement. The default value of the ResultSet property of GOLDILOCKS is 0. 0 automatically determines th e number of rows fetched by the server. For forward only, it is the maximum number of rows fetched in a communication packet. For scrollable, it is fixed to 100. If the value is too large, the amount of memory u sed by JDBC ResultSet becomes large. If the value is too small, the communication is frequently executed.

The value is mainly used in a scrollable ResultSet because it is not sensitive even when it is scroll sensitive while moving within the row cache of ResultSet. If the value is set too large, the latest information about the changes of rows can not be known.

Field Size Limit

JDBC may limit the maximum length of the column through setMaxFieldSize(int size) of statement. The maximum length can be limited for the types of CHAR, VARCHAR, LONG VARCHAR, BINARY, VARBINAR Y, and LONG VARBINARY. The data bigger than the length is truncated. The default value is 0, and the m aximum length is not limited in this case.

This property is ignored for other types such as INTEGER, DATE, etc..

ResultSet Scroll

Getting Scrollable ResultSet

There are two kinds of scrollable ResultSet, which are scroll sensitive and scroll insensitive. Scroll sensitive is the cursor type which can view the values changed by the transaction itself or other transaction while tr aversing the ResultSet, but scroll insensitive is not. GOLDILOCKS server supports both of the scroll types, a nd those facilities can be used by JDBC.

```
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                                     ResultSet.CONCUR_READ_ONLY);
ResultSet rs = stmt.executeQuery("select id, name, age from emp");
```

If the ResultSet type is TYPE_SCROLL_INSENSITIVE when creating statement as line 1, ResultSet which is c reated from the statement object is scroll sensitive cursor. However, the current GOLDILOCKS does not su pport the updatable ResultSet.

The type should be set to TYPE_SCROLL_INSENSITIVE to obtain scroll insensitive ResultSet. The default val ue is TYPE_FORWARD_ONLY.

However, even it is a scroll sensitive ResultSet, a user can not know the latest value or whether it is chang ed, when moving within the row cache of ResultSet. The row should be fetched again from the server to find out the latest value or whether it is changed.

Scrolling

JDBC API provides the following methods to scroll the cursor.

```
public boolean next() throws SQLException;
public boolean previous() throws SQLException;
public boolean first() throws SQLException;
public boolean last() throws SQLException;
public boolean absolute(int rows) throws SQLException;
public boolean relative(int rows) throws SQLException;
public void beforeFirst() throws SQLException;
public void afterLast() throws SQLException;
```

next() is the method which can be used for forward only or scrollable ResultSet, and it sets the cursor pos ition to the next row. If the row can be read at the moved position, it is true. Otherwise, it is false. Other methods can be used only for scrollable ResultSet.

previous() moves the cursor to the previous row, first() moves the cursor to the first row, last() moves the

cursor to the last row, absolute() moves the cursor to the absolute position and relative() moves the curso r to the relative position from the current position. beforeFirst() moves the cursor to before the first row. afterLast()moves the cursor to after the last row.

```
rs.afterLast();
while (rs.previous())
{
    String id = rs.getString(1);
    String name = rs.getString(2);
    int age = rs.getInt(3);
    ...
}
```

The code above will query backwards from the last row to the first row.

The current cursor position can be known by the following method.

```
public int getRow() throws SQLException;
```

Scrolling Principle

The ResultSet scroll is easy to use, but it may cause poor performance without knowing internal operatin g principles. Scrolling is performed in result cache within ResultSet of JDBC driver, but it is performed on t he server in case when the range exceeds the cache, so the communication may occur. Therefore, it shou ld be used carefully to prevent the poor performance.

The internal default value of the fetch size of scrollable ResultSet is 100 (the external default value is 0 an d it can be changed through setFetchSize() method). It means that the number of rows that are fetched t o JDBC driver at once are 100. The row cache of ResultSet consists of 100 rows, and the cursor position o f ResultSet should be changed if the row position to be moved is in the cache. However, if it is not in the cache, the row at the position where to be moved should be got from the server. At this time, it is necess ary to know how to get the row set including the row to be moved.

For example, if the ResultSet has rows from 1 to 100 and the current cursor position is at 100, calling nex t () will fetch rows 101 through 200 from the server because the next row does not exist in the cache. On the other hand, the rows from 101 to 200 are in ResultSet cache, and it is not appropriate to fetch rows f rom 100 to 199 from the server by calling previous() if the current cursor position is 101 row. When this approach is used, row should be fetched from the server whenever previous() is called. However, GOLDIL OCKS fetches the rowset which includes the corresponding last row from the server when it is required to fetch the row from the server due to calling previous () to prevent this inefficiency. For example, when it i s necessary to fetch row of the number 100 by calling previous(), GOLDILOCKS fetches the rows from 1 t o 100. Then it is fetched in favor of previous ().

The rule is generalized as follows.

- If the position to be moved is after the current position, 100 rows (the fetch size) after that position a re fetched.
- If the position to be moved is before the current position, 100 rows before that position are fetched.
- When moved with first(), the first 100 rows are fetched.
- When moved with last(), the last 100 rows are fetched.

When moved backward, the fetch is performed in favor of next(). When moved forward, the fetch is perf ormed in favor of previous(). This is also applied to absolute () and relative ().

Using Other Data Types

Interval Type

GOLDILOCKS supports the following 13 types related to interval.

- interval year
- interval month
- interval year to month
- interval day
- interval hour
- interval minute
- interval second
- interval minute to second
- interval hour to minute
- interval hour to second
- interval day to hour
- interval day to minute
- interval day to second

sunje.goldilocks.jdbc.GoldilocksInterval object is created and the insert operation is executed by setObjec t() of PreparedStatement to insert the interval data into a table using JDBC.

GoldilocksInterval object can be created by using createIntervalXXX which is the static method of Goldilo cksInterval.

```
import sunje.goldilocks.jdbc.GoldilocksInterval;
...
GoldilocksInterval interval = GoldilocksInterval.createIntervalDayToSecond(2, 6, "2
08:23:54.560843");
```

```
PreparedStatement pstmt = con.prepareStatement("insert into interval_table values (?,?)");
pstmt.setString(1, someId);
pstmt.setObject(2, interval);
pstmt.executeUpdate();
```

For more information about createIntervalXXX method, refer to **GoldilocksInterval**. Likewise, GoldilocksInterval is created and it is bound via setObject. Or, a type is explicitly specified as follows.

```
import sunje.goldilocks.jdbc.GoldilocksTypes;
...
pstmt.setObject(2, interval, GoldilocksTypes.INTERVAL_DAY_TO_SECOND);
```

The data may be inserted by using a character string without GoldilocksInterval object.

```
pstmt.setObject(2, "2 08:23:54.560843", GoldilocksTypes.INTERVAL_DAY_TO_SECOND);
```

In this case, a GoldilocksInterval object is created within JDBC driver and bound to a host variable. Or it may be bound as a string by using setString() method.

```
pstmt.setString(2, "2 08:23:54.560843");
```

The string is literally sent to the server when using this method, and the string is converted to Interval day to second type and inserted in the server.

getObject() or getString() of ResultSet queries the Interval type from the DB.

```
import sunje.goldilocks.jdbc.GoldilocksInterval;
ResultSet rs = stmt.executeQuery("select interval_value from some_table");
while (rs.next())
{
    GoldilocksInterval interval = (GoldilocksInterval)rs.getObject(1);
    System.out.println("type = " + interval.getTypeName() +
                       ", value = " + interval.toString());
}
```

The data of GoldilocksInterval type can be retrieved via getObject() of ResultSet. GoldilocksInterval class provides getter like getYear(), getHour(), etc. that return various time data. For more information about getter API, refer to **GoldilocksInterval**.

Time with Time Zone and Timestamp with Time Zone Types

Concerning time, GOLDILOCKS provides not only SQL standard types such as Date, Time, Timestamp but also Time with time zone and Timestamp with time zone types.

```
CREATE TABLE SAMPLE_TABLE ( C1 TIME WITH TIME ZONE,
                            C2 TIMESTAMP WITH TIME ZONE );
```

GOLDILOCKS JDBC provides setTimeTimeZone(int colIndex, Time time, Calendar timezone) method and setTimestampTimeZone(int colIndex, Timestamp time, Calendar timezone) method in GoldilocksPreparedStatement class to insert the data of Time with time zone and Timestamp with time zone types. For more information about specifications refer to **setTimeTimeZone** and **setTimestampTimeZone**.

The data can be inserted via existing setTime() and setTimestamp() methods, but the time zone value of the column is unconditionally set to the local time zone of Java execution environment. Therefore, setTimeTimeZone() or setTimestampTimeZone should be used to insert various time zone information.

Time, Timestamp data can be inserted with the given time zone value via setTimeTimeZone and setTimestampTimeZone.

```
import sunje.goldilocks.jdbc.GoldilocksPreparedStatement;
PreparedStatement pstmt = con.prepareStatement(
    "INSERT INTO SAMPLE_TABLE VALUES (?,?,?,?)");
Calendar now = Calendar.getInstance();
Calendar usNow = Calendar.getInstance(TimeZone.getTimeZone("GMT-8"));
Time t = new Time(now.getTimeInMillis());
Timestamp ts = new Timestamp(now.getTimeInMillis());
pstmt.setTime(1, t);
pstmt.setTimestamp(2, ts);
pstmt.executeUpdate();
((GoldilocksPreparedStatement)pstmt).setTimeTimeZone(1, t, now);
((GoldilocksPreparedStatement)pstmt).setTimestampTimeZone(2, ts, now);
pstmt.executeUpdate();
((GoldilocksPreparedStatement)pstmt).setTimeTimeZone(1, t, usNow);
((GoldilocksPreparedStatement)pstmt).setTimestampTimeZone(2, ts, usNow);
pstmt.executeUpdate();
```

Line 11, 12 binds the column to time type and timestamp type. Each value of t and ts is sent to the server and the default time zone value is inserted to DB column in the server.

On the other hand, line 15, 16 binds the column to time with time zone type and timestamp with time zone type, and is sent to the server together with time zone information. As a result, if the time zone settings of a server and a client are same, the rows which are inserted by lines 11~13 and 15~17 are same.

Line 19, 20 inserts time zone information of GMT-8 time zone.

There are two ways to query data, which are time with time zone and timestamp with time zone. One is t he way to get data as char from the server, and the other is the way to get data of the corresponding typ e by JDBC and to obtain as time or as timestamp object.

```
ResultSet rs = stmt.executeQuery("select c1, cast(c1 as char(33)), c2, cast(c2 as char(33))
from sample_table");
while (rs.next())
{
    System.out.println("c1 = " + rs.getTime(1).toString());
    System.out.println("c1 as char = " + rs.getString(2));
    System.out.println("c2 = " + rs.getTimestamp(3).toString());
    System.out.println("c2 as char = " + rs.getString(4));
}
```

When executing the code above, the results are as follows.

```
c1 = 12:35:26
c1 as char = 12:35:26.052000 +09:00
c2 = 2014-03-27 12:35:26.052
c2 as char = 2014-03-27 12:35:26.052000 +09:00
c1 = 12:35:26
c1 as char = 12:35:26.052000 +09:00
c2 = 2014-03-27 12:35:26.052
c2 as char = 2014-03-27 12:35:26.052000 +09:00
c1 = 12:35:26
c1 as char = 19:35:26.052000 -08:00
c2 = 2014-03-27 12:35:26.052
c2 as char = 2014-03-26 19:35:26.052000 -08:00
```

REF CURSOR Type

REF CURSOR which was used as PARAMETER in PSM is available in JDBC as well.

```
CREATE OR REPLACE PROCEDURE proc_cursor1( p1 OUT SYS_REFCURSOR )
AS
  var1 INTEGER;
BEGIN
  OPEN p1 FOR SELECT r_c1 FROM r;
  FETCH p1 INTO var1;
  DBMS_OUTPUT.PUT_LINE(var1);
```

```
END;
/
```

```
CREATE OR REPLACE PROCEDURE proc_cursor2( p1 IN SYS_REFCURSOR )
AS
  var1 INTEGER;
BEGIN
  LOOP
    FETCH p1 INTO var1;
    EXIT WHEN p1%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(var1);
  END LOOP;
  CLOSE p1;
END;
/
```

Set sqlType typr to Types.REF_CURSOR (JDK 1.8 or higher) or GoldilocksTypes.REF_CURSOR (lower than JDK 1.8) to use REF CURSOR as OUTPUT PARAMETER in JDBC.

```
import import java.sql.Types;
CallableStatement cstmt1 = con.prepareCall("{CALL proc_cursor1(?)}");
cstmt1.registerOutParameter(1, Types.REF_CURSOR);
cstmt1.execute();
```

```
import sunje.goldilocks.jdbc.GoldilocksTypes;
CallableStatement cstmt1 = con.prepareCall("{CALL proc_cursor1(?)}");
cstmt1.registerOutParameter(1, GoldilocksTypes.REF_CURSOR);
cstmt1.execute();
```

ResultSet object is received through CallableStatement.getObject() after executing CallableStatement.

```
ResultSet rs = (ResultSet)cstmt1.getObject(1);
if (!rs)
{
    if (rs.next())
    {
        System.out.println(rs.getInt(1));
    }
}
```

Set ResultSet object which was received through CallableStatement.getObject() to PreparedStatement.setObject() to use REF CURSOR received as OUTPUT PARAMETER as INPUT PARAMETER in the new statem

ent.

```
CallableStatement cstmt2 = con.prepareCall("{CALL proc_cursor2(?)}");
cstmt2.setObject(1, rs);
cstmt2.execute();
```

It is fetched as may times as the value set with Statement.setFetchSize() when executing ResultSet.next(). When using REF CURSOR received as OUTPUT PARAMETER as INPUT PARAMETER in the new statement, then set the fetch size of CallableStatement which uses OUTPUT PARAMETER.

Logging

Logging Types

When developing a project by using JDBC, it is helpful in many ways for JDBC driver to leave various logs. GOLDILOCKS provides the facility to log the useful information even during operation, as well as the project development.
There are three types of log, which are trace log, protocol log and query log.

Trace log leaves information every time when JDBC API is called. It can be known which JDBC API is called. Protocol log shows the situation to send and receive communication packets between JDBC driver and GOLDILOCKS server. Query log records the SQL statement to be executed, when PreparedStatement or Statement is executed.

Logging by Using DriverManager

First, the logging is left by using DriverManager. The logging media can be determined by using setLogWriter() method in DriverManager, and the type of logging to be left can be specified in connection url.

```
DriverManager.setLogWriter(new PrintWriter(System.out));
String url = "jdbc:goldilocks://localhost:22581/test?trace_log=on&query_log=on";
Connection con = DriverManager.getConnection(url, "TEST", "test");
```

Line 1 defines the logging media. All logging is output to the console. Line 2 defines the connection url, and the property can be defined after ? character. It means to using trace_log, query_log.

Likewise, these properties can be specified in URL, or they can be defined with properties object.

```
Properties prop = new Properties();
prop.put("trace_log", "on");
prop.put("query_log", "on");
prop.put("protocol_log", "on");
prop.put("user", "TEST");
prop.put("password", test");
Connection con = DriverManager.getConnection(url, prop);
```

The connection properties which can be used in GOLDILOCKS are defined in **Table 32-1 Connection property**.

Sometimes it is difficult to call setLogWriter() of DriverManager. When using the middleware, the code like that can not be added. For such a case, GOLDILOCKS JDBC provides a global property called global_logger. Other general properties are limited to a single connection, but this property is global. In other words, if the property is set, it does not have to call DriverManager.setLogWriter ().

```
String url = "jdbc:goldilocks://localhost:22581/test?" +
             "global_logger=console&trace_log=on&query_log=on";
Connection con = DriverManager.getConnection(url, "TEST", "test");
```

global_logger property is applied only once initially, and it is ignored if log writer is already set in DriverManager. Only the current console is applicable as the property value. Other value does not cause any operation.

Logging by Using DataSource

Unlike DriverManager, DataSource can set the log writer by each object. If the log writer is set for a DataSource object, all connections which are generated from the DataSource are logged by the log writer. The following describes how to set the logging in the DataSource.

```
import sunje.goldilocks.jdbc.GoldilocksDataSource;
...
GoldilocksDataSource ds = new sunje.goldilocks.jdbc.GoldilocksDataSource();
ds.setServerName("localhost");
ds.setDatabaseName("test");
ds.setUser("TEST");
ds.setPassword("test");
ds.setPortNumber(22581);
ds.setLogTarget("console");
ds.setTraceLog("on");
ds.setQueryLog("on");
ds.setProtocolLog("on");
```

```
Connection con = ds.getConnection();
```

Likewise, the logging feature is set by using setLogTarget(), setTraceLog(), setQueryLog() and setProtocol Log() methods. setLogWriter() of DataSource can be called instead of setLogTarget(). When using the mi ddleware, the method such as setLogWriter() can not be directly called, so it should be controlled with th e property. In this case, if logTarget, traceLog, queryLog and protocolLog properties are defined, the mid dleware automatically calls the methods.

Viewing Plan Text

Usage

The plan text can be obtained by using GOLDILOCKS JDBC like as the plan text of the executed statement is obtained by using GOLDILOCKS ODBC. It can be configured to generate a plan text by using non-stand ard API method in GoldilocksStatement class, or obtain the plan text which is already generated.

```
Connection con = ...
GoldilocksStatement stmt = (GoldilocksStatement)con.createStatement();
stmt.setExplainPlanOption(GoldilocksStatement.EXPLAIN_PLAN_OPTION_ON);
ResultSet rs = stmt.executeQuery("select * from t1");
System.out.println(stmt.getExplainPlan());
```

For example, when querying a simple table, the plan text is output as follows.

```
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                                |       ROWS |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                |            |
|    1  |    TABLE ACCESS ("T1")                           |          0 |
================================================================================
     1  -  READ COLUMNS : A
```

Option Types

GOLDILOCKS provides the following four properties about the plan text generation.

- EXPLAIN_PLAN_OPTION_OFF: It does not generate the plan text as the default value of the property r elated to the plan text of statement.
- EXPLAIN_PLAN_OPTION_ON: It generates the plan text when executing or fetching.

- EXPLAIN_PLAN_OPTION_ON_VERBOSE: It generates more detailed plan text such as execution time when executing or fetching.
- EXPLAIN_PLAN_OPTION_ONLY: It generates the plan text like as EXPLAIN_PLAN_OPTION_ON when executing or fetching but it is not actually executed.

These properties are set by using GoldilocksStatement.setExplainPlanOption(int) method, and the property which is set once is continuously maintained. A constant value of the property is defined in GoldilocksStatement.

The plan text for non-SELECT DML statements or other SQL statements is generated at run-time, but the plan text for the SELECT statements is generated when the SELECT statement is fetched and the cursor is positioned at the end in the server. The plan text should be obtained after all rows are traversed with ResultSet for the SELECT statement. The plan text can be obtained when there are few rows in a table because the cursor can traverse until the end in the server without fetching all rows.

Connection Failover

GOLDILOCKS supports the connection failover of the client level (JDBC, ODBC). The application which accesses with client/ server mode by using the JDBC or ODBC to GOLDILOCKS, may perform connection to the alternative server automatically when the connection fails or disconnected. This allows a user to use high availability applications which utilize multiple servers. In particular, when a user registers the main server URL and the alternative server URL through the connection property under interworking web server environment, it can safely use the connection object obtained from the connection pool without worrying about the interruption. Failover is performed entirely automatically inside the driver and the user does not need to worry about the reconnection logics.

alternate_servers property should be given to use failover. IP and port of the alternative server are given for this property, and it may indicate the multiple alternative servers by using the comma (,).

```
Class.forName("sunje.goldilocks.jdbc.GoldilocksDriver");
String url = "jdbc:goldilocks://192.168.0.101:22581/test";
Property prop = new Properties();
prop.setProperty("user", "TEST");
prop.setProperty("password", "test");
prop.setProperty("alternate_servers", "192.168.0.201:22581");
Connection con = DriverManager.getConnection(url, prop);  ❶
Statement stmt = con.createStatement();
stmt.executeUpdate("insert into time_tab values (sysdate)");  ❷
...
```

For example, if ❶ fails to connect to the main server (192.168.0.101), it will automatically connect to an alternative server (192.168.0.201) and returns a connection object. Any error is not returned to the user. If it can not communicate with the server because the connection is broken at ❷ during executeUpdate () or before then, it automatically connects to the alternative server inside the driver and continues to use the existing connection object and returns SQLException. Users can continue to use the existing connecti on object after processing SQLException.

Whether failover or not can be viewed through getWarnings() of connection object.

A failover feature of GOLDILOCKS JDBC is that a user can continue to use statement object or PreparedSt atement object created from connection object. (ResultSet can not continue to be used.)

In particular, a user can continue to use the object because PreparedStatements perform the prepare ope rations internally again in the alternative server if a failover occurs.

```
Connection con = DriverManager.getConnection(url, prop);
PreparedStatement pstmt =
    con.prepareStatement("insert into t1 values (?,?)");
...
try
{
    pstmt.setInt(1, 100);
    pstmt.setString(2, "John");
    pstmt.executeUpdate();
}
catch (SQLException sException)
{
    if (sException.getErrorCode() == 21012)
    {
```

- Failover occurs due to the communication link failure error.
- A user can continue to use the pstmt object.

```
pstmt.executeUpdate();
    }
    else
    {
        ...
    }
}
```

GOLDILOCKS JDBC driver provides some properties related to failover. These can help a user to decide th e failover type, connection order and simple policy used for failover.

failover_type

The property is used to select the failover type. In the example above, the failover at ❶ is the connection failover and the failover at ❷ is the session failover. If the property value is *connection*, only the connecti on failover is used, and if it is *session*, both of the connection failover and session failover can be used. Th e default value is *session*.

failover_granularity

When failover occurs, the prepare operation is performed after the PreparedStatement objects which are generated from the current connection object are connected to the alternative server. If the prepare oper ation fails in the alternative server (An error may occur due to the different server environment.), the pro perty is used to determine whether to consider the failover failed or to ignore the prepare error. The prop erty value is either 0 or 1. If the property value is 0, the prepare error is ignored and the failover continuo usly proceeds. If the property value is 1, the failover is failed. The default value is 0. If the value is 0 and th e prepare operation is failed, PreparedStatement object can not continue to be used and the user should directly create the PreparedStatement object again.

Connecting in Direct Attach Mode

Since JDBC 1.1, Goldilocks provides the connection in direct attach mode (D/A mode) besides the existin g connection in Client/ Server mode (C/S mode) based on TCP/ IP. Like as ODBC D/A connection mode, t his is operated directly interworking with the server process. so the server module interworks within a sin gle process (in a process same as jvm). Therefore, the remote host can not connect in D/A mode.

D/A mode is designed to utilize the merit of GOLDILOCKS, which is a fast processing. TCP/ IP based conn ection offsets the fast processing of GOLDILOCKS by expensive cost, so an alternative method is required when a fast processing is required. Though it is restricted to be operated within the host which is same as the host of GOLDILOCKS server, this D/A mode connection is a good solution.

JDBC program connecting in D/A mode can directly call the server feature by loading GOLDILOCKS jni libr ary when DB connection is created first within jvm. A server module can be directly called through native interface in jvm without network cost, it enables faster processing than the existing C/S mode.

Connecting Method

Use *0.0.0.0:0* instead of existing ip:port in the connection URL to connect in D/A mode. The existing URL can be literaly used and using a special ip:port address (0.0.0.0:0) can minimize changing exisiting applic ations. Or, use a special protocol (*da*) to connect in D/A mode.

```
Connection con =
 DriverManager.getConnection("jdbc:goldilocks://0.0.0.0:0/test","TEST","test");
```

or,

```
Connection con =
 DriverManager.getConnection("jdbc:goldilocks:da/test", "TEST", "test");
```

Features of D/A Connection

GOLDILOCKS server module can be directly called in jvm when connecting in D/A mode. It is called throu gh JNI (Java Native Interface) between JDBC program and GOLDILOCKS server module. It uses the server module with minimum call cost considering that it is expensive to call JNI, so it is faster double than the J DBC based on existing TCP/ IP.

When using the connection based on the existing TCP/ IP, it uses only goldilocks6.jar file. However, when using the connection based on D/A mode, it uses libgoldilocksjni.so file and libgoldilocksas.so file in $GO LDILOCKS_HOME/lib by dynamically loading them. Therefore, an error occurs when connecting if those t wo library files do not exist. However, the location of the library files need not to be separately specified when operating java program. Because it searches for those two library files and loads them as long as it i s in the directory as same as the directory of goldilocks6.jar file.

Global Connection

It support the global connection feature. When using the global connection, an application selects a nod e appropriate for a query process and performs it in the cluster environment.

Settings

locator_file or locator_host, locator_port should be set together with locality_aware_transaction property value to use the global connection. use_global_session property value should be set to 1 to use the global connection.

```
Properties prop = new Properties();
prop.put("locality_aware_transaction", "1");
prop.put("locator_file", "/home/goldilocks/.location.ini");
prop.put("user", "TEST");
prop.put("password", test");
Connection con = DriverManager.getConnection(url, prop);
```

```
String url = "jdbc:goldilocks://192.168.0.1:22581/test?" +
             "locality_aware_transaction=1&locator_host=192.168.0.2&locator_port=42581";
Connection con = DriverManager.getConnection(url, "TEST", "test");
```

Procedure

1. Connect to the database.

After connecting to the server input by a user and receiving the information of the cluster system, the cluster system information is built through the locator file or glocator, then it connects to all nodes in the cluster system.

2. Execute the statement.

If the cluster system information is not built, then the cluster system information is built through the locator file or glocator, then it connects to all nodes in the cluster system.

When connecting to a new server by adding a node to the cluster, then it creates all statements of another node on the node, and is ready to execute SQL.

The statement class does not have the information about the sharding key, so the node is selected according to locality_group_policy or locality_member_policy property and the statement is executed.

If PreparedStatement or CallableStatement class does not have the information about the sharding key, then it builds the information about the sharding key of the SQL from an arbitrary server. If the information about the sharding key is built, then an appropriate node is selected by using the information about the sharding key or locality_group_policy, locality_member_policy, and the query is executed.

If an error occurs on the selected node, then it selects an appropriate node again except for that node, and executes the query.

If the sharding information is updated after executing the statement, then the built sharding key information is dropped.

If the cluster system information is updated by adding/ dropping the cluster node after executing the statement, then the built cluster system information is dropped.

3. Receive ResultSet data.

It receives the data from the node where SQL was executed.

4. Close ResultSet.

It closes a cursor on the node where SQL was executed.

5. Close the statement.

It releases the statement from all connected node.

6. Close the database.

It releases connection to all nodes.

Exception Handling for High Availability

When using GLOBAL CONNECTION, if an error occurs on the selected node during the operation, then it is operated as follows according to the transaction occurrence and SELECT progress.

- When a transaction did not occur

If an error occurs on the selected node when a transaction did not occur, then it selects another node wit hin JDBC and executes the query. Though an error occurred on the selected node, the query was normall y executed on another node, so it does not transfer an error to a user.

- When a transaction occurred or SELECT is in progress

If an error occurs on the selected node when a transaction occurred or ResultSet.next() is in progress, the n JDBC can not proceeds the current operation any more so it transfers 21047(Retry the transactional op erations again). If 21047 error occurs, then a user should perform the transaction or SELECT again.

```
PreparedStatement pstmt = con.prepareStatement("INSERT INTO T1 VALUES (?)");
boolean retry;
do
{
    retry = false;
    try
    {
        pstmt.setInt(1, 1);
        pstmt.executeUpdate();
    }
    catch (SQLException e)
    {
        if (e.getErrorCode() == 21047)
        {
            retry = true;
        }
        else
        {
            throw e;
        }
```

```
    }
} while (retry == true);
```

```
PreparedStatement pstmt = con.prepareStatement("SELECT * FROM T1 WHERE C1 = ?");
boolean retry;
do
{
    retry = false;
    try
    {
        pstmt.setInt(1, 1);
        ResultSet rs = pstmt.executeQuery();
        while (rs.next())
        {
            ...
        }
        rs.close();
    }
    catch (SQLException e)
    {
        if (e.getErrorCode() == 21047)
        {
            retry = true;
        }
        else
        {
            throw e;
        }
    }
} while (retry == true);
```

- When committing or rolling back a transaction

If an error occurs on the selected node when committing a transaction, then JDBC checks whether the tra nsaction has been committed before the error occurred through another node. Though an error occurred on the selected node, if the transaction was normally committed, then it does not transfer an error. Also, if an error occurred on the selected node when a transaction has not been committed, then it transfers 2 1047(Retry the transactional operations again) error. If 21047error occurs, then a user should perform th e transaction again.

If an error occurs on the selected node when rolling back a transaction, then JDBC does not transfer an er ror. It is because the transaction already has been rolled back due to the node error.

```
con.setAutoCommit(false);
PreparedStatement pstmt = con.prepareStatement("INSERT INTO T1 VALUES (?)");
boolean retry;
do
{
    retry = false;
    try
    {
        pstmt.setInt(1, 1);
        pstmt.executeUpdate();
        con.commit();
    }
    catch (SQLException e)
    {
        if (e.getErrorCode() == 21047)
        {
            retry = true;
        }
        else
        {
            throw e;
        }
    }
} while (retry == true);
```

Constraints

- Use only PreparedStatement and CallableStatement class to select a node appropriate for SQL.

The statement class does not have the information required to select the node appropriate for the query, so the node is selected according to locality_group_policy or locality_member_policy property.

- Use Connection.commit(). Connection.rollback() to commit or rollback a transaction.

If committing or rolling back with SQL statement when using GLOBAL CONNECTION, then the status cha nge of the transaction is not detected. It is mandatory to use Connection.commit(). Connection.rollback() to commit or rollback the transaction.

- Global session feature does not support Data Definition Language (DDL) among SQL statements.

Statement Pooling

GOLDILOCKS supports the statement pooling feature. Statement pooling improves the performance by p ooling statements which are repeatedly used such as a repetitive statement and a repeatedly called meth od. The interface for the statement pooling is defined in JDBC 3.0.

An application pools the statement related to a specific connection by using the statement pool. Each co nnection object has its own pool. GoldilocksConnection includes a method which enables the statement pool. If the statement pool is used, then the statement object is pooled when the close method is called.

Description

If the statement pool is enabled and the close method of the statement object is called, then GOLDILOCK S JDBC driver automatically pools the statement, PreparedStatement and CallableStatement. PreparedSta tement and CallableStatement objects perform pooling by using SQL string as a key value. JDBC driver au tomatically compares/ retrieves PreparedStatement or CallableStatement when creating them.

It is compared based on the following basis.

- SQL strings should be same.
- The statement types should be same.
- The result set properties should be same.

SQL string of the statement object is subject to change, so SQL string is not used for the pooling, but JDB C driver automatically processes it.

If the matching statement is retrieved while searching in the pool, then it is returned, and if it is not foun d, then a new statement is created. The statement, the cursor and the status are pooled in both cases if t he close method of the object is called. However, the statement becomes closed. Create the statement w hose SQL string, the statement type and the result set property are same to use the closed statement agai n.

If pooled PreparedStatement and CallableStatement objects are retrieved, then the status and the data in formation are automatically initialized again, and reset to the default value. When the statement pool is f ull, then it is dropped from the pool by LRU algorithm. Statements stored in the statement pool can be cl eared when the close method of the connection object is called.

Usage

Enabling Statement Pool

STATEMENT_POOL_ON and STATEMENT_POOL_SIZE properties should be set to use the statement pool of the connection object. Even when STATEMENT_POOL_ON property is enabled, the default value of ST ATEMENT_POOL_SIZE is 0, so use the figure bigger than 0.

The statement pool feature is enabled by adding STATEMENT_POOL_ON and STATEMENT_POOL_SIZE pr operties to the properties object. The statement pool feature is also enabled through GoldilocksDataSour ce API and GoldilocksConnection API.

Enabling through GoldilocksDataSource

If GoldilocksDataSource class is used, then all connection objects gets statement pools with the identical STATEMENT_POOL_SIZE.

- Call setStatementPoolOn(true) method.
- Call setStatementPoolsize method.

```
GoldilocksDataSource sDataSource = new GoldilocksDataSource();
sDataSource.setStatementPoolOn( true );
sDataSource.setStatementPoolSize( 10 );
```

The property setting values can be viewed as follows.

```
System.out.println("Statement Pool on:" + sDataSource.getStatementPoolOn());
System.out.println("Statement Pool size:" + sDataSource.getStatementPoolSize());
```

Enabling through GoldilocksConnection

If GoldilocksConnection class is used, then the statement pool is enabled aside from other objects.

- Call setStatementPoolOn(true) method.
- Call setStatementpoolSize method.

```
GoldilocksConnection sCon = sDataSource.getConnection();
sCon.setStatementPoolOn( true );
sCon.setStatementPoolSize( 10 );
```

The property setting values can be viewed as follows.

```
System.out.println("Statement Pool on:" + sCon.getStatementPoolOn());
System.out.println("Statement Pool size:" + sCon.getStatementPoolSize());
```

Disabling Statement Pool

Enabled statement pool can be disabled. If switching the enabled status to the disabled status, then state ments stored in the statement pool are dropped and closed.

Disabling by using setStatementPoolOn method

```
sCon.setStatementPoolOn(false);
```

Disabling by using setStatementPoolSize method

```
sCon.setStatementPoolSize(0);
```

Creating Statement

The method to created the statement, PreparedStatement, and CallableStatement while the statement p ool is enabled is as same as the general creating method.

The following is a code to create the new statement object.

```
PreparedStatement sPstmt = sCon.prepareStatement( "INSERT INTO EMP VALUES ( ?, ? )" );
```

Disabling Specific Statement

If the statement pool is enabled, then GOLDILOCKS JDBC driver automatically pools all statements. Using setPoolable method excludes a specific statement from the pooling.

The following is an example of checking whether it is pooled by using isPoolable method and setPoolable method.

```
PreparedStatement sPstmt = sCon.prepareStatement( "SELECT 1 FROM DUAL" );
System.out.println( "Is poolable: " + sPstmt.isPoolable() );
sPstmt.setPoolable( false );
System.out.println( "Is poolable: " + sPstmt.isPoolable() );
```

32.3 JDBC API References

Array

The class is not implemented.

free

```
void free() throws SQLException
```

getArray

```
Object getArray() throws SQLException
```

```
Object getArray(Map<String,Class<?>> map) throws SQLException
```

```
Object getArray(long index, int count) throws SQLException
```

```
Object getArray(long index, int count, Map<String,Class<?>> map) throws SQLException
```

getBaseType

```
int getBaseType() throws SQLException
```

getBaseTypeName

```
String getBaseTypeName() throws SQLException
```

getResultSet

```
ResultSet getResultSet() throws SQLException
```

```
ResultSet getResultSet(Map<String,Class<?>> map) throws SQLException
```

```
ResultSet getResultSet(long index, int count) throws SQLException
```

```
ResultSet getResultSet(long index, int count, Map<String,Class<?>> map) throws SQLException
```

Blob

free

```
void free() throws SQLException
```

- Operation: It frees the sources owned by this object.
- Exception: It does not occur.

getBinaryStream

```
InputStream getBinaryStream() throws SQLException
```

- Operation: It returns the blob object value in InputStream type.
- Exception: If the blob is freed, it throws SQLException.

```
InputStream getBinaryStream(long pos, long length) throws SQLException
```

- Operation: It returns InputStream type which includes the blob object value as big as the length from the byte defined as pos.
- Exception: When the blob is freed, if pos is smaller than 1 or bigger than the byte length of the blob object, or if pos + length is bigger than the byte length of blob object, then it throws SQLException.

getBytes

```
byte[] getBytes(long pos, int length) throws SQLException
```

- Operation: It returns byte array which includes the blob object value as big as the length from the byt e defined as pos.
- Exception: When the blob is freed, if pos is smaller than 1 or if the length is smaller than 0, then it thr ows SQLException.

length

```
long length() throws SQLException
```

- Operation: It returns the byte length of the blob object.
- Exception: If the blob is freed, it throws SQLException.

position

```
long position(byte[] pattern, long start) throws SQLException
```

- Operation: It returns the location of the byte from which the byte array pattern starts defined in the b lob object value. pattern search starts from the location of start. If the pattern is not found in the blo b object then -1 is returned.
- Exception: When the blob is freed, if start is smaller than 1, then it throws SQLException.

```
long position(Blob pattern, long start) throws SQLException
```

- Operation: It returns the location of the byte from which the blob object pattern starts defined in the blob object value. pattern search starts from the location of start. If the pattern is not found in the bl ob object then -1 is returned.
- Exception: When the blob is freed, if start is smaller than 1, then it throws SQLException.

setBinaryStream

```
OutputStream setBinaryStream(long pos) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setBytes

```
int setBytes(long pos, byte[] bytes) throws SQLException
```

- Operation: It records the byte array bytes from the given pos location in the blob object value, and ret urns the length of the recorded bytes. If the value exists in pos location, then the byte array is overwri tten. If it reaches to the end of the blob value while recording byte array, then the length of the blob value is extended so that it can include the additional bytes.
- Exception: When the blob is freed, if pos is smaller than 1, then it throws SQLException.

```
int setBytes(long pos, byte[] bytes, int offset, int len) throws SQLException
```

- Operation: It records the blob object value from pos location as long as the len length from offset loc ation of the given byte array bytes, and returns the length of the recorded bytes. If the value exists in pos location, then the byte array is overwritten. If it reaches to the end of the blob value while recordi ng byte array, then the length of the blob value is extended so that it can include the additional bytes.
- Exception: When the blob is freed, if pos is smaller than 1 or offset is smaller than 0 or offset + len is bigger than bytes length, then it throws SQLException.

truncate

```
void truncate(long len) throws SQLException
```

- Operation: It truncates the length of the blob object into the given len length.
- Exception: When the blob is freed, if len is smaller than 0, then it throws SQLException.

CallableStatement

getArray

```
Array getArray(int parameterIndex) throws SQLException
```

- Operation: It does not support an array type.
- Exception: It always returns SQLFeatureNotSupportedException.

```
Array getArray(String parameterName) throws SQLException
```

- Operation: It does not support an array type.
- Exception: It always returns SQLFeatureNotSupportedException.

getBigDecimal

```
BigDecimal getBigDecimal(int parameterIndex) throws SQLException
```

- Operation: It obtains column data at parameterIndex-th position in BigDecimal type. For more information about whether it supports GOLDILOCKS types, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If the parameterIndex is invalid, database is not accessible, or CallableStatement is already closed, then it throws SQLException.

```
BigDecimal getBigDecimal(int parameterIndex, int scale) throws SQLException
```

- Operation: It is not implemented. (It is a deprecated method.)
- Exception: It always throws SQLFeatureNotSupportedException.

```
BigDecimal getBigDecimal(String parameterName) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

getBlob

```
Blob getBlob(int parameterIndex) throws SQLException
```

- Operation: It obtains column data at parameterIndex-th position in blob type.
- Exception: If the parameterIndex is invalid, database is not accessible, or CallableStatement is already

closed, then it throws SQLException.

```
Blob getBlob(String parameterName) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

getBoolean

```
boolean getBoolean(int parameterIndex) throws SQLException
```

- Operation: It obtains column data at parameterIndex-th position in boolean type. For more informati on about whether it supports GOLDILOCKS types, refer to **Whether supporting getter method for GO LDILOCKS type - 1** .
- Exception: If the parameterIndex is invalid, database is not accessible, or CallableStatement is already closed, then it throws SQLException.

```
boolean getBoolean(String parameterName) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

getByte

```
byte getByte(int parameterIndex) throws SQLException
```

- Operation: It obtains column data at parameterIndex-th position in byte type. For more information a bout whether it supports GOLDILOCKS types, refer to **Whether supporting getter method for GOLDIL OCKS type - 1** .
- Exception: If the parameterIndex is invalid, database is not accessible, or CallableStatement is already closed, then it throws SQLException.

```
byte getByte(String parameterName) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

getBytes

```
byte[] getBytes(int parameterIndex) throws SQLException
```

- Operation: It obtains column data at parameterIndex-th position in byte[] type. For more information about whether it supports GOLDILOCKS types, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If the parameterIndex is invalid, database is not accessible, or CallableStatement is already closed, then it throws SQLException.

```
byte[] getBytes(String parameterName) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

getCharacterStream

```
Reader getCharacterStream(int parameterIndex) throws SQLException
```

- Operation: It obtains column data at parameterIndex-th position in reader type. For more information about whether it supports GOLDILOCKS types, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If the parameterIndex is invalid, database is not accessible, or CallableStatement is already closed, then it throws SQLException.

```
Reader getCharacterStream(String parameterName) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

getClob

```
Clob getClob(int parameterIndex) throws SQLException
```

- Operation: It obtains column data at parameterIndex-th position in clob type.
- Exception: If the parameterIndex is invalid, database is not accessible, or CallableStatement is already closed, then it throws SQLException.

```
Clob getClob(String parameterName) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

getDate

```
Date getDate(int parameterIndex) throws SQLException
```

- Operation: It obtains column data at parameterIndex-th position in date type. For more information a bout whether it supports GOLDILOCKS types, refer to **Whether supporting getter method for GOLDIL OCKS type - 1** . It uses local time zone and locale when creating a date object.
- Exception: If the parameterIndex is invalid, database is not accessible, or CallableStatement is already closed, then it throws SQLException.

```
Date getDate(int parameterIndex, Calendar cal) throws SQLException
```

- Operation: It obtains column data at parameterIndex-th position in date type. For more information a bout whether it supports GOLDILOCKS types, refer to **Whether supporting getter method for GOLDIL OCKS type - 1** . It uses time zone and locale of cal when creating a date object.
- Exception: If the parameterIndex is invalid, database is not accessible, or CallableStatement is already closed, then it throws SQLException.

```
Date getDate(String parameterName) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

```
Date getDate(String parameterName, Calendar cal) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

getDouble

```
double getDouble(int parameterIndex) throws SQLException
```

- Operation: It obtains column data at parameterIndex-th position in double type. For more informatio n about whether it supports GOLDILOCKS types, refer to **Whether supporting getter method for GOL DILOCKS type - 1** .
- Exception: If the parameterIndex is invalid, database is not accessible, or CallableStatement is already closed, then it throws SQLException.

```
double getDouble(String parameterName) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

getFloat

```
float getFloat(int parameterIndex) throws SQLException
```

- Operation: It obtains column data at parameterIndex-th position in float type. For more information a bout whether it supports GOLDILOCKS types, refer to **Whether supporting getter method for GOLDIL OCKS type - 1** .
- Exception: If the parameterIndex is invalid, database is not accessible, or CallableStatement is already closed, then it throws SQLException.

```
float getFloat(String parameterName) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

getInt

```
int getInt(int parameterIndex) throws SQLException
```

- Operation: It obtains column data at parameterIndex-th position in int type. For more information ab out whether it supports GOLDILOCKS types, refer to **Whether supporting getter method for GOLDIL OCKS type - 1** .
- Exception: If the parameterIndex is invalid, database is not accessible, or CallableStatement is already closed, then it throws SQLException.

```
int getInt(String parameterName) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

getLong

```
long getLong(int parameterIndex) throws SQLException
```

- Operation: It obtains column data at parameterIndex-th position in long type. For more information a bout whether it supports GOLDILOCKS types, refer to **Whether supporting getter method for GOLDIL OCKS type - 1** .
- Exception: If the parameterIndex is invalid, database is not accessible, or CallableStatement is already closed, then it throws SQLException.

```
long getLong(String parameterName) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

getNCharacterStream

```
Reader getNCharacterStream(int parameterIndex) throws SQLException
```

- Operation: Currently, it does not support NCHAR-family type.
- Exception: It always throws SQLFeatureNotSupportedException.

```
Reader getNCharacterStream(String parameterName) throws SQLException
```

- Operation: Currently, it does not support NCHAR-family type.
- Exception: It always throws SQLFeatureNotSupportedException.

getNClob

```
NClob getNClob(int parameterIndex) throws SQLException
```

- Operation: Currently, it does not support NClob-family type.
- Exception: It always throws SQLFeatureNotSupportedException.

```
NClob getNClob(String parameterName) throws SQLException
```

- Operation: Currently, it does not support NClob-family type.
- Exception: It always throws SQLFeatureNotSupportedException.

getNString

```
String getNString(int parameterIndex) throws SQLException
```

- Operation: Currently, it does not support NCHAR-family type.
- Exception: It always throws SQLFeatureNotSupportedException.

```
String getNString(String parameterName) throws SQLException
```

- Operation: Currently, it does not support NCHAR-family type.
- Exception: It always throws SQLFeatureNotSupportedException.

getObject

```
Object getObject(int parameterIndex) throws SQLException
```

- Operation: It obtains column data at parameterIndex-th position in Java object type. For more information about whether it supports GOLDILOCKS types, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If the parameterIndex is invalid, database is not accessible, or CallableStatement is already closed, then it throws SQLException.

```
Object getObject(int parameterIndex, Map<String,Class<?>> map) throws SQLException
```

- Operation: It is not supported.
- Exception: It always throws SQLFeatureNotSupportedException.

```
Object getObject(String parameterName) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

```
Object getObject(String parameterName, Map<String,Class<?>> map) throws SQLException
```

- Operation: It is not supported.
- Exception: It always throws SQLFeatureNotSupportedException.

getRef

```
Ref getRef(int parameterIndex) throws SQLException
```

- Operation: It does not support ref type.
- Exception: It always returns SQLFeatureNotSupportedException.

```
Ref getRef(String parameterName) throws SQLException
```

- Operation: It does not support ref type.
- Exception: It always returns SQLFeatureNotSupportedException.

getRowId

```
RowId getRowId(int parameterIndex) throws SQLException
```

- Operation: It obtains column data at parameterIndex-th position in rowid type. For more information

about whether it supports GOLDILOCKS types, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If the parameterIndex is invalid, database is not accessible, or CallableStatement is already closed, then it throws SQLException.

```
RowId getRowId(String parameterName) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

getShort

```
short getShort(int parameterIndex) throws SQLException
```

- Operation: It obtains column data at parameterIndex-th position in short type. For more information about whether it supports GOLDILOCKS types, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If the parameterIndex is invalid, database is not accessible, or CallableStatement is already closed, then it throws SQLException.

```
short getShort(String parameterName) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

getSQLXML

```
SQLXML getSQLXML(int parameterIndex) throws SQLException
```

- Operation: It does not support SQLXML type.
- Exception: It always returns SQLFeatureNotSupportedException.

```
SQLXML getSQLXML(String parameterName) throws SQLException
```

- Operation: It does not support SQLXML type.
- Exception: It always returns SQLFeatureNotSupportedException.

getString

```
String getString(int parameterIndex) throws SQLException
```

- Operation: It obtains column data at parameterIndex-th position in string type. For more information about whether it supports GOLDILOCKS types, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If the parameterIndex is invalid, database is not accessible, or CallableStatement is already closed, then it throws SQLException.

```
String getString(String parameterName) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

getTime

```
Time getTime(int parameterIndex) throws SQLException
```

- Operation: It obtains column data at parameterIndex-th position in time type. For more information about whether it supports GOLDILOCKS types, refer to **Whether supporting getter method for GOLDILOCKS type - 1** . It uses local time zone when creating a time object.
- Exception: If the parameterIndex is invalid, database is not accessible, or CallableStatement is already closed, then it throws SQLException.

```
Time getTime(int parameterIndex, Calendar cal) throws SQLException
```

- Operation: It obtains column data at parameterIndex-th position in time type. For more information about whether it supports GOLDILOCKS types, refer to **Whether supporting getter method for GOLDILOCKS type - 1** . It uses time zone of cal when creating a time object.
- Exception: If the parameterIndex is invalid, database is not accessible, or CallableStatement is already closed, then it throws SQLException.

```
Time getTime(String parameterName) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

```
Time getTime(String parameterName, Calendar cal) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

getTimestamp

```
Timestamp getTimestamp(int parameterIndex) throws SQLException
```

- Operation: It obtains column data at parameterIndex-th position in timestamp type. For more inform ation about whether it supports GOLDILOCKS types, refer to **Whether supporting getter method for GOLDILOCKS type - 1** . It uses local time zone when creating a timestamp object.
- Exception: If the parameterIndex is invalid, database is not accessible, or CallableStatement is already closed, then it throws SQLException.

```
Timestamp getTimestamp(int parameterIndex, Calendar cal) throws SQLException
```

- Operation: It obtains column data at parameterIndex-th position in timestamp type. For more inform ation about whether it supports GOLDILOCKS types, refer to **Whether supporting getter method for GOLDILOCKS type - 1** . It uses time zone of cal when creating a timestamp object.
- Exception: If the parameterIndex is invalid, database is not accessible, or CallableStatement is already closed, then it throws SQLException.

```
Timestamp getTimestamp(String parameterName) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

```
Timestamp getTimestamp(String parameterName, Calendar cal) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

getURL

```
URL getURL(int parameterIndex) throws SQLException
```

- Operation: It does not support URL type.
- Exception: It always returns SQLFeatureNotSupportedException.

```
URL getURL(String parameterName) throws SQLException
```

- Operation: It does not support URL type.
- Exception: It always returns SQLFeatureNotSupportedException.

registerOutParameter

```
void registerOutParameter(int parameterIndex, int sqlType) throws SQLException
```

- Operation: It registers out parameters positioned in parameterIndex as sqlType. All out parameters sh ould be registered before executing the stored procedure. JDBC type of out parameter specified as sq lType determines Java type which is used in a get method to read parameter parameter value. For mo re information about whether it supports GOLDILOCKS types, refer to **Whether supporting getter me thod for GOLDILOCKS type - 1** .
- Exception: If the parameterIndex is invalid, database is not accessible, or CallableStatement is already closed, then it throws SQLFeatureNotSupprtedException.

```
void registerOutParameter(int parameterIndex, int sqlType, int scale) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

```
void registerOutParameter(int parameterIndex, int sqlType, String typeName) throws
SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

```
void registerOutParameter(String parameterName, int sqlType) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

```
void registerOutParameter(String parameterName, int sqlType, int scale) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

```
void registerOutParameter(String parameterName, int sqlType, String typeName) throws
SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setAsciiStream

```
void setAsciiStream(String parameterName, InputStream x) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

```
void setAsciiStream(String parameterName, InputStream x, int length) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

```
void setAsciiStream(String parameterName, InputStream x, long length) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setBigDecimal

```
void setBigDecimal(String parameterName, BigDecimal x) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setBinaryStream

```
void setBinaryStream(String parameterName, InputStream x) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

```
void setBinaryStream(String parameterName, InputStream x, int length) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

```
void setBinaryStream(String parameterName, InputStream x, long length) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setBlob

```
void setBlob(String parameterName, Blob x) throws SQLException
```

- Operation: It does not support blob type.
- Exception: It always returns SQLFeatureNotSupportedException.

```
void setBlob(String parameterName, InputStream inputStream) throws SQLException
```

- Operation: It does not support blob type.
- Exception: It always returns SQLFeatureNotSupportedException.

```
void setBlob(String parameterName, InputStream inputStream, long length) throws SQLException
```

- Operation: It does not support blob type.
- Exception: It always returns SQLFeatureNotSupportedException.

setBoolean

```
void setBoolean(String parameterName, boolean x) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setByte

```
void setByte(String parameterName, byte x) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setBytes

```
void setBytes(String parameterName, byte[] x) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setCharacterStream

```
void setCharacterStream(String parameterName, Reader reader) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

```
void setCharacterStream(String parameterName, Reader reader, int length) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

```
void setCharacterStream(String parameterName, Reader reader, long length) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setClob

```
void setClob(String parameterName, Clob x) throws SQLException
```

- Operation: It does not support clob type.
- Exception: It always returns SQLFeatureNotSupportedException.

```
void setClob(String parameterName, Reader reader) throws SQLException
```

- Operation: It does not support clob type.
- Exception: It always returns SQLFeatureNotSupportedException.

```
void setClob(String parameterName, Reader reader, long length) throws SQLException
```

- Operation: It does not support clob type.
- Exception: It always returns SQLFeatureNotSupportedException.

setDate

```
void setDate(String parameterName, Date x) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

```
void setDate(String parameterName, Date x, Calendar cal) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setDouble

```
void setDouble(String parameterName, double x) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setFloat

```
void setFloat(String parameterName, float x) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setInt

```
void setInt(String parameterName, int x) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setLong

```
void setLong(String parameterName, long x) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setNCharacterStream

```
void setNCharacterStream(String parameterName, Reader value) throws SQLException
```

- Operation: It does not support NChar.
- Exception: It always throws SQLFeatureNotSupportedException.

```
void setNCharacterStream(String parameterName, Reader value, long length) throws SQLException
```

- Operation: It does not support NChar.
- Exception: It always throws SQLFeatureNotSupportedException.

setNClob

```
void setNClob(String parameterName, NClob value) throws SQLException
```

- Operation: It does not support NClob.
- Exception: It always throws SQLFeatureNotSupportedException.

```
void setNClob(String parameterName, Reader reader) throws SQLException
```

- Operation: It does not support NClob.
- Exception: It always throws SQLFeatureNotSupportedException.

```
void setNClob(String parameterName, Reader reader, long length) throws SQLException
```

- Operation: It does not support NClob.
- Exception: It always throws SQLFeatureNotSupportedException.

setNString

```
void setNString(String parameterName, String value) throws SQLException
```

- Operation: It does not support NChar.
- Exception: It always throws SQLFeatureNotSupportedException.

setNull

```
void setNull(String parameterName, int sqlType) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

```
void setNull(String parameterName, int sqlType, String typeName) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setObject

```
void setObject(String parameterName, Object x) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

```
void setObject(String parameterName, Object x, int targetSqlType) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

```
void setObject(String parameterName, Object x, int targetSqlType, int scale) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setRowId

```
void setRowId(String parameterName, RowId x) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setShort

```
void setShort(String parameterName, short x) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setSQLXML

```
void setSQLXML(String parameterName, SQLXML xmlObject) throws SQLException
```

- Operation: It does not support SQLXML type.
- Exception: It always returns SQLFeatureNotSupportedException.

setString

```
void setString(String parameterName, String x) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setTime

```
void setTime(String parameterName, Time x) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

```
void setTime(String parameterName, Time x, Calendar cal) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setTimestamp

```
void setTimestamp(String parameterName, Timestamp x) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

```
void setTimestamp(String parameterName, Timestamp x, Calendar cal) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setURL

```
void setURL(String parameterName, URL val) throws SQLException
```

- Operation: It does not support URL type.
- Exception: It always throws SQLFeatureNotSupportedException.

wasNull

```
boolean wasNull()
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

isWrapperFor

```
boolean isWrapperFor(Class<?> iface)
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

unwrap

```
boolean isWrapperFor(Class<?> iface)
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

Clob

free

```
void free() throws SQLException
```

- Operation: It frees the sources owned by this object.
- Exception: It does not occur.

getAsciiStream

```
InputStream getAsciiStream() throws SQLException
```

- Operation: It returns the clob object value in InputStream type.
- Exception: If the clob is freed, it throws SQLException.

getCharacterStream

```
Reader getCharacterStream() throws SQLException
```

- Operation: It returns the clob object value in reader type.
- Exception: If the clob is freed, it throws SQLException.

```
Reader getCharacterStream(long pos, long length) throws SQLException
```

- Operation: It returns the clob object value in reader type as big as the length from the location define d as pos.
- Exception: When the blob is freed, if pos is smaller than 1 or if length is smaller than 1, or if pos + len is equal to or bigger than the length of clob object, then it throws SQLException.

getSubString

```
String getSubString(long pos, int length) throws SQLException
```

- Operation: It returns the string which includes the clob object value as big as the length from the len gth defined as pos.
- Exception: When the clob is freed, if pos is smaller than 1 or if length is smaller than 0, then it throws SQLException.

length

```
long length() throws SQLException
```

- Operation: It returns the data length of the clob object.
- Exception: If the clob object is freed, it throws SQLException.

position

```
long position(Clob searchstr, long start) throws SQLException
```

- Operation: It returns the location from which the clob object searchstr starts defined in the clob objec t value. searchstr search starts from the location of start. If searchstr is not found in the clob object th en -1 is returned.
- Exception: When the clob is freed, if start is smaller than 1, then it throws SQLException.

```
long position(String searchstr, long start) throws SQLException
```

- Operation: It returns the location from which the string object searchstr starts defined in the clob obj ect value. searchstr search starts from the location of start. If searchstr is not found in the clob object then -1 is returned.
- Exception: When the clob is freed, if start is smaller than 1, then it throws SQLException.

setAsciiStream

```
OutputStream setAsciiStream(long pos) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setCharacterStream

```
Writer setCharacterStream(long pos) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setString

```
int setString(long pos, String str) throws SQLException
```

- Operation: It records str from the given pos location in the clob object value, and returns the recorde d length. If the value exists in pos location, then it is overwritten. If it reaches to the end of the clob v alue while recording the string type, then the length of the clob value is extended so that it can inclu de the additional bytes.
- Exception: When the clob is freed, if pos is smaller than 1, then it throws SQLException.

```
int setString(long pos, String str, int offset, int len) throws SQLException
```

- Operation: It records the clob object value from pos location as long as the len length from offset loc ation of the given string object str, and returns the recorded length. If the value exists in pos location, then the value is overwritten. If it reaches to the end of the clob value while recording the string type, then the length of the clob value is extended so that it can include the additional bytes.
- Exception: When the clob is freed, if pos is smaller than 1 or if it is bigger than offset + len, then it thr ows SQLException.

truncate

```
void truncate(long len) throws SQLException
```

- Operation: It truncates the length of the clob object into the given len length.
- Exception: When the clob is freed, if len is smaller than 0 or if the clob size is smaller than len, then it throws SQLException.

CommonDataSource

getLoginTimeout

```
int getLoginTimeout() throws SQLException
```

- Operation: It returns the login timeout setting value. Login timeout is used as a timeout value when p erforming socket connection to the server. If the value is not set, 0 is returned. 0 means infinite stand by.
- Exception: It does not occur.

getLogWriter

```
PrintWriter getLogWriter() throws SQLException
```

- Operation: It returns the log writer which is set in the DataSource. If it is not set, null is returned. Log writer refers to PrintWriter to write various trace logs. For more information about logging, refer to **L ogging**.
- Exception: It does not occur.

setLoginTimeout

```
void setLoginTimeout(int seconds) throws SQLException
```

- Operation: It sets the login timeout value. Login timeout is used as a timeout value when performing socket connection to the server. 0 means infinite standby.
- Exception: It does not occur.

setLogWriter

```
void setLogWriter(PrintWriter out) throws SQLException
```

- Operation: It sets the log writer in DataSource. If the value is not set, the default value is null. Log wri ter refers to PrintWriter to write various trace logs. If the value is set, trace log, query log, and protoc ol log are written according to the options, and connection object which is created from DataSource and all objects which are created from connection object such as Statement, ResultSet, perform loggi ng. Trace log, query log, protocol log options can be specified in the connection url or property. For more information about logging, refer to **Logging**.
- Exception: It does not occur.

setDataSourceName

```
void setDataSourceName(String aDataSourceName)
```

- Operation: It sets the data source name. It is not the mandatory information required for the connecti on. It is the information charged separately to distinguish objects.
- Exception: It does not occur.

setServerName

```
void setServerName(String aServerName)
```

- Operation: It sets the server name, which is the connection URL. It is the mandatory information requi red for the connection.
- Exception: It does not occur.

setDatabaseName

```
void setDatabaseName(String aDBName)
```

- Operation: It sets the database name. It is the mandatory information required for the connection.
- Exception: It does not occur.

setNetworkProtocol

```
void setNetworkProtocol(String aProtocol)
```

- Operation: It is the network protocol information. It is not the mandatory information required for th e connection.
- Exception: It does not occur.

setUser

```
void setUser(String aUser)
```

- Operation: It sets the connection account name. It is the mandatory information required for the con nection.
- Exception: It does not occur.

setPassword

```
void setPassword(String aPassword)
```

- Operation: It sets the connection account password. It is the mandatory information required for the connection.
- Exception: It does not occur.

setPortNumber

```
void setPortNumber(int aPort)
```

- Operation: It sets the port number used when connecting to the server. It is the mandatory informati on required for the connection.
- Exception: It does not occur.

```
void setPortNumber(String aPort)
```

- Operation: It sets the port number used when connecting to the server. It is the mandatory informati on required for the connection.
- Exception: It does not occur.

setRoleName

```
void setRoleName(String aRoleName)
```

- Operation: It specifies the role of when connecting to server. It is not mandatory information required for the connection, but it is the connection related information. One of "", "ADMIN", "SYSDBA" is spe cified.
- Exception: It does not occur.

setDescription

```
void setDescription(String aDescription)
```

- Operation: It sets the description about the data source. It is not used for the connection.
- Exception: It does not occur.

setConnectionProperties

```
void setConnectionProperties(Properties aProps)
```

- Operation: It defines the properties which can be used in various connections.
- Exception: It does not occur.

setURL

```
void setURL(String aURL) throws SQLException
```

- Operation: It specifies the serverName, portNumber, and database Name in URL form. It has URL as same as the URL of when connecting via DriverManager.
- Exception: If it is the wrong format, it throws SQLException.

```
void setUrl(String aUrl) throws SQLException
```

It is as same as setURL (String aURL).

setLogTarget

```
void setLogTarget(String aTarget)
```

- Operation: If setLogWriter can not be called, the log writer is set by this method. Currently, it is supported only when aTarget is "console", other values are ignored. If it is set to "console", all loggings below the connection object which is generated from the DataSource are output to the console.
- Exception: It does not occur.

setTraceLog

```
void setTraceLog(String aMode)
```

- Operation: It sets the trace log. If aMode is *on*, the trace logging is on.
- Exception: It does not occur.

setQueryLog

```
void setQueryLog(String aMode)
```

- Operation: It sets the query log. If aMode is *on*, the query logging is on.

- Exception: It does not occur.

setProtocolLog

```
void setProtocolLog(String aMode)
```

- Operation: It sets the protocol log. If aMode is *on*, the protocol logging is on.
- Exception: It does not occur.

Connection

abort

```
void abort(Executor executor) throws SQLException
```

- Operation: It terminates the connection object. The connection of the connection object is physically closed and the status becomes closed. It returns the resource used by the connection object. Either al l threads accessing to the current connection is normally terminated or it throws SQLException.
- Exception: If the connection is closed or the executor is null, then it throws SQLException. If the secur ity manager exists and checkPermission method rejects the abort call, then it throws SecurityExceptio n.
- Since: 1.7

clearWarnings

```
void clearWarnings() throws SQLException
```

- Operation: It clears the warning object(s) owned by the current connection object.
- Exception: It does not occur.

close

```
void close() throws SQLException
```

- Operation: It breaks the connection with GOLDILOCKS not to use anymore the current connection an d closes all statement objects created from the object. If already closed, any operation is not perform ed.
- Exception: It may happen when an error occurs from the server or it does not respond.

commit

```
void commit() throws SQLException
```

- Operation: For non-auto commit mode, the commit is executed for the current connection.
- Exception: If it is already closed or is on the auto-commit mode, it throws SQLException.

createArrayOf

```
Array createArrayOf(String typeName, Object[] elements) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

createBlob

```
Blob createBlob() throws SQLException
```

- Operation: It creates the object which implemented the blob interface. The first returned object does not have a data. The data can be added to the blob object by using setBytes method of the blob inter face.
- Exception: If it is already closed, it throws SQLException.

createClob

```
Clob createClob() throws SQLException
```

- Operation: It creates the object which implemented the clob interface. The first returned object does not have a data. The data can be added to the clob object by using setString method of the clob inter face.
- Exception: If it is already closed, it throws SQLException.

createNClob

```
NClob createNClob() throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

createSQLXML

```
SQLXML createSQLXML() throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

createStatement

```
Statement createStatement() throws SQLException
```

- Operation: It creates statement object. The ResultSet type created by the statement is ResultSet.TYPE_FORWARD_ONLY, and concurrency is ResultSet.CONCUR_READ_ONLY, and holdability is ResultSet.HOLD_CURSORS_OVER_COMMIT.
- Exception: If it is already closed, it throws SQLException.

```
Statement createStatement(int resultSetType, int resultSetConcurrency) throws SQLException
```

- Operation: It creates statement object which creates ResultSet including the user defined resultset type and resultset concurrency. The holdability of ResultSet which are generated from the statement object is as same as the holdability of the connection type. The default holdability of connection is ResultSet.HOLD_CURSORS_OVER_COMMIT.
- Exception: If it is already closed, it throws SQLException.

```
Statement createStatement(int resultSetType, int resultSetConcurrency, int
resultSetHoldability) throws SQLException
```

- Operation: It creates a statement object which creates ResultSet including the user defined resultset type, resultset concurrency and holdability.
- Exception: If it is already closed, it throws SQLException.

createStruct

```
Struct createStruct(String typeName, Object[] attributes) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

getAutoCommit

```
boolean getAutoCommit() throws SQLException
```

- Operation: It returns the current auto commit mode. If setAutoCommit() has not been called, it returns true.
- Exception: It does not occur.

getCatalog

```
String getCatalog() throws SQLException
```

- Operation: It gets the current catalog name of the database.
- Exception: If an error occurs from the server or it does not respond, then it throws SQLException.

getClientInfo

```
Properties getClientInfo() throws SQLException
```

- Operation: It returns client information set by the user. If setting information does not exist, It returns null.
- Exception: It does not occur.

```
String getClientInfo(String name) throws SQLException
```

- Operation: It returns specific client information set by the user. If the corresponding information does not exist, it returns null.
- Exception: It does not occur.

getHoldability

```
int getHoldability() throws SQLException
```

- Operation: It returns the default holdability value of statements created by this object. The default val ue is ResultSet.HOLD_CURSORS_OVER_COMMIT.
- Exception: It does not occur.

getMetaData

```
DatabaseMetaData getMetaData() throws SQLException
```

- Operation: It gets a DatabaseMetaData object which can be queried for metadata information from t his object. It always returns the same object.
- Exception: If it is already closed, it throws SQLException.

getNetworkTimeout

```
int getNetworkTimeout() throws SQLException
```

- Operation: It returns the current network timeout (millisecond). 0 means that it is limitless.
- Exception: If it is already closed, it throws SQLException.
- Since: 1.7

getTransactionIsolation

```
int getTransactionIsolation() throws SQLException
```

- Operation: It gets the transaction isolation level which is set on the current session (connection). The default value which is set on the server is Connection.TRANSACTION_READ_COMMITTED.
- Exception: If an error occurs from the server, it throws SQLException.

getTypeMap

```
Map<String,Class<?>> getTypeMap() throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

getWarnings

```
SQLWarning getWarnings() throws SQLException
```

- Operation: It returns a warning which the server responds to connection object until now. If a warnin g does not exist or clearWarnings() is already performed, it returns null.
- Exception: It does not occur.

isClosed

```
boolean isClosed() throws SQLException
```

- Operation: It queries whether close() is successfully called. If close() is successfully performed, it retur ns true. Otherwise, it returns false.
- Exception: It does not occur.

This method does not inform a user whether the connection to a physical server is broken. Even if the actual connection is broken it returns true when close () has never been called.

isReadOnly

```
boolean isReadOnly() throws SQLException
```

- Operation: If the session(connection) is read only mode, it returns true. Otherwise, it returns false. The default value is false. Communication with the server occurs.
- Exception: If an error occurs from the server or it does not respond, it throws SQLException.

isValid

```
boolean isValid(int timeout) throws SQLException
```

- Operation: If isClosed() returns true or the heart beat query is sent to the server and response is not successfully received, it returns false. Otherwise, it returns true.
- Exception: It does not occur.

nativeSQL

```
String nativeSQL(String sql) throws SQLException
```

- Operation: It returns the native SQL recognized by the server for the given user SQL statement. GOLDILOCKS always return the value as same as given by the user because the server recognizes the user's SQL statement literally.
- Exception: It does not occur.

prepareCall

```
CallableStatement prepareCall(String sql) throws SQLException
```

- Operation: It returns CallableStatement object for calling stored procedures. ResultSet type created from this CallableStatement is ResultSet.TYPE_FORWARD_ONLY, the concurrency is ResultSet.CONCUR_READ_ONLY, and the holdability is ResultSet.HOLD_CURSORS_OVER_COMMIT.
- Exception: If it is already closed or the SQL statement is wrong, then it throws SQLException.

```
CallableStatement prepareCall(String sql, int resultSetType, int resultSetConcurrency) throws SQLException
```

- Operation: It creates CallableStatement object which creates ResultSet with the resultset type and the resultset concurrency specified by the user. The holdability of ResultSet created from this CallableStatement is as same as that of the connection. The default holdability value of the connection is ResultSet.HOLD_CURSORS_OVER_COMMIT.
- Exception: If it is already closed or the SQL statement is wrong, then it throws SQLException.

```
CallableStatement prepareCall(String sql, int resultSetType, int resultSetConcurrency, int resultSetHoldability) throws SQLException
```

- Operation: It creates CallableStatement object which creates ResultSet with the resultset type, the resultset concurrency and the holdability specified by the user.
- Exception: If it is already closed or the SQL statement is wrong, then it throws SQLException.

prepareStatement

```
PreparedStatement prepareStatement(String sql) throws SQLException
```

- Operation: It sends the SQL statement to the server and prepares (parsing, validation, optimization), and it returns the PreparedStatement object which controls the prepared statement to the user. The type of ResultSet generated by the PreparedStatement is ResultSet.TYPE_FORWARD_ONLY, and concurrency is ResultSet. CONCUR_READ_ONLY, and holdability is ResultSet.HOLD_CURSORS_OVER_COMMIT.
- Exception: If it is already closed or the SQL statement is wrong, it throws SQLException.

```
PreparedStatement prepareStatement(String sql, int autoGeneratedKeys) throws SQLException
```

- Operation: If autoGeneratedKeys is Statement.NO_GENERATED_KEYS, it operates in the same way as prepareStatement(String sql) method. If autoGeneratedKeys is Statement.RETURN_GENERATED_KEYS and sql is INSERT statement, then it returns PreparedStatement object searching for the auto generated key to the user.
- Exception: If it is already closed, the SQL statement is wrong or autoGeneratedKeys value is neither Statement.NO_GENERATED_KEYS nor Statement.RETURN_GENERATED_KEYS, then it throws SQLException.

If the auto generated key does not exist in the table, then the ResultSet is closed.

```
PreparedStatement prepareStatement(String sql, int[] columnIndexes) throws SQLException
```

- Operation: If columnIndexes is null, it operates in the same way as the prepareStatement(String sql)

method. If sql is INSERT statement and columnIndexes is not null, then it returns PreparedStatement object searching for the designated auto generated key with the given index array to the user.
- Exception: If it is already closed or the SQL statement is wrong, it throws SQLException. If an invalid i ndex exists in columnIndexes, then it throws SQLException.

```
PreparedStatement prepareStatement(String sql, int resultSetType, int resultSetConcurrency)
throws SQLException
```

- Operation: It sends the SQL statement to the server and prepares (parsing, validation, optimization). I t returns the PreparedStatement object which controls the prepared statement to the user. It creates the PreparedStatement object generating ResultSet which includes the user defined result set type an d result set concurrency. The holdability of ResultSet which is generated by PreparedStatement is as s ame as the holdability of connection. The default holdability value of connection is ResultSet.HOLD_C URSORS_OVER_COMMIT.
- Exception: If it is already closed or the SQL statement is wrong, it throws SQLException.

```
PreparedStatement prepareStatement(String sql, int resultSetType, int resultSetConcurrency,
int resultSetHoldability) throws SQLException
```

- Operation: It sends the SQL statement to the server and prepares (parsing, validation, optimization). I t returns the PreparedStatement object which controls the prepared statement to the user. It creates the PreparedStatement object generating ResultSet which includes the user defined result set type an d result set concurrency and holdability.
- Exception: If it is already closed or the SQL statement is wrong, it throws SQLException.

```
PreparedStatement prepareStatement(String sql, String[] columnNames) throws SQLException
```

- Operation: If columnNames is null, it operates in the same way as prepareStatement(String sql) meth od. If sql is INSERT statement and columnNames is not null, then it returns PreparedStatement object searching for the designated auto generated key with the given column name to the user.
- Exception: If it is already closed or the SQL statement is wrong, it throws SQLException. If an invalid c olumn name exists in columnNames, then it throws SQLException.

releaseSavepoint

```
void releaseSavepoint(Savepoint savepoint) throws SQLException
```

- Operation: It removes the savepoint from the server. It also removes all savepoints after this savepoint.
- Exception: If it is already closed or the savepoint object was already released or it is not the GOLDILO CKS savepoint object, it throws SQLException.

rollback

```
void rollback() throws SQLException
```

- Operation: If it is non auto commit mode, it rolls back the current transaction.
- Exception: If it is already closed or it is the auto commit mode or an error is returned from the server, it throws SQLException.

```
void rollback(Savepoint savepoint) throws SQLException
```

- Operation: If it is non auto commit mode, it performs partial rollback to the savepoint for the current transaction.
- Exception: If it is already closed or it is the auto commit mode or an error is returned from the server or the savepoint is not valid, it throws SQLException.

setAutoCommit

```
void setAutoCommit(boolean autoCommit) throws SQLException
```

- Operation: It changes the auto commit mode for the current connection. If the performed transactio n exists and the non auto commit mode is changed to the auto commit, it performs commit.
- Exception: It is as same as the exception which may occur during the commit process.

setCatalog

```
void setCatalog(String catalog) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setClientInfo

```
void setClientInfo(Properties properties) throws SQLException
```

- Operation: It sets the user defined client information. Existing client information is eliminated. It does not affect the server.
- Exception: It does not occur.

```
void setClientInfo(String name, String value) throws SQLException
```

- Operation: It adds the user defined client information. Existing client information is retained.

- Exception: It does not occur.

setHoldability

```
void setHoldability(int holdability) throws SQLException
```

- Operation: It sets the ResultSet holdability which is generated by the statement generated from this o bject. If the method is not called, the default value is ResultSet.HOLD_CURSORS_OVER_COMMIT.
- Exception: It does not occur.

setNetworkTimeout

```
void setNetworkTimeout(Executor executor, int milliseconds) throws SQLException
```

- Operation: It sets the maximum waiting time until the database responds to the request by the conne ction or the object created from the connection. If it remains without receiving the response, then it i s returned together with SQLException and the connection object is closed.
- Exception: If it is already closed, the executor is null, or the millisecond is smaller than 0, then it thro ws SQLException. If the security manager exists and checkPermission method rejects the setNetwork Timeout call, then it throws SecurityException.
- Since: 1.7

setReadOnly

```
void setReadOnly(boolean readOnly) throws SQLException
```

- Operation: It sets the read only property of the current session (connection). The default value is false.
- Exception: If it is already closed or an error is returned from the server, it throws SQLException.

setSavepoint

```
Savepoint setSavepoint() throws SQLException
```

- Operation: If it is the non auto commit mode, it sets the savepoint for the current transaction. The sa vepoint name is internally determined.
- Exception: If it is already closed or it is the auto commit mode or an error is returned from the server, it throws SQLException.

```
Savepoint setSavepoint(String name) throws SQLException
```

- Operation: If it is the non auto commit mode, it sets the savepoint whose name is specfied by the use

r for the current transaction.

- Exception: If it is already closed or it is the auto commit mode or an error is returned from the server, it throws SQLException.

setTransactionIsolation

```
void setTransactionIsolation(int level) throws SQLException
```

- Operation: It changes the transaction isolation for the current session (connection). The supported va lue is Connection.TRANSACTION_READ_COMMITED, Connection.TRANSACTION_READ_UNCOMMIT TED, and Connection.TRANSACTION_SERIALIZABLE. Connection.READ_UNCOMMITTED is set to Con nection.TRANSACTION_READ_COMMITED, and Connection.TRANSACTION_REPEATABLE_READ is s et to Connection.TRANSACTION_SERIALIZABLE.
- Exception: If it is already closed or the level has the wrong value, it throws SQLException.

setTypeMap

```
void setTypeMap(Map<String,Class<?>> map) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

isWrapperFor

```
void isWrapperFor(Class<?> iface) throws SQLException
```

- Operation: It queries whether this object is a class which implements the iface interface. If it is, it retu rns true. Otherwise, it returns false. It does not determine the presence of the wrapper but it only qu eries only whether the given argument class type is implemented because GOLDILOCKS connection o bject is not the wrapper of any other class.
- Exception: It does not occur.

unwrap

```
<T> T unwrap(Class<T> iface) throws SQLException
```

- Operation: It eventually returns itself, even when it is unwrapped because GOLDILOCKS connection is not the wrapper of any other class. It returns itself after casting it to the corresponding type. If iface is an argument of isWrapperFor() method and false is returned, the method throws an exception.
- Exception: If iface is not the type of this object (if this object returns the unimplemented type), it thro ws SQLException.

ConnectionPoolDataSource

getPooledConnection

```
PooledConnection getPooledConnection() throws SQLException
```

- Operation: It creates the PooledConnection object. Various connection information, such as user na me, password, connection URL should be set through the setter method in advance.
- Exception: If the connection fails, it throws SQLException.

```
PooledConnection getPooledConnection(String user, String password) throws SQLException
```

- Operation: It creates the PooledConnection object. Username and password shall comply with the ar gument. Other connection information should be set to the setter method in advance.
- Exception: If the connection fails, it throws SQLException.

DatabaseMetaData

allProceduresAreCallable

```
boolean allProceduresAreCallable() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

allTablesAreSelectable

```
boolean allTablesAreSelectable() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

autoCommitFailureClosesAllResultSets

```
boolean autoCommitFailureClosesAllResultSets() throws SQLException
```

- Operation: It always returns false.
- Exception: It does not occur.

dataDefinitionCausesTransactionCommit

```
boolean dataDefinitionCausesTransactionCommit() throws SQLException
```

- Operation: It always returns false because it does not automatically commit DDL statements at run-time.
- Exception: It does not occur.

dataDefinitionIgnoredInTransactions

```
boolean dataDefinitionIgnoredInTransactions() throws SQLException
```

- Operation: It always returns false, because DDL statements are included in a transaction.
- Exception: It does not occur.

deletesAreDetected

```
boolean deletesAreDetected(int type) throws SQLException
```

- Operation: If type is ResultSet.TYPE_SCROLL_SENSITIVE, it returns true. Otherwise, it returns false.
- Exception: It does not occur.

doesMaxRowSizeIncludeBlobs

```
boolean doesMaxRowSizeIncludeBlobs() throws SQLException
```

- Operation: It always returns false.
- Exception: It does not occur.

getAttributes

```
ResultSet getAttributes(String catalog, String schemaPattern, String typeNamePattern, String attributeNamePattern) throws SQLException
```

- Operation: It returns an empty ResultSet.
- Exception: If an error is returned from the server, it throws SQLException.

getBestRowIdentifier

```
ResultSet getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable) throws SQLException
```

- Operation: It returns a ResultSet including one row consisting with rowid information because the ro wid type is supported for all tables. If the table does not exist, it returns an empty ResultSet.
- Exception: If the table is null or an error is returned from the server, it throws SQLException.

getCatalogs

```
ResultSet getCatalogs() throws SQLException
```

- Operation: It returns a ResultSet which has the catalog name in a column.
- Exception: If an error is returned from the server, it throws SQLException.

getCatalogSeparator

```
String getCatalogSeparator() throws SQLException
```

- Operation: It returns a catalog separator character.
- Exception: If an error is returned from the server, it throws SQLException.

getCatalogTerm

```
String getCatalogTerm() throws SQLException
```

- Operation: It returns a catalog term character.
- Exception: If an error is returned from the server, it throws SQLException.

getClientInfoProperties

```
ResultSet getClientInfoProperties() throws SQLException
```

- Operation: It returns a ResultSet including client information.
- Exception: If an error is returned from the server, it throws SQLException.

getColumnPrivileges

```
ResultSet getColumnPrivileges(String catalog, String schema, String table, String
columnNamePattern) throws SQLException
```

- Operation: It returns a ResultSet including column privilege information of the column in the table. If the table or the column corresponding to the column name pattern does not exist, it returns an empt y ResultSet.
- Exception: If an error is returned from the server, it throws SQLException.

getColumns

```
ResultSet getColumns(String catalog, String schemaPattern, String tableNamePattern, String
columnNamePattern) throws SQLException
```

- Operation: It returns a ResultSet including all column information of the table. If the table or the colu mn corresponding to the column name pattern does not exist, it returns an empty ResultSet.
- Exception: If an error is returned from the server, it throws SQLException.

getConnection

```
Connection getConnection() throws SQLException
```

- Operation: It returns the connection object which created the DatabaseMetaData object.
- Exception: It does not occur.

getCrossReference

```
ResultSet getCrossReference(String parentCatalog, String parentSchema, String parentTable,
String foreignCatalog, String foreignSchema, String foreignTable) throws SQLException
```

- Operation: It returns a ResultSet including information of the foreign keys which refers to the given p arent table in the given foreign key table. If reference relationship does not exist, it returns an empty ResultSet.
- Exception: If the foreign table or parent table is null or an error is returned from the server, it throws SQLException.

getDatabaseMajorVersion

```
int getDatabaseMajorVersion() throws SQLException
```

- Operation: It returns the major version of the product.
- Exception: If an error is returned from the server, it throws SQLException.

getDatabaseMinorVersion

```
int getDatabaseMinorVersion() throws SQLException
```

- Operation: It returns the minor version of the product.
- Exception: If an error is returned from the server, it throws SQLException.

getDatabaseProductName

```
String getDatabaseProductName() throws SQLException
```

- Operation: It returns the product name.
- Exception: If an error is returned from the server, it throws SQLException.

getDatabaseProductVersion

```
String getDatabaseProductVersion() throws SQLException
```

- Operation: It returns the product version.
- Exception: If an error is returned from the server, it throws SQLException.

getDefaultTransactionIsolation

```
int getDefaultTransactionIsolation() throws SQLException
```

- Operation: It returns the default transaction isolation. The default value which is set on the server is C onnection.TRANSACTION_READ_COMMITTED.
- Exception: If an error is returned from the server, it throws SQLException.

getDriverMajorVersion

```
int getDriverMajorVersion() throws SQLException
```

- Operation: It returns the major version of GOLDILOCKS JDBC driver.
- Exception: It does not occur.

getDriverMinorVersion

```
int getDriverMinorVersion() throws SQLException
```

- Operation: It returns the minor version of GOLDILOCKS JDBC driver.
- Exception: It does not occur.

getDriverName

```
String getDriverName() throws SQLException
```

- Operation: It returns "GOLDILOCKS JDBC Driver".
- Exception: It does not occur.

getDriverVersion

```
String getDriverVersion() throws SQLException
```

- Operation: It returns GOLDILOCKS JDBC driver version string. It includes the protocol version.
- Exception: It does not occur.

getExportedKeys

```
ResultSet getExportedKeys(String catalog, String schema, String table) throws SQLException
```

- Operation: It returns a ResultSet including foreign key information referring to the column in the give n table.
- Exception: If an error is returned from the server, it throws SQLException.

getExtraNameCharacters

```
String getExtraNameCharacters() throws SQLException
```

- Operation: It returns "-$".
- Exception: It does not occur.

getFunctionColumns

```
ResultSet getFunctionColumns(String catalog, String schemaPattern, String functionNamePattern,
String columnNamePattern) throws SQLException
```

- Operation: It returns an empty ResultSet.
- Exception: If an error is returned from the server, it throws SQLException.

getFunctions

```
ResultSet getFunctions(String catalog, String schemaPattern, String functionNamePattern)
throws SQLException
```

- Operation: It returns an empty ResultSet.
- Exception: If an error is returned from the server, it throws SQLException.

getIdentifierQuoteString

```
String getIdentifierQuoteString() throws SQLException
```

- Operation: It returns an identifier quote character. The value that is set on the server is ".
- Exception: If an error is returned from the server, it throws SQLException.

getImportedKeys

```
ResultSet getImportedKeys(String catalog, String schema, String table) throws SQLException
```

- Operation: It returns a ResultSet including parent key information to which the foreign key column in the given table refers.
- Exception: If an error is returned from the server, it throws SQLException.

getIndexInfo

```
ResultSet getIndexInfo(String catalog, String schema, String table, boolean unique, boolean approximate) throws SQLException
```

- Operation: It returns a ResultSet including index information of the given table. If unique is true, only unique index information is displayed. The approximate argument is ignored.
- Exception: If an error is returned from the server, it throws SQLException.

getJDBCMajorVersion

```
int getJDBCMajorVersion() throws SQLException
```

- Operation: It returns JDBC major version of GOLDILOCKS JDBC driver. It can vary depending on the ja r file in use.
- Exception: It does not occur.

getJDBCMinorVersion

```
int getJDBCMinorVersion() throws SQLException
```

- Operation: It returns JDBC minor version of GOLDILOCKS JDBC driver. It can vary depending on the ja r file in use.
- Exception: It does not occur.

getMaxBinaryLiteralLength

```
int getMaxBinaryLiteralLength() throws SQLException
```

- Operation: It gets the maximum binary length from the server. 0 refers that the maximum length is in finite.
- Exception: If an error is returned from the server, it throws SQLException.

getMaxCatalogNameLength

```
int getMaxCatalogNameLength() throws SQLException
```

- Operation: It gets the maximum length of the catalog name from the server. 0 refers that the maxim um length is infinite.
- Exception: If an error is returned from the server, it throws SQLException.

getMaxCharLiteralLength

```
int getMaxCharLiteralLength() throws SQLException
```

- Operation: It gets the maximum literal length from the server. 0 refers that the maximum length is inf inite.
- Exception: If an error is returned from the server, it throws SQLException.

getMaxColumnNameLength

```
int getMaxColumnNameLength() throws SQLException
```

- Operation: It gets the maximum length of the column name from the server. 0 refers that the maxim um length is infinite.
- Exception: If an error is returned from the server, it throws SQLException.

getMaxColumnsInGroupBy

```
int getMaxColumnsInGroupBy() throws SQLException
```

- Operation: It gets the maximum number of columns available to use in *group by* clause from the serv er. 0 refers that the maximum number is infinite.
- Exception: If an error is returned from the server, it throws SQLException.

getMaxColumnsInIndex

```
int getMaxColumnsInIndex() throws SQLException
```

- Operation: It gets the maximum number of columns available to use as the index from the server. 0 r efers that the maximum number is infinite.
- Exception: If an error is returned from the server, it throws SQLException.

getMaxColumnsInOrderBy

```
int getMaxColumnsInOrderBy() throws SQLException
```

- Operation: It gets the maximum number of columns available to use in *order by* clause from the serve r. 0 refers that the maximum number is infinite.
- Exception: If an error is returned from the server, it throws SQLException.

getMaxColumnsInSelect

```
int getMaxColumnsInSelect() throws SQLException
```

- Operation: It gets the maximum number of columns available to use in select target clause from the s erver. 0 refers that the maximum number is infinite.
- Exception: If an error is returned from the server, it throws SQLException.

getMaxColumnsInTable

```
int getMaxColumnsInTable() throws SQLException
```

- Operation: It gets the maximum number of columns available to use in the table from the server. 0 re fers that the maximum number is infinite.
- Exception: If an error is returned from the server, it throws SQLException.

getMaxConnections

```
int getMaxConnections() throws SQLException
```

- Operation: It gets the maximum number of connection to the server from the server. 0 refers that the maximum number is infinite.
- Exception: If an error is returned from the server, it throws SQLException.

getMaxCursorNameLength

```
int getMaxCursorNameLength() throws SQLException
```

- Operation: It gets the maximum length of the cursor name from the server. 0 refers that the maximum length is infinite.
- Exception: If an error is returned from the server, it throws SQLException.

getMaxIndexLength

```
int getMaxIndexLength() throws SQLException
```

- Operation: It gets the maximum size for a single index key from the server.
- Exception: If an error is returned from the server, it throws SQLException.

The JDBC specification defines that the available maximum size of a single index is returned in bytes, but its value is meaningless, because index size does not have limit. It operates in this way for it to have the same meaning as ODBC.

getMaxProcedureNameLength

```
int getMaxProcedureNameLength() throws SQLException
```

- Operation: It gets the maximum length of the procedure name from the server. 0 refers that the maximum length is infinite.
- Exception: If an error is returned from the server, it throws SQLException.

getMaxRowSize

```
int getMaxRowSize() throws SQLException
```

- Operation: It gets the maximum number of rows in a table from the server. 0 refers that the maximum number is infinite.
- Exception: If an error is returned from the server, it throws SQLException.

getMaxSchemaNameLength

```
int getMaxSchemaNameLength() throws SQLException
```

- Operation: It gets the maximum length of the schema name from the server. 0 refers that the maxim um length is infinite.
- Exception: If an error is returned from the server, it throws SQLException.

getMaxStatementLength

```
int getMaxStatementLength() throws SQLException
```

- Operation: It gets the maximum string length of an SQL statement from the server. 0 refers that the maximum length is infinite.
- Exception: If an error is returned from the server, it throws SQLException.

getMaxStatements

```
int getMaxStatements() throws SQLException
```

- Operation: It gets the maximum number of statements which can be open at once from the server. 0 refers that the maximum number is infinite.
- Exception: If an error is returned from the server, it throws SQLException.

getMaxTableNameLength

```
int getMaxTableNameLength() throws SQLException
```

- Operation: It gets the maximum string length of the table name from the server. 0 refers that the ma ximum length is infinite.
- Exception: If an error is returned from the server, it throws SQLException.

getMaxTablesInSelect

```
int getMaxTablesInSelect() throws SQLException
```

- Operation: It gets the maximum number of tables which can be used in the select statement from th e server. 0 refers that the maximum number is infinite.
- Exception: If an error is returned from the server, it throws SQLException.

getMaxUserNameLength

```
int getMaxUserNameLength() throws SQLException
```

- Operation: It gets the maximum string length of the user name from the server. 0 refers that the max imum length is infinite.
- Exception: If an error is returned from the server, it throws SQLException.

getNumericFunctions

```
String getNumericFunctions() throws SQLException
```

- Operation: It gets a list of numeric-related functions corresponding to SQL standard from the server. Each function name is distinguished by comma (,).
- Exception: If an error is returned from the server, it throws SQLException.

getPrimaryKeys

```
ResultSet getPrimaryKeys(String catalog, String schema, String table) throws SQLException
```

- Operation: It returns a ResultSet including all primary keys in a given table.
- Exception: If an error is returned from the server, it throws SQLException.

getProcedureColumns

```
ResultSet getProcedureColumns(String catalog, String schemaPattern, String
procedureNamePattern, String columnNamePattern) throws SQLException
```

- Operation: It returns a ResultSet including column information corresponding to the given name patt ern for a given procedure.
- Exception: If an error is returned from the server, it throws SQLException.

getProcedures

```
ResultSet getProcedures(String catalog, String schemaPattern, String procedureNamePattern)
throws SQLException
```

- Operation: It returns a ResultSet including procedure information of a given name pattern.
- Exception: If an error is returned from the server, it throws SQLException.

getProcedureTerm

```
String getProcedureTerm() throws SQLException
```

- Operation: It gets the keyword which refers to the procedure from the server.
- Exception: If an error is returned from the server, it throws SQLException.

getResultSetHoldability

```
int getResultSetHoldability() throws SQLException
```

- Operation: It returns the default holdability property of ResultSet. It is ResultSet.HOLD_CURSORS_OVER_COMMIT.
- Exception: It does not occur.

getRowIdLifetime

```
RowIdLifetime getRowIdLifetime() throws SQLException
```

- Operation: It always returns RowIdLifetime.ROWID_VALID_FOREVER.
- Exception: It does not occur.

getSchemas

```
ResultSet getSchemas() throws SQLException
```

- Operation: It returns a ResultSet including information about all schemas in the server.
- Exception: If an error is returned from the server, it throws SQLException.

```
ResultSet getSchemas(String catalog, String schemaPattern) throws SQLException
```

- Operation: It returns a ResultSet including information for all schemas which meet the pattern of a given schema name. The catalog is ignored.
- Exception: If an error is returned from the server, it throws SQLException.

getSchemaTerm

```
String getSchemaTerm() throws SQLException
```

- Operation: It gets the keyword which refers to the schema from the server.

- Exception: If an error is returned from the server, it throws SQLException.

getSearchStringEscape

```
String getSearchStringEscape() throws SQLException
```

- Operation: It returns the escape characters used in *like* clause as a string.
- Exception: If an error is returned from the server, it throws SQLException.

getSQLKeywords

```
String getSQLKeywords() throws SQLException
```

- Operation: It returns the keywords which can not be used in SQL statement, separated by comma (,).
- Exception: If an error is returned from the server, it throws SQLException.

getSQLStateType

```
int getSQLStateType() throws SQLException
```

- Operation: It always returns DatabaseMetaData.sqlStateSQL99.
- Exception: It does not occur.

getStringFunctions

```
String getStringFunctions() throws SQLException
```

- Operation: It gets a list of functions which handle the strings and corresponds to SQL standard. Each function name is distinguished by comma (,).
- Exception: If an error is returned from the server, it throws SQLException.

getSuperTables

```
ResultSet getSuperTables(String catalog, String schemaPattern, String tableNamePattern) throws SQLException
```

- Operation: It returns an empty ResultSet.
- Exception: If an error is returned from the server, it throws SQLException.

getSuperTypes

```
ResultSet getSuperTypes(String catalog, String schemaPattern, String typeNamePattern) throws SQLException
```

- Operation: It returns an empty ResultSet.
- Exception: If an error is returned from the server, it throws SQLException.

getSystemFunctions

```
String getSystemFunctions() throws SQLException
```

- Operation: It gets a list of system functions corresponding to SQL standard. Each function name is separated by the comma (,).
- Exception: If an error is returned from the server, it throws SQLException.

getTablePrivileges

```
ResultSet getTablePrivileges(String catalog, String schemaPattern, String tableNamePattern) throws SQLException
```

- Operation: It returns a ResultSet including privilege information of all tables which satisfy the condition.
- Exception: If an error is returned from the server, it throws SQLException.

getTables

```
ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types) throws SQLException
```

- Operation: It returns a ResultSet including information of all tables which satisfy the condition.
- Exception: If an error is returned from the server, it throws SQLException.

getTableTypes

```
ResultSet getTableTypes() throws SQLException
```

- Operation: It returns a ResultSet containing information of all table types.
- Exception: If an error is returned from the server, it throws SQLException.

getTimeDateFunctions

```
String getTimeDateFunctions() throws SQLException
```

- Operation: It gets functions which are related to time and date and corresponds to SQL standard. Eac h function name is separated by the comma(,).
- Exception: If an error is returned from the server, it throws SQLException.

getTypeInfo

```
ResultSet getTypeInfo() throws SQLException
```

- Operation: It returns a ResultSet including information of the data type.
- Exception: If an error is returned from the server, it throws SQLException.

getUDTs

```
ResultSet getUDTs(String catalog, String schemaPattern, String typeNamePattern, int[] types)
throws SQLException
```

- Operation: It returns an empty ResultSet.
- Exception: If an error is returned from the server, it throws SQLException.

getURL

```
String getURL() throws SQLException
```

- Operation: It returns the URL used to connect to the server.
- Exception: It does not occur.

getUserName

```
String getUserName() throws SQLException
```

- Operation: It returns the current user name which maintains a session.
- Exception: If an error is returned from the server, it throws SQLException.

> ⚠️ A user name may be different from the user name used to connect firstly because the user can be changed during a session.

getVersionColumns

```
ResultSet getVersionColumns(String catalog, String schema, String table) throws SQLException
```

- Operation: It returns an empty ResultSet.
- Exception: If an error is returned from the server, it throws SQLException.

insertsAreDetected

```
boolean insertsAreDetected(int type) throws SQLException
```

- Operation: It always returns false regardless of the type.
- Exception: It does not occur.

isCatalogAtStart

```
boolean isCatalogAtStart() throws SQLException
```

- Operation: It always returns false.
- Exception: It does not occur.

isReadOnly

```
boolean isReadOnly() throws SQLException
```

- Operation: It gets the information whether the current connection is the read only mode from the ser ver.
- Exception: If an error is returned from the server, it throws SQLException.

locatorsUpdateCopy

```
boolean locatorsUpdateCopy() throws SQLException
```

- Operation: It always returns false.
- Exception: It does not occur.

nullPlusNonNullIsNull

```
boolean nullPlusNonNullIsNull() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

nullsAreSortedAtEnd

```
boolean nullsAreSortedAtEnd() throws SQLException
```

- Operation: It always returns false. It does not separately sort null.
- Exception: It does not occur.

nullsAreSortedAtStart

```
boolean nullsAreSortedAtStart() throws SQLException
```

- Operation: It always returns false. It does not separately sort null.
- Exception: It does not occur.

nullsAreSortedHigh

```
boolean nullsAreSortedHigh() throws SQLException
```

- Operation: It always returns true. Null is positioned at last by default.
- Exception: It does not occur.

nullsAreSortedLow

```
boolean nullsAreSortedLow() throws SQLException
```

- Operation: It always returns false. Null is positioned at last by default.
- Exception: It does not occur.

othersDeletesAreVisible

```
boolean othersDeletesAreVisible(int type) throws SQLException
```

- Operation: If the type is ResultSet.TYPE_SCROLL_SENSITIVE, it returns true. Otherwise, it returns false.
- Exception: It does not occur.

othersInsertsAreVisible

```
boolean othersInsertsAreVisible(int type) throws SQLException
```

- Operation: It always returns false regardless of the type.
- Exception: It does not occur.

othersUpdatesAreVisible

```
boolean othersUpdatesAreVisible(int type) throws SQLException
```

- Operation: If the type is ResultSet.TYPE_SCROLL_SENSITIVE, it returns true. Otherwise, it returns false.
- Exception: It does not occur.

ownDeletesAreVisible

```
boolean ownDeletesAreVisible(int type) throws SQLException
```

- Operation: If the type is ResultSet.TYPE_SCROLL_SENSITIVE, it returns true. Otherwise, it returns false.
- Exception: It does not occur.

ownInsertsAreVisible

```
boolean ownInsertsAreVisible(int type) throws SQLException
```

- Operation: It always returns false regardless of the type.
- Exception: It does not occur.

ownUpdatesAreVisible

```
boolean ownUpdatesAreVisible(int type) throws SQLException
```

- Operation: If the type is ResultSet.TYPE_SCROLL_SENSITIVE, it returns true. Otherwise, it returns false.
- Exception: It does not occur.

storesLowerCaseIdentifiers

```
boolean storesLowerCaseIdentifiers() throws SQLException
```

- Operation: It always returns false.
- Exception: It does not occur.

storesLowerCaseQuotedIdentifiers

```
boolean storesLowerCaseQuotedIdentifiers() throws SQLException
```

- Operation: It always returns false.
- Exception: It does not occur.

storesMixedCaseIdentifiers

```
boolean storesMixedCaseIdentifiers() throws SQLException
```

- Operation: It always returns false.
- Exception: It does not occur.

storesMixedCaseQuotedIdentifiers

```
boolean storesMixedCaseQuotedIdentifiers() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

storesUpperCaseIdentifiers

```
boolean storesUpperCaseIdentifiers() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

storesUpperCaseQuotedIdentifiers

```
boolean storesUpperCaseQuotedIdentifiers() throws SQLException
```

- Operation: It always returns false.
- Exception: It does not occur.

supportsAlterTableWithAddColumn

```
boolean supportsAlterTableWithAddColumn() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsAlterTableWithDropColumn

```
boolean supportsAlterTableWithDropColumn() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsANSI92EntryLevelSQL

```
boolean supportsANSI92EntryLevelSQL() throws SQLException
```

- Operation: It always returns false.
- Exception: It does not occur.

supportsANSI92FullSQL

```
boolean supportsANSI92FullSQL() throws SQLException
```

- Operation: It always returns false.
- Exception: It does not occur.

supportsANSI92IntermediateSQL

```
boolean supportsANSI92IntermediateSQL() throws SQLException
```

- Operation: It always returns false.
- Exception: It does not occur.

supportsBatchUpdates

```
boolean supportsBatchUpdates() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsCatalogsInDataManipulation

```
boolean supportsCatalogsInDataManipulation() throws SQLException
```

- Operation: It always returns false.
- Exception: It does not occur.

supportsCatalogsInIndexDefinitions

```
boolean supportsCatalogsInIndexDefinitions() throws SQLException
```

- Operation: It always returns false.
- Exception: It does not occur.

supportsCatalogsInPrivilegeDefinitions

```
boolean supportsCatalogsInPrivilegeDefinitions() throws SQLException
```

- Operation: It always returns false.
- Exception: It does not occur.

supportsCatalogsInProcedureCalls

```
boolean supportsCatalogsInProcedureCalls() throws SQLException
```

- Operation: It always returns false.
- Exception: It does not occur.

supportsCatalogsInTableDefinitions

```
boolean supportsCatalogsInTableDefinitions() throws SQLException
```

- Operation: It always returns false.
- Exception: It does not occur.

supportsColumnAliasing

```
boolean supportsColumnAliasing() throws SQLException
```

- Operation: It gets information whether to support the column aliasing from the server.
- Exception: If an error is returned from the server, it throws SQLException.

supportsConvert

```
boolean supportsConvert() throws SQLException
```

- Operation: It always returns false.
- Exception: It does not occur.

supportsConvert

```
boolean supportsConvert(int fromType, int toType) throws SQLException
```

- Operation: It always returns false.
- Exception: It does not occur.

supportsCoreSQLGrammar

```
boolean supportsCoreSQLGrammar() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsCorrelatedSubqueries

boolean supportsCorrelatedSubqueries() throws SQLException

- Operation: It always returns true.
- Exception: It does not occur.

supportsDataDefinitionAndDataManipulationTransactions

```
boolean supportsDataDefinitionAndDataManipulationTransactions() throws SQLException
```

- Operation: It always returns true. It can perform DML and DDL with a single transaction.
- Exception: It does not occur.

supportsDataManipulationTransactionsOnly

```
boolean supportsDataManipulationTransactionsOnly() throws SQLException
```

- Operation: It always returns false.
- Exception: It does not occur.

supportsDifferentTableCorrelationNames

```
boolean supportsDifferentTableCorrelationNames() throws SQLException
```

- Operation: It gets information whether the table correlation name should be different from the table name from the server.
- Exception: If an error is returned from the server, it throws SQLException.

supportsExpressionsInOrderBy

```
boolean supportsExpressionsInOrderBy() throws SQLException
```

- Operation: It gets information whether the calculation can be used in *order by* clause from the server.
- Exception: If an error is returned from the server, it throws SQLException.

supportsExtendedSQLGrammar

```
boolean supportsExtendedSQLGrammar() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsFullOuterJoins

```
boolean supportsFullOuterJoins() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsGetGeneratedKeys

```
boolean supportsGetGeneratedKeys() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsGroupBy

```
boolean supportsGroupBy() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsGroupByBeyondSelect

```
boolean supportsGroupByBeyondSelect() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsGroupByUnrelated

```
boolean supportsGroupByUnrelated() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsIntegrityEnhancementFacility

```
boolean supportsIntegrityEnhancementFacility() throws SQLException
```

- Operation: It gets information whether to support SQL integrity enhancing feature from the server.
- Exception: If an error is returned from the server, it throws SQLException.

supportsLikeEscapeClause

```
boolean supportsLikeEscapeClause() throws SQLException
```

- Operation: It gets information whether to support escape clause in like statement from the server.
- Exception: If an error is returned from the server, it throws SQLException.

supportsLimitedOuterJoins

```
boolean supportsLimitedOuterJoins() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsMinimumSQLGrammar

```
boolean supportsMinimumSQLGrammar() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsMixedCaseIdentifiers

```
boolean supportsMixedCaseIdentifiers() throws SQLException
```

- Operation: It always returns false.
- Exception: It does not occur.

supportsMixedCaseQuotedIdentifiers

```
boolean supportsMixedCaseQuotedIdentifiers() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsMultipleOpenResults

```
boolean supportsMultipleOpenResults() throws SQLException
```

- Operation: It always returns false.
- Exception: It does not occur.

supportsMultipleResultSets

```
boolean supportsMultipleResultSets() throws SQLException
```

- Operation: It always returns false.
- Exception: It does not occur.

supportsMultipleTransactions

```
boolean supportsMultipleTransactions() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsNamedParameters

```
boolean supportsNamedParameters() throws SQLException
```

- Operation: It always returns false.
- Exception: It does not occur.

supportsNonNullableColumns

```
boolean supportsNonNullableColumns() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsOpenCursorsAcrossCommit

```
boolean supportsOpenCursorsAcrossCommit() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsOpenCursorsAcrossRollback

```
boolean supportsOpenCursorsAcrossRollback() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsOpenStatementsAcrossCommit

```
boolean supportsOpenStatementsAcrossCommit() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsOpenStatementsAcrossRollback

```
boolean supportsOpenStatementsAcrossRollback() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsOrderByUnrelated

```
boolean supportsOrderByUnrelated() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsOuterJoins

```
boolean supportsOuterJoins() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsPositionedDelete

```
boolean supportsPositionedDelete() throws SQLException
```

- Operation: It always returns false.
- Exception: It does not occur.

supportsPositionedUpdate

```
boolean supportsPositionedUpdate() throws SQLException
```

- Operation: It always returns false.
- Exception: It does not occur.

supportsResultSetConcurrency

```
boolean supportsResultSetConcurrency(int type, int concurrency) throws SQLException
```

- Operation: It returns true for all ResultSet types and all concurrency. It returns false for the invalid arg uments.
- Exception: It does not occur.

supportsResultSetHoldability

```
boolean supportsResultSetHoldability(int holdability) throws SQLException
```

- Operation: If the holdability is ResultSet.CLOSE_CURSORS_AT_COMMIT or ResultSet.HOLD_CURSOR S_OVER_COMMIT, it returns true. Otherwise, it returns false.
- Exception: It does not occur.

supportsResultSetType

```
boolean supportsResultSetType(int type) throws SQLException
```

- Operation: If the type is ResultSet.TYPE_FORWARD_ONLY, ResultSet.TYPE_SCROLL_INSENSITIVE or R esultSet.TYPE_SCROLL_SENSITIVE, it returns true. Otherwise, it returns false.
- Exception: It does not occur.

supportsSavepoints

```
boolean supportsSavepoints() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsSchemasInDataManipulation

```
boolean supportsSchemasInDataManipulation() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsSchemasInIndexDefinitions

```
boolean supportsSchemasInIndexDefinitions() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsSchemasInPrivilegeDefinitions

```
boolean supportsSchemasInPrivilegeDefinitions() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsSchemasInProcedureCalls

```
boolean supportsSchemasInProcedureCalls() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsSchemasInTableDefinitions

```
boolean supportsSchemasInTableDefinitions() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsSelectForUpdate

```
boolean supportsSelectForUpdate() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsStatementPooling

```
boolean supportsStatementPooling() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsStoredFunctionsUsingCallSyntax

```
boolean supportsStoredFunctionsUsingCallSyntax() throws SQLException
```

- Operation: It always returns false.
- Exception: It does not occur.

supportsStoredProcedures

```
boolean supportsStoredProcedures() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsSubqueriesInComparisons

```
boolean supportsSubqueriesInComparisons() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsSubqueriesInExists

```
boolean supportsSubqueriesInExists() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsSubqueriesInIns

```
boolean supportsSubqueriesInIns() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsSubqueriesInQuantifieds

```
boolean supportsSubqueriesInQuantifieds() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsTableCorrelationNames

```
boolean supportsTableCorrelationNames() throws SQLException
```

- Operation: It gets information whether to support the table correlation name from the server.
- Exception: If an error is returned from the server, it throws SQLException.

supportsTransactionIsolationLevel

```
boolean supportsTransactionIsolationLevel(int level) throws SQLException
```

- Operation: It gets information whether to support the transaction isolation level from the server.
- Exception: If an error is returned from the server, it throws SQLException.

supportsTransactions

```
boolean supportsTransactions() throws SQLException
```

- Operation: It always returns true.
- Exception: It does not occur.

supportsUnion

```
boolean supportsUnion() throws SQLException
```

- Operation: It gets information whether to support the union operation from the server.
- Exception: If an error is returned from the server, it throws SQLException.

supportsUnionAll

```
boolean supportsUnionAll() throws SQLException
```

- Operation: It gets information whether to support the union all operation from the server.
- Exception: If an error is returned from the server, it throws SQLException.

updatesAreDetected

```
boolean updatesAreDetected(int type) throws SQLException
```

- Operation: If the type is ResultSet.TYPE_SCROLL_SENSITIVE, it returns true. Otherwise, it returns false.
- Exception: It does not occur.

usesLocalFilePerTable

```
boolean usesLocalFilePerTable() throws SQLException
```

- Operation: It always returns false.
- Exception: It does not occur.

usesLocalFiles

```
boolean usesLocalFiles() throws SQLException
```

- Operation: It always returns false.
- Exception: It does not occur.

DataSource

getConnection

```
Connection getConnection() throws SQLException
```

- Operation: It opens and returns a new connection object. The information required for connection sh ould be set through a separate non-standard method in advance.
- Exception: If the connection to the server fails, it throws SQLException.

```
Connection getConnection(String username, String password) throws SQLException
```

- Operation: It opens and returns a new connection object with the username and password. Other inf ormation required for the connection should be set through a separate non-standard method in adva nce.
- Exception: If the connection to the server fails, it throws SQLException.

isWrapperFor

```
boolean isWrapperFor(Class<?> iface) throws SQLException
```

- Operation: This class is not implemented as the wrapper of any other class. If the object is an instance of iface, it returns true. Otherwise, it returns false.
- Exception: It does not occur.

unwrap

```
<T> T unwrap(Class<T> iface) throws SQLException
```

- Operation: It returns *this* because this class is not implemented as the wrapper of any other class.
- Exception: If the object is not an instance of iface, it throws SQLException.

Driver

acceptsURL

```
boolean acceptsURL(String url) throws SQLException
```

- Operation: If the url is not null and it starts with "jdbc:goldilocks:", it returns true. Otherwise, it return s false.
- Exception: It does not occur.

connect

```
Connection connect(String url, Properties info) throws SQLException
```

- Operation: It creates and returns a new connection object. The url should include the server address, DB name and port.
- Exception: If the url is invalid or the connection from the server fails, it throws SQLException.

getMajorVersion

```
int getMajorVersion() throws SQLException
```

- Operation: It returns the major version of GOLDILOCKS JDBC driver.
- Exception: It does not occur.

getMinorVersion

```
int getMinorVersion() throws SQLException
```

- Operation: It returns the minor version of GOLDILOCKS JDBC driver.
- Exception: It does not occur.

getPropertyInfo

```
DriverPropertyInfo[] getPropertyInfo(String url, Properties info) throws SQLException
```

- Operation: It gets the list of available property when GOLDILOCKS JDBC driver is connected.
- Exception: It does not occur.

jdbcCompliant

```
boolean jdbcCompliant() throws SQLException
```

- Operation: It always returns false.
- Exception: It does not occur.

NClob

This class is not implemented.

free

```
void free() throws SQLException
```

getAsciiStream

```
InputStream getAsciiStream() throws SQLException
```

getCharacterStream

```
Reader getCharacterStream() throws SQLException
```

```
Reader getCharacterStream(long pos, long length) throws SQLException
```

getSubString

```
String getSubString(long pos, int length) throws SQLException
```

length

```
long length() throws SQLException
```

position

```
long position(Clob searchstr, long start) throws SQLException
```

```
long position(String searchstr, long start) throws SQLException
```

setAsciiStream

```
OutputStream setAsciiStream(long pos) throws SQLException
```

setCharacterStream

```
Writer setCharacterStream(long pos) throws SQLException
```

setString

```
int setString(long pos, String str) throws SQLException
```

```
int setString(long pos, String str, int offset, int len) throws SQLException
```

truncate

```
void truncate(long len) throws SQLException
```

ParameterMetaData

ParameterMetaData object is returned by PreparedStatement.getParameterMetaData(). The parameterMetaData of the GOLDILOCKS JDBC does not use the actual DB information but it is based on the basic type varchar for other information (such as type, etc.) except for in/out. It is because in/out is the only information got from the server on the parameter after prepared. For example, if it is prepared with the query statement which is "Select * from t1 where a =?", the parameter type used in the condition clause is not determined. The server generally assumes it a varchar type.

getParameterClassName

```
String getParameterClassName(int param) throws SQLException
```

- Operation: It returns java.lang.String. The parameter type is regarded as varchar.
- Exception: If the param value exceeds the range of the number of parameters, it throws SQLException.

getParameterCount

```
int getParameterCount() throws SQLException
```

- Operation: It returns the number of parameters. It is also the number of ? used in the query statement.
- Exception: It does not occur.

getParameterMode

```
int getParameterMode(int param) throws SQLException
```

- Operation: It returns the in/out mode of the param-th parameter. It returns one of ParameterMetaData.parameterModeIn, ParameterMetaData.parameterModeInOut, ParameterMetaData.parameterModeOut, or ParameterMetaData.parameterModeUnknown. The case of returning parameterModeUnknown value does not exist until now.
- Exception: If the param value exceeds range of the number of parameters, it throws SQLException.

getParameterType

```
int getParameterType(int param) throws SQLException
```

- Operation: It returns Types.VARCHAR for all parameters.
- Exception: If the param value exceeds range of the number of parameters, it throws SQLException.

getParameterTypeName

```
String getParameterTypeName(int param) throws SQLException
```

- Operation: It returns "VARCHAR" for all parameters.
- Exception: If the param value exceeds range of the number of parameters, it throws SQLException.

getPrecision

```
int getPrecision(int param) throws SQLException
```

- Operation: It returns 4000 for all parameters. The parameter is regarded as varchar(4000) by default.
- Exception: If the param value exceeds range of the number of parameters, it throws SQLException.

getScale

```
int getScale(int param) throws SQLException
```

- Operation: It returns 0 for all parameters. The parameter is regarded as varchar(4000) by default.
- Exception: If the param value exceeds range of the number of parameters, it throws SQLException.

isNullable

```
int isNullable(int param) throws SQLException
```

- Operation: It always returns ParameterMetaData.parameterNullableUnknown.
- Exception: If the param value exceeds range of the number of parameters, it throws SQLException.

isSigned

```
boolean isSigned(int param) throws SQLException
```

- Operation: It always returns false.
- Exception: If the param value exceeds range of the number of parameters, it throws SQLException.

isWrapperFor

```
boolean isWrapperFor(Class<?> iface) throws SQLException
```

- Operation: It enquires if it the object is an instance of iface. If so, it returns true. Otherwise, it returns false.
- Exception: It does not occur.

unwrap

```
<T> T unwrap(Class<T> iface) throws SQLException
```

- Operation: If the object is an instance of iface, it casts this object to iface type and returns it.
- Exception: If the object is not an instance of iface, it throws SQLException.

PooledConnection

addConnectionEventListener

```
void addConnectionEventListener(ConnectionEventListener listener) throws SQLException
```

- Operation: It registers the ConnectionEventListener object. After then, if close() of the connection obj ect (logical connection) which is returned from PooledConnection is called or the actual connection is broken, ConnectionEvent is generated to the registered listeners.
- Exception: It does not occur.

addStatementEventListener

```
void addStatementEventListener(StatementEventListener listener) throws SQLException
```

- Operation: It does not perform any operation. GOLDILOCKS JDBC driver does not implement the met hod. Statement pooling feature is performed by the external middleware.
- Exception: It does not occur.

close

```
void close() throws SQLException
```

- Operation: It calls close() of the physical connection that the object has.
- Exception: It may occur in close() of physical connection.

getConnection

```
Connection getConnection() throws SQLException
```

- Operation: It returns logical connection owned by the object.
- Exception: It does not occur.

removeConnectionEventListener

```
void removeConnectionEventListener(ConnectionEventListener listener) throws SQLException
```

- Operation: It removes the registered ConnectionEventListener. After then, ConnectionEvent is not tra nsferred to this listener.

- Exception: It does not occur.

removeStatementEventListener

```
void removeStatementEventListener(StatementEventListener listener) throws SQLException
```

- Operation: It does not perform any operation.
- Exception: It does not occur.

PreparedStatement

addBatch

```
void addBatch() throws SQLException
```

- Operation: It registers the currently bound data as batch job. If any of this batch job is registered, an error occurs when performing execute (), executeUpdate (), executeQuery () execution. If any one is not bound for the parameter, an error occurs. After addBatch if another addBatch is performed again at the state without binding with the method of setXXX () type, it registers the batch job as the previously bound value.
- Exception: If a parameter which has never been bound exists, it throws an exception.

clearParameters

```
void clearParameters() throws SQLException
```

- Operation: It removes all of currently bound data and information. But if at least one batch job is registered, operation is not performed.
- Exception: It does not occur.

execute

```
boolean execute() throws SQLException
```

- Operation: It executes the prepared statement based on the bound data to the current parameter. If the executed statement is the select statement, it returns true and gets a ResultSet from the object. Otherwise, it returns false.
- Exception: If the batch job is registered, the bound parameters are insufficient or an error occurs on the server when executing, it throws an exception.

executeQuery

```
ResultSet executeQuery() throws SQLException
```

- Operation: It executes the prepared statement based on the bound data to the current parameter and fetches, then creates and returns a ResultSet object.
- Exception: If the batch job is registered or the bound parameters are insufficient or an error occurs on the server when executing, it throws an exception.

executeUpdate

```
int executeUpdate() throws SQLException
```

- Operation: It executes the prepared statement based on the bound data to the current parameter. It r eturns the number of updated records. If updated record with DDL statements does not exist, it retur ns 0.
- Exception: If the batch job is registered or the statement is not a select statement or the bound para meters are insufficient or an error occurs on the server when executing, it throws an exception.

getMetaData

```
ResultSetMetaData getMetaData() throws SQLException
```

- Operation: It gets ResultSetMetaData object. If the prepared statement is not the select statement (It is the statement which does not return ResultSet), it returns an empty ResultSetMetaData. The meth od can be called before executing.
- Exception: If an error occurs on the server, it throws an exception.

getParameterMetaData

```
ParameterMetaData getParameterMetaData() throws SQLException
```

- Operation: It gets the ParameterMetaData object. It can be called before executing. However, all par ameters are assumed to be varchar(4000) because information about the exact parameter type is not known to the server.
- Exception: If an error occurs on the server, it throws an exception.

setArray

```
void setArray(int parameterIndex, Array x) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setAsciiStream

```
void setAsciiStream(int parameterIndex, InputStream x) throws SQLException
```

- Operation: It binds the InputStream object to the parameter index as LONG VARCHAR type. It does n

ot execute the encoding operation because the input data is a binary form. It does not consider the c haracter set and assumes it as ascii data.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

⚠️ The cost of conversion to VARCHAR occurs in the server because it is bound to LONG VARCHA R. If the length can be known, it is recommended to use void setAsciiStream(int parameterIndex, InputStream x, int length).

```
void setAsciiStream(int parameterIndex, InputStream x, int length) throws SQLException
```

- Operation: It binds the InputStream object to the parameter index as LONG VARCHAR type. It does n ot execute the encoding operation because the input data is a binary form. It does not consider the c haracter set and and assumes it as ascii data. If English data is inserted or the server client character s et environment is identical, this method is faster than setCharacterStream or setString when inserting the data as varchar.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

```
void setAsciiStream(int parameterIndex, InputStream x, long length) throws SQLException
```

- Operation: It binds the InputStream object to the parameter index as LONG VARCHAR type. It does n ot execute the encoding operation because the input data is a binary form. It does not consider the c haracter set and assumes it as ascii data. If English data is inserted or the server client character set en vironment is identical, this method is faster than setCharacterStream or setString when inserting the data as varchar.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

setBigDecimal

```
void setBigDecimal(int parameterIndex, BigDecimal x) throws SQLException
```

- Operation: It binds the BigDecimal object to the parameter index as NUMBER type.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

setBinaryStream

```
void setBinaryStream(int parameterIndex, InputStream x) throws SQLException
```

- Operation: It binds the InputStream object to the parameter index as LONG VARBINARY type.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

```
void setBinaryStream(int parameterIndex, InputStream x, int length) throws SQLException
```

- Operation: It binds the InputStream object to the parameter index as LONG VARBINARY type.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

```
void setBinaryStream(int parameterIndex, InputStream x, long length) throws SQLException
```

- Operation: It binds the InputStream object to the parameter index as LONG VARBINARY type.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

setBlob

```
void setBlob(int parameterIndex, Blob x) throws SQLException
```

- Operation: It binds the blob object to the parameter index as LONG VARBINARY type.
- Exception: If parameterIndex is smaller than 0 or it is bigger than the number of parameters, it throw s SQLException.

```
void setBlob(int parameterIndex, InputStream inputStream) throws SQLException
```

- Operation: It binds the InputStream object to the parameter index as LONG VARBINARY type.
- Exception: If parameterIndex is smaller than 0 or it is bigger than the number of parameters, it throw s SQLException.

```
void setBlob(int parameterIndex, InputStream inputStream, long length) throws SQLException
```

- Operation: It binds the InputStream object to the parameter index as LONG VARBINARY type. The nu mber of characters included in InputStream is as many as the length.
- Exception: If parameterIndex is smaller than 0 or it is bigger than the number of parameters, it throw s SQLException.

setBoolean

```
void setBoolean(int parameterIndex, boolean x) throws SQLException
```

- Operation: It binds the data x to the parameter index as BOOLEAN type.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

setByte

```
void setByte(int parameterIndex, byte x) throws SQLException
```

- Operation: It binds the data x to the parameter index as NATIVE_SMALLINT type.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

setBytes

```
void setBytes(int parameterIndex, byte[] x) throws SQLException
```

- Operation: It binds the data x to the parameter index as VARBINARY or LONG VARBINARY type. If th e length of x is equal to or smaller than 4000, it is bound as VARBINARY, and if it is bigger than 4000, it is bound as LONG VARBINARY type.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

setCharacterStream

```
void setCharacterStream(int parameterIndex, Reader reader) throws SQLException
```

- Operation: It binds the reader object to the parameter index as LONG VARCHAR type.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

```
void setCharacterStream(int parameterIndex, Reader reader, int length) throws SQLException
```

- Operation: It binds the reader object to the parameter index as LONG VARCHAR type.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

```
void setCharacterStream(int parameterIndex, Reader reader, long length) throws SQLException
```

- Operation: It binds the reader object to the parameter index as LONG VARCHAR type.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

setClob

```
void setClob(int parameterIndex, Clob x) throws SQLException
```

- Operation: It binds the clob object to the parameter index as LONG VARCHAR type.
- Exception: If parameterIndex is smaller than 0 or it is bigger than the number of parameters, it throw s SQLException.

```
void setClob(int parameterIndex, Reader reader) throws SQLException
```

- Operation: It binds the reader object to the parameter index as LONG VARCHAR type.
- Exception: If parameterIndex is smaller than 0 or it is bigger than the number of parameters, it throw s SQLException.

```
void setClob(int parameterIndex, Reader reader, long length) throws SQLException
```

- Operation: It binds the reader object to the parameter index as LONG VARCHAR type. The number of characters included in reader is as many as the length.
- Exception: If parameterIndex is smaller than 0 or it is bigger than the number of parameters, it throw s SQLException.

setDate

```
void setDate(int parameterIndex, Date x) throws SQLException
```

- Operation: It binds the data x to the parameter index as DATE type. It is equivalent to calling setDate (parameterIndex, x, Calendar.getInstance()). The local timezone is applied.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

```
void setDate(int parameterIndex, Date x, Calendar cal) throws SQLException
```

- Operation: It binds the data x to the parameter index as DATE type. Date x is regarded as timezone of cal.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

setDouble

```
void setDouble(int parameterIndex, double x) throws SQLException
```

- Operation: It binds the data x to the parameter index as NATIVE_DOUBLE type.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

setFixedCHAR

```
void setFixedCHAR(int parameterIndex, String x) throws SQLException
```

- Operation: It binds the data x to the parameter index as CHAR type.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

setFloat

```
void setFloat(int parameterIndex, float x) throws SQLException
```

- Operation: It binds the data x to the parameter index as NATIVE_REAL type.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

setInt

```
void setInt(int parameterIndex, int x) throws SQLException
```

- Operation: It binds the data x to the parameter index as NATIVE_INTEGER type.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

setLong

```
void setLong(int parameterIndex, long x) throws SQLException
```

- Operation: It binds the data x to the parameter index as NATIVE_BIGINT type.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

setNCharacterStream

```
void setNCharacterStream(int parameterIndex, Reader value) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

```
void setNCharacterStream(int parameterIndex, Reader value, long length) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setNClob

```
void setNClob(int parameterIndex, NClob value) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

```
void setNClob(int parameterIndex, Reader reader) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

```
void setNClob(int parameterIndex, Reader reader, long length) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setNString

```
void setNString(int parameterIndex, String value) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setNull

```
void setNull(int parameterIndex, int sqlType) throws SQLException
```

- Operation: It binds null to the parameter index as GOLDILOCKS type corresponding to sqlType. For more information about GOLDILOCKS type mapped to sqlType, refer to **SQL types → GOLDILOCKS types**.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

```
void setNull(int parameterIndex, int sqlType, String typeName) throws SQLException
```

- Operation: It binds null to the parameter index as GOLDILOCKS type corresponding to sqlType type. For more information about GOLDILOCKS type mapped to sqlType, refer to **SQL types → GOLDILOCKS types**. The third argument, typeName, is ignored because REF or the user type is not supported.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

setObject

```
void setObject(int parameterIndex, Object x) throws SQLException
```

- Operation: It binds the data x to the parameter index as the mapped GOLDILOCKS type.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

Table 32-2 Java objects → GOLDILOCKS types

Java class	GOLDILOCKS type
null	VARCHAR
Boolean	BOOLEAN
Byte	NATIVE_SMALLINT
Short	NATIVE_SMALLINT
Integer	NATIVE_INTEGER
Long	NATIVE_BIGINT
Float	NATIVE_REAL
Double	NATIVE_DOUBLE
BigInteger	NATIVE_BIGINT
BigDecimal	NUMBER
String	VARCHAR or LONG VARCHAR
byte[]	VARBINARY or LONG VARBINARY
Date	DATE
Time	TIME
Timestamp	TIMESTAMP
Blob	N/A
Clob	N/A
InputStream	LONG VARBINARY
Reader	LONG VARCHAR
GoldilocksInterval	Corresponding INTERVAL type
RowId	ROWID
Others	N/A

```
void setObject(int parameterIndex, Object x, int targetSqlType) throws SQLException
```

- Operation: It binds the data x to the parameter index as GOLDILOCKS type corresponding to targetSqlType. For more information about GOLDILOCKS type mapped to targetSqlType, refer to **SQL types → GOLDILOCKS types**.
- Exception: It occurs if parameterIndex is smaller than 0 or it is greater than the number of parameters.

```
void setObject(int parameterIndex, Object x, int targetSqlType, int scaleOrLength) throws SQLException
```

- Operation: It binds the data x to the parameter index as GOLDILOCKS type corresponding to targetSqlType. For more information about GOLDILOCKS type mapped to targetSqlType, refer to **SQL types → GOLDILOCKS types**. If x is InputStream or Reader then scaleOrLength indicates the data length. For other types this value is ignored.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

setRef

```
void setRef(int parameterIndex, Ref x) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setRowId

```
void setRowId(int parameterIndex, RowId x) throws SQLException
```

- Operation: It binds the data x to the parameter index as ROWID type.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

setShort

```
void setShort(int parameterIndex, short x) throws SQLException
```

- Operation: It binds the data x to the parameter index as NATIVE_SMALLINT type.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

setSQLXML

```
void setSQLXML(int parameterIndex, SQLXML xmlObject) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

setString

```
void setString(int parameterIndex, String x) throws SQLException
```

- Operation: It binds the data x to the parameter index as VARCHAR or LONG VARCHAR type. If the le ngth of x is equal to or smaller than 4000, it is bound as VARCHAR, and if it is bigger than 4000, it is bound as LONG VARCHAR type.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

setTime

```
void setTime(int parameterIndex, Time x) throws SQLException
```

- Operation: It binds the data x to the parameter index as TIME type. It is equivalent to calling setTime (parameterIndex, x, Calendar.getInstance()). The local timezone is applied.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

```
void setTime(int parameterIndex, Time x, Calendar cal) throws SQLException
```

- Operation: It binds the data x to the parameter index as TIME type. Time x is regarded as timezone of cal.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

setTimestamp

```
void setTimestamp(int parameterIndex, Timestamp x) throws SQLException
```

- Operation: It binds the data x to the parameter index as TIME type. It is equivalent to calling setTimes tamp(parameterIndex, x, Calendar.getInstance()). The local timezone is applied.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

```
void setTimestamp(int parameterIndex, Timestamp x, Calendar cal) throws SQLException
```

- Operation: It binds the data x to the parameter index as TIMESTAMP type. Timestamp x is regarded a s timezone of cal.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

setUnicodeStream

```
void setUnicodeStream(int parameterIndex, InputStream x, int length) throws SQLException
```

- Operation: It is not implemented. (It is a deprecated method.)
- Exception: It always throws SQLFeatureNotSupportedException.

setURL

```
void setURL(int parameterIndex, URL x) throws SQLException
```

- Operation: It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

isWrapperFor

```
boolean isWrapperFor(Class<?> iface) throws SQLException
```

- Operation: It queries whether this object is the class implementing the iface interface. If so, it returns true. Otherwise, it returns false. It does not determine the presence of the wrapper and it queries onl y whether the given argument class type is implemented because GOLDILOCKS PreparedStatement o bject is not the wrapper of any other class.
- Exception: It does not occur.

unwrap

```
<T> T unwrap(Class<T> iface) throws SQLException
```

- Operation: It eventually returns itself even when it is unwrapped because GOLDILOCKS PreparedStat ement is not the wrapper of any other class. It returns after casting to that type. If iface is an argume nt of isWrapperFor() method and false is returned, the method throws an exception.
- Exception: If iface is not this object type (The type which is not implemented by this object is given), i t throws SQLException.

executeBatchAtomic

```
boolean executeBatchAtomic() throws SQLException
```

- Operation: It is identical to executeBatch(), but it is atomically executed. The batch job is either entirel y succeeded or failed. It is performed faster than executeBatch(). If the executed statement is the sele ct statement, it returns true, otherwise, it returns false.
- Exception: If batch job is not registered or an error occurs on the server, it throws SQLException.

It is GOLDILOCKS JDBC-specific feature, and PreparedStatement object can be used after it is ca sted as GoldilocksPreparedStatement.
e.g. ((GoldilocksPreparedStatement)pstmt).executeBatchAtomic();

setTimeTimeZone

```
void setTimeTimeZone(int parameterIndex, Time x, Calendar cal) throws SQLException
```

- Operation: It binds the data x to the parameter index as TIME WITH TIME ZONE type. Time x is regard ed as timezone of cal. Timezone information for DB column refers to timezone of cal.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

It is GOLDILOCKS JDBC-specific feature, and PreparedStatement object can be used after it is ca sted as GoldilocksPreparedStatement.
e.g. ((GoldilocksPreparedStatement)pstmt).setTimeTimeZone(1, aTime, aCalendar);

setTimestampTimeZone

```
void setTimestampTimeZone(int parameterIndex, Timestamp x, Calendar cal) throws SQLException
```

- Operation: It binds the data x to the parameter index as TIMESTAMP WITH TIME ZONE type. Timesta mp x is regarded as timezone of cal. Timezone information for DB column refers to timezone of cal.
- Exception: It occurs if parameterIndex is smaller than 0 or it is bigger than the number of parameters.

It is GOLDILOCKS JDBC-specific feature, and PreparedStatement object can be used after it is ca sted as GoldilocksPreparedStatement.
e.g. ((GoldilocksPreparedStatement)pstmt).setTimestampTimeZone(1, aTimestamp, aCalendar);

Ref

The class is not implemented.

getBaseTypeName

```
String getBaseTypeName() throws SQLException
```

getObject

```
Object getObject() throws SQLException
```

```
Object getObject(Map<String,Class<?>> map) throws SQLException
```

setObject

```
void setObject(Object value) throws SQLException
```

ResultSet

absolute

```
boolean absolute(int row) throws SQLException
```

- Operation: The fetched row cursor position is at the row-th. The first row is 1. 0 points to the previou s first row. If it is a negative number, it points to the last row. -1 points to the last row, and -2 points to the second row from the last row. If the cursor can be positioned within the fetched row cache, o nly the position information is changed within the cache. If it is not within the cache, it is fetched fro m the server again. If the row exceeds the range, the cursor is positioned before first or after last, and it returns false. Otherwise, the cursor is positioned at the corresponding row and it returns true.
- Exception: If ResultSet is closed or the ResultSet type is ResultSet.TYPE_FORWARD_ONLY or an error occurs from the server when fetching, it throws an exception.

If it is not in the cache, it should be fetched again. If the row is behind the current position, the number of rows(n) from row position are fetched from the server in favor of next(). If the row is prior to the current position, the number of n rows from the position(row-n+1) are fetched fro m the server in favor of previous().

afterLast

```
void afterLast() throws SQLException
```

- Operation: The row cursor is positioned at *after last*. If the row cache is the last row set (It is a part of the entire result set), only the cursor position is changed. Otherwise, it fetches the last row set (The t otal number of rows-n + 1 to n rows, n is the number of rows fetched from the server), then position s the cursor at *after last*.
- Exception: If ResultSet is closed or the ResultSet type is ResultSet.TYPE_FORWARD_ONLY or an error occurs from the server when fetching, it throws an exception.

beforeFirst

```
void beforeFirst() throws SQLException
```

- Operation: The row cursor is positioned at *before first*. If the row cache is the first row set (It is a part of the entire result set), only the cursor position is changed. Otherwise, it fetches the first row set (1 t o n rows, n is the number of rows fetched from the server), then positions the cursor at *before first*.
- Exception: If ResultSet is closed or the ResultSet type is ResultSet.TYPE_FORWARD_ONLY or an error occurs from the server when fetching, it throws an exception.

cancelRowUpdates

```
void cancelRowUpdates() throws SQLException
```

- Operation: Cursor update feature has not been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

clearWarnings

```
void clearWarnings() throws SQLException
```

- Operation: It clears all SQLWarning objects which is owned by ResultSet object.
- Exception: It does not occur.

close

```
void close() throws SQLException
```

- Operation: If the cursor is open on the server, it closes the cursor (If the cursor is already closed on the server, this operation is not performed. Namely, the protocol is not transferred.), and the current state of the ResultSet object is changed to be closed. If it is already closed, then any operation is not performed.
- Exception: If an error occurs on the server, it throws SQLException.

deleteRow

```
void deleteRow() throws SQLException
```

- Operation: Cursor update feature is not implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

findColumn

```
int findColumn(String columnLabel) throws SQLException
```

- Operation: It returns the index of the column name. The index of the first column is 1.

- Exception: If it is already closed or the column name is not found, it throws SQLException.

first

```
boolean first() throws SQLException
```

- Operation: It positions the row cursor at the first (the first row). If the row cache is the first row set (It is a part of the entire result set), only the cursor position is changed. Otherwise, it fetches the first row set (1 to n rows, n is the number of rows fetched from the server), then positions the cursor at the first. If the row exists, it returns true. Otherwise, it returns false.
- Exception: If ResultSet is closed or the ResultSet type is ResultSet.TYPE_FORWARD_ONLY or an error occurs from the server when fetching, it throws an exception.

getArray

```
Array getArray(int columnIndex) throws SQLException
```

- Operation: Array type is not supported.
- Exception: It always throws SQLFeatureNotSupportedException.

```
Array getArray(String columnLabel) throws SQLException
```

- Operation: Array type is not supported. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: It always throws SQLFeatureNotSupportedException.

getAsciiStream

```
InputStream getAsciiStream(int columnIndex) throws SQLException
```

- Operation: It gets the columnIndex-th column data as InputStream type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If ResultSet is already closed or the columnIndex exceeds the range or the type can not be converted to InputStream, it throws SQLException.

```
InputStream getAsciiStream(String columnLabel) throws SQLException
```

- Operation: It gets the column data whose name is columnLabel as InputStream type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If ResultSet is already closed or the corresponding columnLabel does not exist or the type

can not be converted to InputStream, it throws SQLException.

getBigDecimal

```
BigDecimal getBigDecimal(int columnIndex) throws SQLException
```

- Operation: It gets the columnIndex-th column data as BigDecimal type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If ResultSet is already closed or the columnIndex exceeds the range or the type can not be converted to BigDecimal, it throws SQLException.

```
BigDecimal getBigDecimal(int columnIndex, int scale) throws SQLException
```

- Operation: This is a deprecated method. It operates in the same way as getBigDecimal(int columnIndex). The scale is ignored.
- Exception: For more information, refer to **getBigDecimal**(int columnIndex).

```
BigDecimal getBigDecimal(String columnLabel) throws SQLException
```

- Operation: It gets the column data whose name is columnLabel as BigDecimal type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If ResultSet is already closed or the corresponding columnLabel does not exist or the type can not be converted to BigDecimal, it throws SQLException.

```
BigDecimal getBigDecimal(String columnLabel, int scale) throws SQLException
```

- Operation: This is a deprecated method. It operates in the same way as getBigDecimal(String columnLabel). The scale is ignored.
- Exception: For more information, refer to **getBigDecimal**(String columnLabel).

getBinaryStream

```
InputStream getBinaryStream(int columnIndex) throws SQLException
```

- Operation: It is as same as **getAsciiStream**(int columnIndex).
- Exception: For more information, refer to **getAsciiStream**(int columnIndex).

```
InputStream getBinaryStream(String columnLabel) throws SQLException
```

- Operation: It is as same as **getAsciiStream**(String columnLabel).

- Exception: For more information, refer to **getAsciiStream**(String columnLabel).

getBlob

```
Blob getBlob(int columnIndex) throws SQLException
```

- Operation: It gets the columnIndex-th column data as blob type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If ResultSet is already closed or the columnIndex exceeds the range or the type can not be converted to blob, it throws SQLException.

```
Blob getBlob(String columnLabel) throws SQLException
```

- Operation: It gets the column data whose name is columnLabel as blob type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If ResultSet is already closed or the corresponding columnLabel does not exist or the type can not be converted to blob, it throws SQLException.

getBoolean

```
boolean getBoolean(int columnIndex) throws SQLException
```

- Operation: It gets the columnIndex-th column data as Boolean type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If ResultSet is already closed or the columnIndex exceeds the range or the type can not be converted to Boolean, it throws SQLException.

```
boolean getBoolean(String columnLabel) throws SQLException
```

- Operation: It gets the column data whose name is columnLabel as Boolean type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If ResultSet is already closed or the corresponding columnLabel does not exist or the type can not be converted to Boolean, it throws SQLException.

getByte

```
byte getByte(int columnIndex) throws SQLException
```

- Operation: It gets the columnIndex-th column data as byte type. For more information about GOLDIL

OCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .

- Exception: If ResultSet is already closed or the columnIndex exceeds the range or the type can not be converted to byte, it throws SQLException.

```
byte getByte(String columnLabel) throws SQLException
```

- Operation: It gets the column data whose name is columnLabel as byte type. For more information a bout GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCK S type - 1** .
- Exception: If ResultSet is already closed or the corresponding columnLabel does not exist or the type can not be converted to byte, it throws SQLException.

getBytes

```
byte[] getBytes(int columnIndex) throws SQLException
```

- Operation: It gets the columnIndex-th column data as byte[] type. For more information about GOLDI LOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If ResultSet is already closed or the columnIndex exceeds the range, it throws SQLExceptio n.

If getBytes is performed for all GOLDILOCKS data types, it gets the binary form stored in DB.

```
byte[] getBytes(String columnLabel) throws SQLException
```

- Operation: It gets the column data whose name is columnLabel as byte[] type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOC KS type - 1** .
- Exception: If ResultSet is already closed or the corresponding columnLabel does not exist, it throws S QLException.

If getBytes is performed for all GOLDILOCKS data types, it gets the binary form stored in DB.

getCharacterStream

```
Reader getCharacterStream(int columnIndex) throws SQLException
```

- Operation: It gets the columnIndex-th column data as reader type. For more information about GOLD ILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If ResultSet is already closed or the columnIndex exceeds the range or the type can not be

converted to reader, it throws SQLException.

```
Reader getCharacterStream(String columnLabel) throws SQLException
```

- Operation: It gets the column data whose name is columnLabel as reader type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If ResultSet is already closed or the corresponding columnLabel does not exist or the type can not be converted to reader, it throws SQLException.

getClob

```
Clob getClob(int columnIndex) throws SQLException
```

- Operation: It gets the columnIndex-th column data as clob type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If ResultSet is already closed or the columnIndex exceeds the range or the type can not be converted to clob, it throws SQLException.

```
Clob getClob(String columnLabel) throws SQLException
```

- Operation: It gets the column data whose name is columnLabel as clob type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If ResultSet is already closed or the corresponding columnLabel does not exist or the type can not be converted to clob, it throws SQLException.

getConcurrency

```
int getConcurrency() throws SQLException
```

- Operation: It returns concurrency of the current ResultSet object. It supports only ResultSet.CONCUR_READ_ONLY currently.
- Exception: It does not occur.

getCursorName

```
String getCursorName() throws SQLException
```

- Operation: It gets the cursor name of the server pointed by the ResultSet. The communication with the server occurs.
- Exception: If ResultSet is already closed or an error occurs on the server, it throws SQLException.

getDate

```
Date getDate(int columnIndex) throws SQLException
```

- Operation: It gets the columnIndex-th column data as date type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** . When creating the date object, local timezone is used.
- Exception: If ResultSet is already closed or the columnIndex exceeds the range or the type can not be converted to date, it throws SQLException.

```
Date getDate(int columnIndex, Calendar cal) throws SQLException
```

- Operation: It gets the columnIndex-th column data as date type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** . When creating the date object, local timezone of cal is used.
- Exception: If ResultSet is already closed or the columnIndex exceeds the range or the type can not be converted to date, it throws SQLException.

```
Date getDate(String columnLabel) throws SQLException
```

- Operation: It gets the column data whose name is columnLabel as date type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** . When creating the date object, the local timezone is used.
- Exception: If ResultSet is already closed or the corresponding columnLabel does not exist or the type can not be converted to date, it throws SQLException.

```
Date getDate(String columnLabel, Calendar cal) throws SQLException
```

- Operation: It gets the column data whose name is columnLabel as date type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** . When creating the date object, the local timezone of cal is used.
- Exception: If ResultSet is already closed or the corresponding columnLabel does not exist or the type can not be converted to date, it throws SQLException.

getDouble

```
double getDouble(int columnIndex) throws SQLException
```

- Operation: It gets the columnIndex-th column data as double type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If ResultSet is already closed or the columnIndex exceeds the range or the type can not be converted to double, it throws SQLException.

```
double getDouble(String columnLabel) throws SQLException
```

- Operation: It gets the column data whose name is columnLabel as double type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If ResultSet is already closed or the corresponding columnLabel does not exist or the type can not be converted to double, it throws SQLException.

getFetchDirection

```
int getFetchDirection() throws SQLException
```

- Operation: It always returns ResultSet.FETCH_FORWARD. The backward fetch is not supported.
- Exception: If ResultSet is already closed, it throws SQLException.

getFetchSize

```
int getFetchSize() throws SQLException
```

- Operation: It gets the number of rows fetched from the server at once. If it is 0, it calculates the maximum number of rows included in a communication packet per transmission. The default value is 0.
- Exception: If ResultSet is already closed, it throws SQLException.

getFloat

```
float getFloat(int columnIndex) throws SQLException
```

- Operation: It gets the columnIndex-th column data as float type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If ResultSet is already closed or the columnIndex exceeds the range or the type can not be converted to float, it throws SQLException.

```
float getFloat(String columnLabel) throws SQLException
```

- Operation: It gets the column data whose name is columnLabel as float type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If ResultSet is already closed or the corresponding columnLabel does not exist or the type can not be converted to float, it throws SQLException.

getHoldability

```
int getHoldability() throws SQLException
```

- Operation: It returns the holdability of the current ResultSet. It is the value determined when the ResultSet object is created, and it can not be changed in the meantime. The default value is ResultSet.HOLD_CURSOR_OVER_COMMIT.
- Exception: If ResultSet is already closed, it throws SQLException.

getInt

```
int getInt(int columnIndex) throws SQLException
```

- Operation: It gets the columnIndex-th column data as int type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If ResultSet is already closed or the columnIndex exceeds the range or the type can not be converted to int, it throws SQLException.

```
int getInt(String columnLabel) throws SQLException
```

- Operation: It gets the column data whose name is columnLabel as int type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If ResultSet is already closed or the corresponding columnLabel does not exist or the type can not be converted to int, it throws SQLException.

getLong

```
long getLong(int columnIndex) throws SQLException
```

- Operation: It gets the columnIndex-th column data as long type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If ResultSet is already closed or the columnIndex exceeds the range or the type can not be converted to long, it throws SQLException.

```
long getLong(String columnLabel) throws SQLException
```

- Operation: It gets the column data whose name is columnLabel as long type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If ResultSet is already closed or the corresponding columnLabel does not exist or the type can not be converted to long, it throws SQLException.

getMetaData

```
ResultSetMetaData getMetaData() throws SQLException
```

- Operation: It creates and returns the ResultSetMetaData object getting detailed information on the c olumn.
- Exception: If ResultSet is already closed or an error occurs while detailed information on the column i s fetched from the server, it throws SQLException.

getNCharacterStream

```
Reader getNCharacterStream(int columnIndex) throws SQLException
```

- Operation: NCHAR family types are not supported currently.
- Exception: It always throws SQLFeatureNotSupportedException.

```
Reader getNCharacterStream(String columnLabel) throws SQLException
```

- Operation: NCHAR family types are not supported currently.
- Exception: It always throws SQLFeatureNotSupportedException.

getNClob

```
NClob getNClob(int columnIndex) throws SQLException
```

- Operation: NCHAR family types are not supported currently.
- Exception: It always throws SQLFeatureNotSupportedException.

```
NClob getNClob(String columnLabel) throws SQLException
```

- Operation: NCHAR family types are not supported currently.
- Exception: It always throws SQLFeatureNotSupportedException.

getNString

```
String getNString(int columnIndex) throws SQLException
```

- Operation: NCHAR family types are not supported currently.
- Exception: It always throws SQLFeatureNotSupportedException.

```
String getNString(String columnLabel) throws SQLException
```

- Operation: NCHAR family types are not supported currently.
- Exception: It always throws SQLFeatureNotSupportedException.

getObject

```
Object getObject(int columnIndex) throws SQLException
```

- Operation: It gets the columnIndex-th column data as the most appropriate Java object type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If ResultSet is already closed or the columnIndex exceeds the range, it throws SQLException.

```
Object getObject(int columnIndex, Map<String,Class<?>> map) throws SQLException
```

- Operation: It is not supported.
- Exception: It always throws SQLFeatureNotSupportedException.

```
Object getObject(String columnLabel) throws SQLException
```

- Operation: It gets the column data whose name is columnLabel as the most appropriate Java object type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If ResultSet is already closed or the corresponding columnLabel does not exist, it throws SQLException.

```
Object getObject(String columnLabel, Map<String,Class<?>> map) throws SQLException
```

- Operation: It is not supported.
- Exception: It always throws SQLFeatureNotSupportedException.

getRef

```
Ref getRef(int columnIndex) throws SQLException
```

- Operation: REF type is not supported.
- Exception: It always throws SQLFeatureNotSupportedException.

```
Ref getRef(String columnLabel) throws SQLException
```

- Operation: REF type is not supported.
- Exception: It always throws SQLFeatureNotSupportedException.

getRow

```
int getRow() throws SQLException
```

- Operation: It returns the cursor position of the current ResultSet object. The first row is 1. If it is *befor e first*, it returns 0.
- Exception: If ResultSet is already closed, it throws SQLException.

getRowId

```
RowId getRowId(int columnIndex) throws SQLException
```

- Operation: It gets the columnIndex-th column data as RowId type. For more information about GOLD ILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If ResultSet is already closed or the columnIndex exceeds the range or the type can not be converted to RowId, it throws SQLException.

```
RowId getRowId(String columnLabel) throws SQLException
```

- Operation: It gets the column data whose name is columnLabel as RowId type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOC KS type - 1** .
- Exception: If ResultSet is already closed or the corresponding columnLabel does not exist or the type can not be converted to RowId, it throws SQLException.

getShort

```
short getShort(int columnIndex) throws SQLException
```

- Operation: It gets the columnIndex-th column data as short type. For more information about GOLDI LOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If ResultSet is already closed or the columnIndex exceeds the range or the type can not be converted to short, it throws SQLException.

```
short getShort(String columnLabel) throws SQLException
```

- Operation: It gets the column data whose name is columnLabel as short type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** .
- Exception: If ResultSet is already closed or the corresponding columnLabel does not exist or the type can not be converted to short, it throws SQLException.

getSQLXML

```
SQLXML getSQLXML(int columnIndex) throws SQLException
```

- Operation: SQLXML type is not supported.
- Exception: It always throws SQLFeatureNotSupportedException.

```
SQLXML getSQLXML(String columnLabel) throws SQLException
```

- Operation: SQLXML type is not supported.
- Exception: It always throws SQLFeatureNotSupportedException.

getStatement

```
Statement getStatement() throws SQLException
```

- Operation: It returns the statement object which created the ResultSet object.
- Exception: If ResultSet is already closed, it throws SQLException.

getString

```
String getString(int columnIndex) throws SQLException
```

- Operation: It gets the columnIndex-th column data as string type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** . If getString() is performed for BINARY, VARBINARY, LONG VARBINARY, the string of hex code is returned.
- Exception: If ResultSet is already closed or the columnIndex exceeds the range, it throws SQLException.

```
String getString(String columnLabel) throws SQLException
```

- Operation: It gets the column data whose name is columnLabel as string. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** . If getString() is performed for BINARY, VARBINARY, LONG VARBINARY, it returns the string of hex code.

- Exception: If ResultSet is already closed or the corresponding columnLabel does not exist, it throws SQLException.

getTime

```
Time getTime(int columnIndex) throws SQLException
```

- Operation: It gets the columnIndex-th column data as time type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** . When creating the time object, local timezone is used.
- Exception: If ResultSet is already closed or the columnIndex exceeds the range or the type can not be converted to time, it throws SQLException.

```
Time getTime(int columnIndex, Calendar cal) throws SQLException
```

- Operation: It gets the columnIndex-th column data as time type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** . When creating the time object, local timezone of cal is used.
- Exception: If ResultSet is already closed or the columnIndex exceeds the range or the type can not be converted to time, it throws SQLException.

```
Time getTime(String columnLabel) throws SQLException
```

- Operation: It gets the column data whose name is columnLabel as time type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** . When creating the time object, the local timezone is used.
- Exception: If ResultSet is already closed or the corresponding columnLabel does not exist or the type can not be converted to time, it throws SQLException.

```
Time getTime(String columnLabel, Calendar cal) throws SQLException
```

- Operation: It gets the column data whose name is columnLabel as time type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** . When creating the time object, the local timezone is used.
- Exception: If ResultSet is already closed or the corresponding columnLabel does not exist or the type can not be converted to time, it throws SQLException.

getTimestamp

```
Timestamp getTimestamp(int columnIndex) throws SQLException
```

- Operation: It gets the columnIndex-th column data as timestamp type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type -**

1 . When creating the timestamp object, local timezone is used.
- Exception: If ResultSet is already closed or the columnIndex exceeds the range or the type can not be converted to timestamp, it throws SQLException.

```
Timestamp getTimestamp(int columnIndex, Calendar cal) throws SQLException
```

- Operation: It gets the columnIndex-th column data as timestamp type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** . When creating the timestamp object, local timezone of cal is used.
- Exception: If ResultSet is already closed or the columnIndex exceeds the range or the type can not be converted to timestamp, it throws SQLException.

```
Timestamp getTimestamp(String columnLabel) throws SQLException
```

- Operation: It gets the column data whose name is columnLabel as timestamp type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** . When creating the timestamp object, the local timezone is used.
- Exception: If ResultSet is already closed or the corresponding columnLabel does not exist or the type can not be converted to timestamp, it throws SQLException.

```
Timestamp getTimestamp(String columnLabel, Calendar cal) throws SQLException
```

- Operation: It gets the column data whose name is columnLabel as timestamp type. For more information about GOLDILOCKS type-specific support, refer to **Whether supporting getter method for GOLDILOCKS type - 1** . When creating the timestamp object, the local timezone of cal is used.
- Exception: If ResultSet is already closed or the corresponding columnLabel does not exist or the type can not be converted to timestamp, it throws SQLException.

getType

```
int getType() throws SQLException
```

- Operation: It returns the current ResultSet type. It returns one of ResultSet.TYPE_FORWARD_ONLY, ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.TYPE_SCROLL_SENSITIVE.
- Exception: If ResultSet is already closed, it throws SQLException.

getUnicodeStream

```
InputStream getUnicodeStream(int columnIndex) throws SQLException
```

- Operation: It is a deprecated method. It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

```
InputStream getUnicodeStream(String columnLabel) throws SQLException
```

- Operation: It is a deprecated method. It is not implemented.
- Exception: It always throws SQLFeatureNotSupportedException.

getURL

```
URL getURL(int columnIndex) throws SQLException
```

- Operation: URL type is not supported.
- Exception: It always throws SQLFeatureNotSupportedException.

```
URL getURL(String columnLabel) throws SQLException
```

- Operation: URL type is not supported.
- Exception: It always throws SQLFeatureNotSupportedException.

getWarnings

```
SQLWarning getWarnings() throws SQLException
```

- Operation: It returns a list of SQLWarning accumulated on the object so far. If it gets a warning from the server, it creates SQLWarning. If clearWarning is not performed, it continues to be accumulated. I f a warning does not occur, null is returned.
- Exception: It does not occur.

insertRow

```
void insertRow() throws SQLException
```

- Operation: Cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSe t.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws S QLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

isAfterLast

```
boolean isAfterLast() throws SQLException
```

- Operation: It queries whether the current cursor position is at *after last*. If so, it returns true. Otherwis

e, it returns false.
- Exception: If ResultSet is already closed, it throws SQLException.

isBeforeFirst

```
boolean isBeforeFirst() throws SQLException
```

- Operation: It queries whether the current cursor position is at *before first*. If so, it returns true. Other wise, it returns false.
- Exception: If ResultSet is already closed, it throws SQLException.

isClosed

```
boolean isClosed() throws SQLException
```

- Operation: It queries whether the current ResultSet is closed. If closed, it returns true. Otherwise, it re turns false. Even if the user does not call the close, ResultSet can be closed by closing the cursor of th e server. It is when, for example, the statement which created the ResultSet is closed, or the transacti on is committed when the holdability is ResultSet.CLOSE_CURSOR_AT_COMMIT mode, or an error o ccurs from the server during fetching.
- Exception: It does not occur.

isFirst

```
boolean isFirst() throws SQLException
```

- Operation: It queries whether the current cursor position is at first (the first row). If so, it returns true. Otherwise, it returns false.
- Exception: If ResultSet is already closed, it throws SQLException.

isLast

```
boolean isLast() throws SQLException
```

- Operation: It queries whether the current cursor position is at last (the last row). If so, it returns true. Otherwise, it returns false.
- Exception: If ResultSet is already closed, it throws SQLException.

last

```
boolean last() throws SQLException
```

- Operation: The row cursor is positioned at last (the last row). If the row cache is the last row set (It is a part of the entire result set), only the cursor position is changed. Otherwise, the row cursor is positioned at last after the last row set is fetched (row from last-n+1 to the last row, n is the number of rows fetched from the server). If the row exists, it returns true. Otherwise, it returns false.
- Exception: If ResultSet is already closed or the ResultSet type is ResultSet.TYPE_FORWARD_ONLY or an error occurs from the server when fetching, it throws an exception.

moveToCurrentRow

```
void moveToCurrentRow() throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

moveToInsertRow

```
void moveToInsertRow() throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

next

```
boolean next() throws SQLException
```

- Operation: The row cursor is positioned at the next to the current row. If the current row is the last row of the row cache, the next row cache is fetched from the server. If the row exists, it returns true. Otherwise, it returns false.
- Exception: If ResultSet is already closed or an error occurs from the server when fetching, it throws an exception.

previous

```
boolean previous() throws SQLException
```

- Operation: The row cursor is positioned at the row before the current position. If the current row is t he first row of the row cache, the previous row cache(n rows from x-n to x-1, x is the current row ind ex) is fetched from the server. If the row exists, it returns true. Otherwise, it returns false.
- Exception: If ResultSet is already closed or the ResultSet type is ResultSet.TYPE_FORWARD_ONLY or a n error occurs from the server during fetching, it throws an exception.

refreshRow

```
void refreshRow() throws SQLException
```

- Operation: If ResultSet type is ResultSet.SCROLL_SENSITIVE, the current row cache is fetched from th e server. If any row is changed (by the same transaction or other transactions), it is reflected. If Result Set type is ResultSet.Scroll_INSENSITIVE, any operation is not performed.
- Exception: If ResultSet is already closed or the ResultSet type is ResultSet.TYPE_FORWARD_ONLY or a n error occurs from the server during fetching, it throws an exception.

relative

```
boolean relative(int rows) throws SQLException
```

- Operation: It moves the row cursor from the current cursor position to the position apart as many as t he number of rows. If it can be moved within the current row cache, only the cursor position is chang ed. Otherwise, the cursor is moved after the row cache is fetched from the server. If the position to b e moved is backward from the current position (next direction), the row cache is fetched from rows t o rows+n-1 (in favor of next). If the position to be moved is forward from the current position(previo us direction), the row cache is fetched from rows-n+1 to rows (in favor of previous).
- Exception: If ResultSet is already closed or the ResultSet type is ResultSet.TYPE_FORWARD_ONLY or a n error occurs from the server when fetching, it throws an exception.

rowDeleted

```
boolean rowDeleted() throws SQLException
```

- Operation: It queries whether the row of the current cursor position is deleted (by the same transacti on or other transactions). If deleted, it returns true. Otherwise, it returns false.
- Exception: If ResultSet is already closed or the ResultSet type is ResultSet.TYPE_FORWARD_ONLY, it t hrows an exception.

rowInserted

```
boolean rowInserted() throws SQLException
```

- Operation: It queries whether the row of the current cursor position is inserted (by the same transacti on or other transactions). If inserted, it returns true. Otherwise, it returns false.
- Exception: If ResultSet is already closed or the ResultSet type is ResultSet.TYPE_FORWARD_ONLY, it t hrows an exception.

rowUpdated

```
boolean rowUpdated() throws SQLException
```

- Operation: It queries whether the row of the current cursor position is updated (by the same transacti on or other transactions). If updated, it returns true. Otherwise, it returns false.
- Exception: If ResultSet is already closed or the ResultSet type is ResultSet.TYPE_FORWARD_ONLY, it t hrows an exception.

setFetchDirection

```
void setFetchDirection(int direction) throws SQLException
```

- Operation: The backward fetch is not supported by the server. Therefore, only ResultSet.FETCH_FOR WARD is available. SQLWarning occurs if other values are inserted.
- Exception: If ResultSet is already closed or the argument does not have the defined value, it throws a n exception.

setFetchSize

```
void setFetchSize(int rows) throws SQLException
```

- Operation: It specifies the number of rows fetched from the server at once. 0 refers that the server de termines it. If it is 0, it refers to the number of rows of which a communication packet can include at once for the forward only cursor. It is specified as 100 for the scrollable cursor.
- Exception: If ResultSet is already closed, it throws an exception.

updateArray

```
void updateArray(int columnIndex, Array x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateArray(String columnLabel, Array x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

updateAsciiStream

```
void updateAsciiStream(int columnIndex, InputStream x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateAsciiStream(int columnIndex, InputStream x, int length) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateAsciiStream(int columnIndex, InputStream x, long length) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateAsciiStream(String columnLabel, InputStream x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateAsciiStream(String columnLabel, InputStream x, int length) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateAsciiStream(String columnLabel, InputStream x, long length) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

updateBigDecimal

```
void updateBigDecimal(int columnIndex, BigDecimal x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateBigDecimal(String columnLabel, BigDecimal x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

updateBinaryStream

```
void updateBinaryStream(int columnIndex, InputStream x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateBinaryStream(int columnIndex, InputStream x, int length) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is Resu

ltSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it thro ws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateBinaryStream(int columnIndex, InputStream x, long length) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is Resu ltSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it thro ws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateBinaryStream(String columnLabel, InputStream x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is Resu ltSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it thro ws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateBinaryStream(String columnLabel, InputStream x, int length) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is Resu ltSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it thro ws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateBinaryStream(String columnLabel, InputStream x, long length) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is Resu ltSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it thro ws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

updateBlob

```
void updateBlob(int columnIndex, Blob x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is Resu ltSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it thro ws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateBlob(int columnIndex, InputStream inputStream) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateBlob(int columnIndex, InputStream inputStream, long length) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateBlob(String columnLabel, Blob x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateBlob(String columnLabel, InputStream inputStream) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateBlob(String columnLabel, InputStream inputStream, long length) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

updateBoolean

```
void updateBoolean(int columnIndex, boolean x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateBoolean(String columnLabel, boolean x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

updateByte

```
void updateByte(int columnIndex, byte x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateByte(String columnLabel, byte x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

updateBytes

```
void updateBytes(int columnIndex, byte[] x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateBytes(String columnLabel, byte[] x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

updateCharacterStream

```
void updateCharacterStream(int columnIndex, Reader x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateCharacterStream(int columnIndex, Reader x, int length) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateCharacterStream(int columnIndex, Reader x, long length) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateCharacterStream(String columnLabel, Reader reader) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateCharacterStream(String columnLabel, Reader reader, int length) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateCharacterStream(String columnLabel, Reader reader, long length) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

updateClob

```
void updateClob(int columnIndex, Clob x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateClob(int columnIndex, Reader reader) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateClob(int columnIndex, Reader reader, long length) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateClob(String columnLabel, Clob x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateClob(String columnLabel, Reader reader) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateClob(String columnLabel, Reader reader, long length) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

updateDate

```
void updateDate(int columnIndex, Date x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateDate(String columnLabel, Date x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

updateDouble

```
void updateDouble(int columnIndex, double x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateDouble(String columnLabel, double x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

updateFloat

```
void updateFloat(int columnIndex, float x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateFloat(String columnLabel, float x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

updateInt

```
void updateInt(int columnIndex, int x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateInt(String columnLabel, int x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

updateLong

```
void updateLong(int columnIndex, long x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateLong(String columnLabel, long x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

updateNCharacterStream

```
void updateNCharacterStream(int columnIndex, Reader x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateNCharacterStream(int columnIndex, Reader x, long length) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateNCharacterStream(String columnLabel, Reader reader) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateNCharacterStream(String columnLabel, Reader reader, long length) throws
SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

updateNClob

```
void updateNClob(int columnIndex, NClob nClob) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateNClob(int columnIndex, Reader reader) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is Resu

ltSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it thro ws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateNClob(int columnIndex, Reader reader, long length) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is Resu ltSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it thro ws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateNClob(String columnLabel, NClob nClob) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is Resu ltSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it thro ws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateNClob(String columnLabel, Reader reader) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is Resu ltSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it thro ws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateNClob(String columnLabel, Reader reader, long length) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is Resu ltSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it thro ws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

updateNString

```
void updateNString(int columnIndex, String nString) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is Resu ltSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it thro ws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateNString(String columnLabel, String nString) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

updateNull

```
void updateNull(int columnIndex) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateNull(String columnLabel) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

updateObject

```
void updateObject(int columnIndex, Object x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateObject(int columnIndex, Object x, int scaleOrLength) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateObject(String columnLabel, Object x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.

- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateObject(String columnLabel, Object x, int scaleOrLength) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

updateRef

```
void updateRef(int columnIndex, Ref x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateRef(String columnLabel, Ref x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

updateRow

```
void updateRow() throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

updateRowId

```
void updateRowId(int columnIndex, RowId x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateRowId(String columnLabel, RowId x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

updateShort

```
void updateShort(int columnIndex, short x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateShort(String columnLabel, short x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

updateSQLXML

```
void updateSQLXML(int columnIndex, SQLXML xmlObject) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateSQLXML(String columnLabel, SQLXML xmlObject) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

updateString

```
void updateString(int columnIndex, String x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateString(String columnLabel, String x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

updateTime

```
void updateTime(int columnIndex, Time x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateTime(String columnLabel, Time x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

updateTimestamp

```
void updateTimestamp(int columnIndex, Timestamp x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

```
void updateTimestamp(String columnLabel, Timestamp x) throws SQLException
```

- Operation: The cursor update feature has not yet been implemented. If ResultSet concurrency is ResultSet.CONCUR_READ_ONLY, it throws SQLException. If it is ResultSet.CONCUR_UPDATABLE, it throws SQLFeatureNotSupportedException.
- Exception: If it is already closed, it throws SQLException. Otherwise, refer to the operation.

wasNull

```
boolean wasNull() throws SQLException
```

- Operation: It queries whether the column value which was read last is NULL. If it is NULL, it returns true. Otherwise, it returns false.
- Exception: If ResultSet is already closed or the column value never has been read, it throws SQLException.

isWrapperFor

```
boolean isWrapperFor(Class<?> iface)
```

- Operation: It queries whether this object is the class which implemented the iface interface. If it so, it returns true. Otherwise, it returns false. It does not determine the presence of the wrapper and it queries only whether the given argument class type is implemented because GOLDILOCKS ResultSet object is not the wrapper of any other class.
- Exception: It does not occur.

unwrap

```
<T> T unwrap(Class<T> iface)
```

- Operation: It eventually returns itself even when it is unwrapped because GOLDILOCKS ResultSet is not the wrapper of any other class. It returns after casting to that type. If iface is an argument of isWrapperFor() method and false is returned, then the method throws an exception.
- Exception: If iface is not this object type (when this object type is not implemented.), it throws SQLException.

ResultSetMetaData

getCatalogName

```
String getCatalogName(int column) throws SQLException
```

- Operation: It returns the catalog name of the column.
- Exception: If the column value is wrong, it throws SQLException.

getColumnClassName

```
String getColumnClassName(int column) throws SQLException
```

- Operation: It returns the name of Java class which is the most appropriate to the column type. It is specified such as java.math.BigDecimal, and it refers to getName() method in Java. For binary type, it refer to byte[].class.getName(), so it can be specified such as '[B'.
- Exception: If the column value is wrong, it throws SQLException.

getColumnCount

```
int getColumnCount() throws SQLException
```

- Operation: It returns the number of columns that the ResultSet has.
- Exception: It does not occur.

getColumnDisplaySize

```
int getColumnDisplaySize(int column) throws SQLException
```

- Operation: It returns the maximum width when the column value is displayed.
- Exception: If the column value is wrong, it throws SQLException.

getColumnLabel

```
String getColumnLabel(int column) throws SQLException
```

- Operation: It gets the label of the column. For example, the label is "C1 + 1" and the name is "" for the query statement such as "select C1 + 1 from t1".
- Exception: If the column value is wrong, it throws SQLException.

getColumnName

```
String getColumnName(int column) throws SQLException
```

- Operation: It returns the alias name of the column. The original column name, not the alias name is defined to be returned in JDBC specification. However, it is recommended to use the alias name because some view names are meaningless or complicated. For example, both the name and label are "C2" for the query statement such as "select C1 as C2 from t1". For more information about the case of when name and label are different, refer to **getColumnLabel**.
- Exception: If the column value is wrong, it throws SQLException.

getColumnType

```
int getColumnType(int column) throws SQLException
```

- Operation: It returns the type of the column. The return value is defined in types. Types.OTHERS is returned for GOLDILOCKS interval family types. The constant values of the corresponding types are returned for the other types.
- Exception: If the column value is wrong, it throws SQLException.

getColumnTypeName

```
String getColumnTypeName(int column) throws SQLException
```

- Operation: It returns GOLDILOCKS column type name of the column.
- Exception: If the column value is wrong, it throws SQLException.

getPrecision

```
int getPrecision(int column) throws SQLException
```

- Operation: It returns the precision of the column. It returns 0 for the type without any precision.
- Exception: If the column value is wrong, it throws SQLException.

getScale

```
int getScale(int column) throws SQLException
```

- Operation: It returns the scale of the column. It returns 0 for the type without any scale.
- Exception: If the column value is wrong, it throws SQLException.

getSchemaName

```
String getSchemaName(int column) throws SQLException
```

- Operation: It returns the schema name of the column.
- Exception: If the column value is wrong, it throws SQLException.

getTableName

```
String getTableName(int column) throws SQLException
```

- Operation: It returns the table name of the column.
- Exception: If the column value is wrong, it throws SQLException.

isAutoIncrement

```
boolean isAutoIncrement(int column) throws SQLException
```

- Operation: It returns whether it is the column which is automatically given the unique value. If so, it r eturns true. Otherwise, it returns false.
- Exception: If the column value is wrong, it throws SQLException.

isCaseSensitive

```
boolean isCaseSensitive(int column) throws SQLException
```

- Operation: It returns whether the column is case sensitive. If it is case sensitive, it returns true. Other wise, it returns false.
- Exception: If the column value is wrong, it throws SQLException.

isCurrency

```
boolean isCurrency(int column) throws SQLException
```

- Operation: It always returns false because the currency of the column can not be determined by the s erver.
- Exception: If the column value is wrong, it throws SQLException.

isDefinitelyWritable

```
boolean isDefinitelyWritable(int column) throws SQLException
```

- Operation: It returns whether the column is updatable. GOLDILOCKS does not support the definitely writable. It always returns the value as same as isUpdatable().
- Exception: If the column value is wrong, it throws SQLException.

isNullable

```
int isNullable(int column) throws SQLException
```

- Operation: It returns whether the column has nullable. It returns either columnNullable or columnNo Nulls.
- Exception: If the column value is wrong, it throws SQLException.

isReadOnly

```
boolean isReadOnly(int column) throws SQLException
```

- Operation: It returns whether the column is read only. It always returns the opposite value of isUpdat able().
- Exception: If the column value is wrong, it throws SQLException.

isSearchable

```
boolean isSearchable(int column) throws SQLException
```

- Operation: It returns whether the column can be used in the conditional clause. It always returns true because all target columns in GOLDILOCKS can be used in the conditional clause.
- Exception: If the column value is wrong, it throws SQLException.

isSigned

```
boolean isSigned(int column) throws SQLException
```

- Operation: It returns whether the column has a sign. If it has a sign, it returns true. Otherwise, it retur ns false.
- Exception: If the column value is wrong, it throws SQLException.

isWritable

```
boolean isWritable(int column) throws SQLException
```

- Operation: It returns whether the column is updatable.
- Exception: If the column value is wrong, it throws SQLException.

isWrapperFor

```
boolean isWrapperFor(Class<?> iface)
```

- Operation: It queries whether the object is a class implementing the iface interface. If so, it returns tr ue. Otherwise, it returns false. It does not determine the presence of the wrapper and it queries only whether the given argument class type is implemented because GOLDILOCKS ResultSetMetaData obj ect is not the wrapper of any other class.
- Exception: It does not occur.

unwrap

```
<T> T unwrap(Class<T> iface)
```

- Operation: It eventually returns itself, even when it is unwrapped because GOLDILOCKS ResultSetMet aData is not the wrapper of any other class. It returns after casting to that type. If iface is an argumen t of isWrapperFor() method and false is returned, the method throws an exception.
- Exception: If iface is not the type of the object (if this object returns the unimplemented type), it thro ws SQLException.

RowId

equals

```
boolean equals(Object obj) throws SQLException
```

- Operation: If RowId of this object is as same as RowId of obj, it returns true. Otherwise, it returns fals e.
- Exception: It does not occur.

getBytes

```
byte[] getBytes() throws SQLException
```

- Operation: It returns the byte array value of RowId.
- Exception: It does not occur.

hashCode

```
int hashCode() throws SQLException
```

- Operation: It returns the hash code value.
- Exception: It does not occur.

toString

```
String toString() throws SQLException
```

- Operation: It returns a base-64 string of RowId value.
- Exception: It does not occur.

RowSet

The class is not implemented

addRowSetListener

```
void addRowSetListener(RowSetListener listener) throws SQLException
```

clearParameters

```
void clearParameters() throws SQLException
```

execute

```
void execute() throws SQLException
```

getCommand

```
String getCommand() throws SQLException
```

getDataSourceName

```
String getDataSourceName() throws SQLException
```

getEscapeProcessing

```
boolean getEscapeProcessing() throws SQLException
```

getMaxFieldSize

```
int getMaxFieldSize() throws SQLException
```

getMaxRows

```
int getMaxRows() throws SQLException
```

getPassword

```
String getPassword() throws SQLException
```

getQueryTimeout

```
int getQueryTimeout() throws SQLException
```

getTransactionIsolation

```
int getTransactionIsolation() throws SQLException
```

getTypeMap

```
Map<String,Class<?>> getTypeMap() throws SQLException
```

getUrl

```
String getUrl() throws SQLException
```

getUsername

```
String getUsername() throws SQLException
```

isReadOnly

```
boolean isReadOnly() throws SQLException
```

removeRowSetListener

```
void removeRowSetListener(RowSetListener listener) throws SQLException
```

setArray

```
void setArray(int i, Array x) throws SQLException
```

setAsciiStream

```
void setAsciiStream(int parameterIndex, InputStream x) throws SQLException
```

```
void setAsciiStream(int parameterIndex, InputStream x, int length) throws SQLException
```

```
void setAsciiStream(String parameterName, InputStream x) throws SQLException
```

```
void setAsciiStream(String parameterName, InputStream x, int length) throws SQLException
```

setBigDecimal

```
void setBigDecimal(int parameterIndex, BigDecimal x) throws SQLException
```

```
void setBigDecimal(String parameterName, BigDecimal x) throws SQLException
```

setBinaryStream

```
void setBinaryStream(int parameterIndex, InputStream x) throws SQLException
```

```
void setBinaryStream(int parameterIndex, InputStream x, int length) throws SQLException
```

```
void setBinaryStream(String parameterName, InputStream x) throws SQLException
```

```
void setBinaryStream(String parameterName, InputStream x, int length) throws SQLException
```

setBlob

```
void setBlob(int i, Blob x) throws SQLException
```

```
void setBlob(int parameterIndex, InputStream inputStream) throws SQLException
```

```
void setBlob(int parameterIndex, InputStream inputStream, long length) throws SQLException
```

```
void setBlob(String parameterName, Blob x) throws SQLException
```

```
void setBlob(String parameterName, InputStream inputStream) throws SQLException
```

```
void setBlob(String parameterName, InputStream inputStream, long length) throws SQLException
```

setBoolean

```
void setBoolean(int parameterIndex, boolean x) throws SQLException
```

```
void setBoolean(String parameterName, boolean x) throws SQLException
```

setByte

```
void setByte(int parameterIndex, byte x) throws SQLException
```

```
void setByte(String parameterName, byte x) throws SQLException
```

setBytes

```
void setBytes(int parameterIndex, byte[] x) throws SQLException
```

```
void setBytes(String parameterName, byte[] x) throws SQLException
```

setCharacterStream

```
void setCharacterStream(int parameterIndex, Reader reader) throws SQLException
```

```
void setCharacterStream(int parameterIndex, Reader reader, int length) throws SQLException
```

```
void setCharacterStream(String parameterName, Reader reader) throws SQLException
```

```
void setCharacterStream(String parameterName, Reader reader, int length) throws SQLException
```

setClob

```
void setClob(int i, Clob x) throws SQLException
```

```
void setClob(int parameterIndex, Reader reader) throws SQLException
```

```
void setClob(int parameterIndex, Reader reader, long length) throws SQLException
```

```
void setClob(String parameterName, Clob x) throws SQLException
```

```
void setClob(String parameterName, Reader reader) throws SQLException
```

```
void setClob(String parameterName, Reader reader, long length) throws SQLException
```

setCommand

```
void setCommand(String cmd) throws SQLException
```

setConcurrency

```
void setConcurrency(int concurrency) throws SQLException
```

setDataSourceName

```
void setDataSourceName(String name) throws SQLException
```

setDate

```
void setDate(int parameterIndex, Date x) throws SQLException
```

```
void setDate(int parameterIndex, Date x, Calendar cal) throws SQLException
```

```
void setDate(String parameterName, Date x) throws SQLException
```

```
void setDate(String parameterName, Date x, Calendar cal) throws SQLException
```

setDouble

```
void setDouble(int parameterIndex, double x) throws SQLException
```

```
void setDouble(String parameterName, double x) throws SQLException
```

setEscapeProcessing

```
void setEscapeProcessing(boolean enable) throws SQLException
```

setFloat

```
void setFloat(int parameterIndex, float x) throws SQLException
```

```
void setFloat(String parameterName, float x) throws SQLException
```

setInt

```
void setInt(int parameterIndex, int x) throws SQLException
```

```
void setInt(String parameterName, int x) throws SQLException
```

setLong

```
void setLong(int parameterIndex, long x) throws SQLException
```

```
void setLong(String parameterName, long x) throws SQLException
```

setMaxFieldSize

```
void setMaxFieldSize(int max) throws SQLException
```

setMaxRows

```
void setMaxRows(int max) throws SQLException
```

setNCharacterStream

```
void setNCharacterStream(int parameterIndex, Reader value) throws SQLException
```

```
void setNCharacterStream(int parameterIndex, Reader value, long length) throws SQLException
```

```
void setNCharacterStream(String parameterName, Reader value) throws SQLException
```

```
void setNCharacterStream(String parameterName, Reader value, long length) throws SQLException
```

setNClob

```
void setNClob(int parameterIndex, NClob value) throws SQLException
```

```
void setNClob(int parameterIndex, Reader reader) throws SQLException
```

```
void setNClob(int parameterIndex, Reader reader, long length) throws SQLException
```

```
void setNClob(String parameterName, NClob value) throws SQLException
```

```
void setNClob(String parameterName, Reader reader) throws SQLException
```

```
void setNClob(String parameterName, Reader reader, long length) throws SQLException
```

setNString

```
void setNString(int parameterIndex, String value) throws SQLException
```

```
void setNString(String parameterName, String value) throws SQLException
```

setNull

```
void setNull(int parameterIndex, int sqlType) throws SQLException
```

```
void setNull(int paramIndex, int sqlType, String typeName) throws SQLException
```

```
void setNull(String parameterName, int sqlType) throws SQLException
```

```
void setNull(String parameterName, int sqlType, String typeName) throws SQLException
```

setObject

```
void setObject(int parameterIndex, Object x) throws SQLException
```

```
void setObject(int parameterIndex, Object x, int targetSqlType) throws SQLException
```

```
void setObject(int parameterIndex, Object x, int targetSqlType, int scaleOrLength) throws
SQLException
```

```
void setObject(String parameterName, Object x) throws SQLException
```

```
void setObject(String parameterName, Object x, int targetSqlType) throws SQLException
```

```
void setObject(String parameterName, Object x, int targetSqlType, int scale) throws
SQLException
```

setPassword

```
void setPassword(String password) throws SQLException
```

setQueryTimeout

```
void setQueryTimeout(int seconds) throws SQLException
```

setReadOnly

```
void setReadOnly(boolean value) throws SQLException
```

setRef

```
void setRef(int i, Ref x) throws SQLException
```

setRowId

```
void setRowId(int parameterIndex, RowId x) throws SQLException
```

```
void setRowId(String parameterName, RowId x) throws SQLException
```

setShort

```
void setShort(int parameterIndex, short x) throws SQLException
```

```
void setShort(String parameterName, short x) throws SQLException
```

setSQLXML

```
void setSQLXML(int parameterIndex, SQLXML xmlObject) throws SQLException
```

```
void setSQLXML(String parameterName, SQLXML xmlObject) throws SQLException
```

setString

```
void setString(int parameterIndex, String x) throws SQLException
```

```
void setString(String parameterName, String x) throws SQLException
```

setTime

```
void setTime(int parameterIndex, Time x) throws SQLException
```

```
void setTime(int parameterIndex, Time x, Calendar cal) throws SQLException
```

```
void setTime(String parameterName, Time x) throws SQLException
```

```
void setTime(String parameterName, Time x, Calendar cal) throws SQLException
```

setTimestamp

```
void setTimestamp(int parameterIndex, Timestamp x) throws SQLException
```

```
void setTimestamp(int parameterIndex, Timestamp x, Calendar cal) throws SQLException
```

```
void setTimestamp(String parameterName, Timestamp x) throws SQLException
```

```
void setTimestamp(String parameterName, Timestamp x, Calendar cal) throws SQLException
```

setTransactionIsolation

```
void setTransactionIsolation(int level) throws SQLException
```

setType

```
void setType(int type) throws SQLException
```

setTypeMap

```
void setTypeMap(Map<String,Class<?>> map) throws SQLException
```

setURL

```
void setURL(int parameterIndex, URL x) throws SQLException
```

setUrl

```
void setUrl(String url) throws SQLException
```

setUsername

```
void setUsername(String name) throws SQLException
```

RowSetMetaData

The class is not implemented

setAutoIncrement

```
void setAutoIncrement(int columnIndex, boolean property) throws SQLException
```

setCaseSensitive

```
void setCaseSensitive(int columnIndex, boolean property) throws SQLException
```

setCatalogName

```
void setCatalogName(int columnIndex, String catalogName) throws SQLException
```

setColumnCount

```
void setColumnCount(int columnCount) throws SQLException
```

setColumnDisplaySize

```
void setColumnDisplaySize(int columnIndex, int size) throws SQLException
```

setColumnLabel

```
void setColumnLabel(int columnIndex, String label) throws SQLException
```

setColumnName

```
void setColumnName(int columnIndex, String columnName) throws SQLException
```

setColumnType

```
void setColumnType(int columnIndex, int SQLType) throws SQLException
```

setColumnTypeName

```
void setColumnTypeName(int columnIndex, String typeName) throws SQLException
```

setCurrency

```
void setCurrency(int columnIndex, boolean property) throws SQLException
```

setNullable

```
void setNullable(int columnIndex, int property) throws SQLException
```

setPrecision

```
void setPrecision(int columnIndex, int precision) throws SQLException
```

setScale

```
void setScale(int columnIndex, int scale) throws SQLException
```

setSchemaName

```
void setSchemaName(int columnIndex, String schemaName) throws SQLException
```

setSearchable

```
void setSearchable(int columnIndex, boolean property) throws SQLException
```

setSigned

```
void setSigned(int columnIndex, boolean property) throws SQLException
```

setTableName

```
void setTableName(int columnIndex, String tableName) throws SQLException
```

Savepoint

getSavepointId

```
int getSavepointId() throws SQLException
```

- Operation: It returns the id value which is automatically assigned.
- Exception: It throws SQLException because it does not have an ID if the savepoint object is created by giving its name.

getSavepointName

```
String getSavepointName() throws SQLException
```

- Operation: It returns the name specified at the time of when a savepoint object is created.
- Exception: If the savepoint created with an automatic id value, it throws SQLException.

SQLData

The class is not implemented.

getSQLTypeName

```
String getSQLTypeName() throws SQLException
```

readSQL

```
void readSQL(SQLInput stream, String typeName) throws SQLException
```

writeSQL

```
void writeSQL(SQLOutput stream) throws SQLException
```

SQLXML

The class is not implemented.

free

```
void free() throws SQLException
```

getBinaryStream

```
InputStream getBinaryStream() throws SQLException
```

getCharacterStream

```
Reader getCharacterStream() throws SQLException
```

getSource

```
<T extends Source> T getSource(Class<T> sourceClass) throws SQLException
```

getString

```
String getString() throws SQLException
```

setBinaryStream

```
OutputStream setBinaryStream() throws SQLException
```

setCharacterStream

```
Writer setCharacterStream() throws SQLException
```

setResult

```
<T extends Result> T setResult(Class<T> resultClass) throws SQLException
```

setString

```
void setString(String value) throws SQLException
```

Statement

addBatch

```
void addBatch(String sql) throws SQLException
```

- Operation: The SQL statement is added to the batch job.
- Exception: It does not occur.

cancel

```
void cancel() throws SQLException
```

- Operation: It is not supported.
- Exception: It always throws SQLFeatureNotSupportedException.

clearBatch

```
void clearBatch() throws SQLException
```

- Operation: It clears all registered batch jobs. If registered batch job does not exist, any operation is no t performed.
- Exception: It does not occur.

clearWarnings

```
void clearWarnings() throws SQLException
```

- Operation: It clears all SQLWarning objects which is owned by the statement object.
- Exception: It does not occur.

close

```
void close() throws SQLException
```

- Operation: The current statement object is closed, and it is released if the statement related informati on assigned to the server exists. If the ResultSet created by the object exists, it is closed. The object is removed from the connection object which created the statement object.
- Exception: If an error occurs when statement information is released from the server, it throws an exc

eption.

execute

```
boolean execute(String sql) throws SQLException
```

- Operation: It executes the SQL statement. If the executed SQL statement has the ResultSet, it returns true. Otherwise, it returns false.
- Exception: If the statement is already closed or the batch job is registered or an error occurs when ex ecuting from the server, it throws an exception.

```
boolean execute(String sql, int autoGeneratedKeys) throws SQLException
```

- Operation: It executes the SQL statement. If autoGeneratedKeys is Statement.RETURN_GENERATED_ KEYS and sql is INSERT statement, then it can get ResultSet searching for the auto generated key. Ot herwise, it it as same as execute(String sql).
- Exception: If the statement is already closed or the batch job is registered or an error occurs when ex ecuting from the server, it throws an exception.

```
boolean execute(String sql, int[] columnIndexes) throws SQLException
```

- Operation: It executes the SQL statement. If columnIndexes is not null and sql is INSERT statement, th en it can get ResultSet searching for the designated auto generated key with the given index. Otherw ise, it it as same as execute(String sql).
- Exception: If the statement is already closed or the batch job is registered or an error occurs when ex ecuting from the server, it throws an exception.

```
boolean execute(String sql, String[] columnNames) throws SQLException
```

- Operation: It executes the SQL statement. If columnNames is not null and sql is INSERT statement, th en it can get ResultSet searching for the designated auto generated key with the given column name. Otherwise, it it as same as execute(String sql).
- Exception: If the statement is already closed or the batch job is registered or an error occurs when ex ecuting from the server, it throws an exception.

executeBatch

```
int[] executeBatch() throws SQLException
```

- Operation: It executes the registered batch job in turn. The communication with the server occurs per each batch job. It returns the array of the number of rows in which the update reflected after executi ng each batch job.
- Exception: If the statement is already closed or any batch job is not registered or an error occurs whe

n executing from the server, it throws an exception.

It does not have definite advantage over the normal execute() because the batch jobs are not tr ansmitted and executed at once. Use a batch execution of the PreparedStatement for fast proce ssing.

executeQuery

```
ResultSet executeQuery(String sql) throws SQLException
```

- Operation: It executes the given SQL statement and gets part of results and creates the ResultSet.
- Exception: If the statement is already closed or the batch job is registered or an error occurs when ex ecuting on the server or the SQL statement is not the select statement, it throws an exception.

Its operation is a bit different from execute(). The communicate with the server occurs twice if g etResultSet() is performed for the same SQL statement after performing execute(). The executio n command is performed when performing execute(), and the fetch related command is perfor med when performing getResultSet(). On the other hand, executeQuery() assumes that the SQL statement is the select statement, and it executes all with a single communication until fetch.

executeUpdate

```
int executeUpdate(String sql) throws SQLException
```

- Operation: It executes the SQL statement. It returns the number of rows updated by the execution.
- Exception: If the statement is already closed or the batch job is registered or an error occurs when ex ecuting on the server or the SQL statement is the select statement, it throws an exception.

```
int executeUpdate(String sql, int autoGeneratedKeys) throws SQLException
```

- Operation: It executes the SQL statement. It returns the number of rows updated by the execution. If autoGeneratedKeys is Statement.RETURN_GENERATED_KEYS and sql is INSERT statement, then it ca n get ResultSet searching for the auto generated key. Otherwise, it it as same as executeUpdate(Strin g sql).
- Exception: If the statement is already closed or the batch job is registered or an error occurs when ex ecuting on the server or the SQL statement is the select statement, it throws an exception.

```
int executeUpdate(String sql, int[] columnIndexes) throws SQLException
```

- Operation: It executes the SQL statement. It returns the number of rows updated by the execution. If columnIndexes is not null and sql is INSERT statement, then it can get ResultSet searching for the desi gnated auto generated key with the given index. Otherwise, it it as same as executeUpdate(String sql) .
- Exception: If the statement is already closed or the batch job is registered or an error occurs when ex ecuting on the server or the SQL statement is the select statement, it throws an exception.

```
int executeUpdate(String sql, String[] columnNames) throws SQLException
```

- Operation: It executes the SQL statement. It returns the number of rows updated by the execution. If columnNames is not null and sql is INSERT statement, then it can get ResultSet searching for the desi gnated auto generated key with the given column name. Otherwise, it it as same as executeUpdate(S tring sql).
- Exception: If the statement is already closed or the batch job is registered or an error occurs when ex ecuting from the server or the SQL statement is the select statement, it throws an exception.

getConnection

```
Connection getConnection() throws SQLException
```

- Operation: It returns the connection object which created the object. If statement object is created w ith logical connection via the PooledConnection, the user gets the logical connection, not the physica l connection via the method.
- Exception: If the statement is already closed, it throws an exception.

getExplainPlan

```
String getExplainPlan() throws SQLException
```

- Operation: It is non-standard method, and it is the unique method of GoldilocksStatement. It gets th e generated plan text. It should set for generating the plan text via setExplainPlanOption to use the method. For more information about the detailed usage, refer to **Viewing Plan Text**.
- Exception: If the statement is already closed or an error occurs on the server, it throws an exception.

getExplainPlanOption

```
int getExplainPlanOption() throws SQLException
```

- Operation: It is non-standard method, and it is the unique method of GoldilocksStatement. It gets th e option for the generating the plan text which is currently set. The return value is one of the followin gs and the default value is GoldilocksStatement.EXPLAIN_PLAN_OPTION_OFF.
  - GoldilocksStatement.EXPLAIN_PLAN_OPTION_OFF

    - GoldilocksStatement.EXPLAIN_PLAN_OPTION_ON
    - GoldilocksStatement.EXPLAIN_PLAN_OPTION_ON_VERBOSE
    - GoldilocksStatement.EXPLAIN_PLAN_OPTION_ONLY
- Exception: It does not occur.

getFetchDirection

```
int getFetchDirection() throws SQLException
```

- Operation: It always returns ResultSet.FETCH_FORWARD.
- Exception: If the statement is already closed, it throws an exception.

getFetchSize

```
int getFetchSize() throws SQLException
```

- Operation: It returns the default fetch size of ResultSet which is got from the statement object. The default value is 0, and 0 refers that the server determines the number of fetched rows. For more information, refer to **getFetchSize** of ResultSet.
- Exception: If the statement is already closed, it throws an exception.

getGeneratedKeys

```
ResultSet getGeneratedKeys() throws SQLException
```

- Operation: It returns ResultSet which can search the auto generated key of the result executed by this statement object. If the statement object has not created the key, then it returns the closed ResultSet object.
- Exception: If the statement is already closed or an error occurs on the server, it throws an exception.

getMaxFieldSize

```
int getMaxFieldSize() throws SQLException
```

- Operation: It returns the max field size. The value limits the maximum length of the column. If the column value is bigger than this length when fetching, the rest of the data is truncated. The default value is 0, and 0 refers that the maximum length is infinite.
- Exception: If the statement is already closed, it throws an exception.

getMaxRows

```
int getMaxRows() throws SQLException
```

- Operation: It returns the max rows. The max rows refer to the maximum number of rows of ResultSe t which is got from the statement. The rows more than the maximum number of rows are ignored. T he default value is 0, and 0 means infinity.
- Exception: If the statement is already closed, it throws an exception.

getMoreResults

```
boolean getMoreResults() throws SQLException
```

- Operation: It moves to the next result of the statement. The current ResultSet is closed.
- Exception: If the statement is already closed, it throws an exception.

```
boolean getMoreResults(int current) throws SQLException
```

- Operation: It moves to the next result of the statement. The current ResultSet is processed according to current value. It supports Statement.CLOSE_CURRENT_RESULT and Statement.CLOSE_ALL_RESUL TS only.
- Exception: If the statement is already closed, it throws an exception.

getQueryTimeout

```
int getQueryTimeout() throws SQLException
```

- Operation: It gets the value of the query timeout. The value is the timeout value which the sever appli es at execution. The execution is canceled and the user gets the error related to timeout if the executi on time exceeds the time. The unit is seconds and it applies the default value of the session if the use r does not specifically set it. The default value of the session is 0 if it is not set with the property, and i t refers to the infinite wait.
- Exception: If the statement is already closed, it throws an exception.

getResultSet

```
ResultSet getResultSet() throws SQLException
```

- Operation: It performs the fetch for the currently execution, and it gets part of fetched results, and cr eates and returns the ResultSet. JDBC specification defines to call this method once per the execution, but it is implemented to return the same object for several calls of the method.

- Exception: If the statement is already closed, it throws an exception. If an error occurs on the server when fetching, it throws an exception.

getResultSetConcurrency

```
int getResultSetConcurrency() throws SQLException
```

- Operation: It returns the ResultSet concurrency. The value determines the concurrency of ResultSet g enerated from the object. The default value is ResultSet.CONCUR_READ_ONLY. The updatable curso r is not yet supported.
- Exception: If the statement is already closed, it throws an exception.

getResultSetHoldability

```
int getResultSetHoldability() throws SQLException
```

- Operation: It returns the ResultSet holdability. The value determines the holdability of ResultSet gener ated from the object. The default value is ResultSet.HOLD_CURSORS_OVER_COMMIT.
- Exception: If the statement is already closed, it throws an exception.

getResultSetType

```
int getResultSetType() throws SQLException
```

- Operation: It returns the ResultSet type. The value determines the type of ResultSet generated from t he object. The default value is ResultSet.TYPE_FORWARD_ONLY.
- Exception: If the statement is already closed, it throws an exception.

getUpdateCount

```
int getUpdateCount() throws SQLException
```

- Operation: It returns the number of rows in which the update for the last execution is reflected. If the last executed SQL statement is not UPDATE statement nor is INSERT statement, it returns -1.
- Exception: It does not occur.

getUpdateRowCount

```
long getUpdateRowCount() throws SQLException
```

- Operation: It is as same as getUpdateCount, but the returned type is long. It is non-standard method, and the type casting to GoldilocksStatement should be performed to use it.
- Exception: It does not occur.

getWarnings

```
SQLWarning getWarnings() throws SQLException
```

- Operation: It returns SQLWarning accumulated on the object. If it does not exist, it returns null.
- Exception: It does not occur.

isClosed

```
boolean isClosed() throws SQLException
```

- Operation: It returns whether the statement is closed. If it is closed, it returns true. Otherwise, it retur ns false. It can be closed not only by the explicit call of close() but also by the server or the connection object.
- Exception: It does not occur.

isPoolable

```
boolean isPoolable() throws SQLException
```

- Operation: It returns whether the statement pooling is allowed for this object.
- Exception: It does not occur.

setCursorName

```
void setCursorName(String name) throws SQLException
```

- Operation: It sets a name for the cursor created by the currently executed statement.
- Exception: If an error occurs on the server when setting the cursor name, it throws an exception.

setEscapeProcessing

```
void setEscapeProcessing(boolean enable) throws SQLException
```

- Operation: JDBC can not ban the feature because the escape processing of the SQL statement is perf ormed in the server parser. Any operation is not performed.
- Exception: If the statement is already closed, it throws an exception.

setExplainPlanOption

```
void setExplainPlanOption(int option) throws SQLException
```

- Operation: It is non standard method, but it is the GoldilocksStatement unique method. It specifies th e plan text generation options. The option should be set to one of the followings, and each meaning is as follows.
  - GoldilocksStatement.EXPLAIN_PLAN_OPTION_OFF: The plan text is not generated.
  - GoldilocksStatement.EXPLAIN_PLAN_OPTION_ON: The plan text is generated when executing.
  - GoldilocksStatement.EXPLAIN_PLAN_OPTION_ON_VERBOSE: Detailed plan text is generated wh en executing.
  - GoldilocksStatement.EXPLAIN_PLAN_OPTION_ONLY: The plan text is generated when executing, but the actual execution is not performed.
- Exception: When setting a value other than four values above, it throws an exception.

setFetchDirection

```
void setFetchDirection(int direction) throws SQLException
```

- Operation: It sets the fetch direction. If the direction is not ResultSet.FETCH_FORWARD, it throws an exception because GOLDILOCKS supports only the forward fetch.
- Exception: If the statement is already closed or the direction is not FETCH_FORWARD, it throws an ex ception.

setFetchSize

```
void setFetchSize(int rows) throws SQLException
```

- Operation: It sets the default fetch size of ResultSet which is got from the statement object. The defa ult value is 0, and 0 refers that the server determines the number of fetched rows. For more informati on, refer to **setFetchSize** of ResultSet.
- Exception: If the statement is already closed, it throws an exception.

setMaxFieldSize

```
void setMaxFieldSize(int max) throws SQLException
```

- Operation: It sets the max field size. The value limits the maximum length of the column. If the colum n value is bigger than this length when fetching, the rest of the data is truncated. The default value is 0, 0 refers that the maximum length is infinity. It is valid for CHAR, VARCHAR, LONG VARCHAR, BIN ARY, VARBINARY, LONG VARBINARY types.
- Exception: If the statement is already closed, it throws an exception.

setMaxRows

```
void setMaxRows(int max) throws SQLException
```

- Operation: It sets the max rows. The max rows refers to the maximum number of rows of ResultSet which is got from the statement. The rows more than the maximum number of rows are ignored. Th e default is 0, and 0 means infinity.
- Exception: If the statement is already closed, it throws an exception.

setPoolable

```
void setPoolable(boolean poolable) throws SQLException
```

- Operation: It sets whether to pool the statement.
- Exception: It does not occur.

setQueryTimeout

```
void setQueryTimeout(int seconds) throws SQLException
```

- Operation: It sets the value of query timeout. The value is the timeout value which the sever applies a t the execution, and the execution is canceled and the user gets the error related to timeout if the ex ecution time exceeds the time. The unit is seconds and it applies the default value of the session if th e user idoes not specifically set it. The default value of the session is 0 if it is not set with the property, and 0 refers to the infinite wait.
- Exception: If the statement is already closed, it throws an exception.

isWrapperFor

```
boolean isWrapperFor(Class<?> iface) throws SQLException
```

- Operation: It queries whether the object is a class which implements the iface interface. If so, it retur ns true. Otherwise, it returns false. It does not determine the presence of the wrapper but it only que ries only whether the given argument class type is implemented because GOLDILOCKS statement obj ect is not the wrapper of any other class.
- Exception: It does not occur.

unwrap

```
<T> T unwrap(Class<T> iface) throws SQLException
```

- Operation: It eventually returns itself, even when it is unwrapped because GOLDILOCKS statement is not the wrapper of any other class. It returns after casting to that type. If iface is an argument of isWr apperFor() method and false is returned, the method throws an exception.
- Exception: If iface is not the type of the object (when this object returns the unimplemented type), it throws SQLException.

Struct

The class is not implemented.

getAttributes

```
Object[] getAttributes() throws SQLException
```

getAttributes

```
Object[] getAttributes(Map<String,Class<?>> map) throws SQLException
```

getSQLTypeName

```
String getSQLTypeName() throws SQLException
```

XAConnection

getXAResource

```
XAResource getXAResource() throws SQLException
```

- Operation: It returns XAResource object which can perform XA command. When the method is called for several times, the same result is continuously returned.
- Exception: It does not occur.

XADataSource

getXAConnection

```
XAConnection getXAConnection() throws SQLException
```

- Operation: It creates XAConnection object and returns it. The information required for the connectio n should be set as the separate non-standard methods in advance.
- Exception: When connection to the server is failed, it throws SQLException.

```
XAConnection getXAConnection(String user, String password) throws SQLException
```

- Operation: It opens and returns a new XAConnection object with the username and password. Other information required for the connection should be set as the separate non-standard methods in adva nce.
- Exception: When connection to the server is failed, it throws SQLException.

XAResource

commit

```
void commit(Xid xid, boolean onePhase) throws XAException
```

- Operation: It performs the XA commit command for the global transaction xid. If onePhase is set to t rue, one phase commit is performed.
- Exception: If the execution result error occurs, it throws XAException.

end

```
void end(Xid xid, int flags) throws XAException
```

- Operation: It performs the XA end command for the global transaction xid. The flags may be one of T MSUCCESS, TMFAIL, or TMSUSPEND.
- Exception: If the execution result error occurs, it throws XAException.

forget

```
void forget(Xid xid) throws XAException
```

- Operation: It performs the XA forget command for the global transaction xid.
- Exception: If the execution result error occurs, it throws XAException.

getTransactionTimeout

```
int getTransactionTimeout() throws XAException
```

- Operation: GOLDILOCKS does not support the transaction timeout. It always return 0.
- Exception: It does not occur.

isSameRM

```
boolean isSameRM(XAResource xares) throws XAException
```

- Operation: It has the unique rmid when XAResource object is created. Whether it is the same XAReso urce object is determined with this rmid.
- Exception: It does not occur.

prepare

```
int prepare(Xid xid) throws XAException
```

- Operation: It performs the XA prepare command for the global transaction xid.
- Exception: If the execution result error occurs, it throws XAException.

recover

```
Xid[] recover(int flag) throws XAException
```

- Operation: It performs XA recover command with the given flag, and the array of the prepared trans action branches is returned. The flag may be one of TMSTARTRSCAN, TMENDRSCAN, TMNOFLAGS.
- Exception: If the execution result error occurs, it throws XAException.

rollback

```
void rollback(Xid xid) throws XAException
```

- Operation: It performs the XA rollback command for the global transaction xid.
- Exception: If the execution result error occurs, it throws XAException.

setTransactionTimeout

```
boolean setTransactionTimeout(int seconds) throws XAException
```

- Operation: GOLDILOCKS does not support the transaction timeout. Any operation is not performed.
- Exception: It does not occur.

start

```
void start(Xid xid, int flags) throws XAException
```

- Operation: It starts the global transaction with the given flag. The flag may be one of TMNOFLAGS, T MJOIN, TMRESUME.
- Exception: If the execution result error occurs, it throws XAException.

GoldilocksInterval

To give value to a column of GOLDILOCKS by using a GoldilocksInterval object, refer to **Using Other Data Types**.

createIntervalYear

```
public static GoldilocksInterval createIntervalYear(int yearPrecision, boolean sign, int year)
throws SQLException
```

- Operation: It creates GoldilocksInterval object with the given year value. The yearPrecision refers to th e number of digits which the year can have. The value of year should be equal to or bigger than 0. If t he time is a positive number, sign is true. If it is a negative number, sign is false.
- Exception: If the given year value exceeds yearPrecision, an error occurs.

```
public static GoldilocksInterval createIntervalYear(int yearPrecision, String year) throws
SQLException
```

- Operation: It creates GoldilocksInterval object with the given year value. The yearPrecision refers to th e number of digits which the year can have. The value of year should be an integer equal to or bigger than 0.
- Exception: If the given year value exceeds yearPrecision, an error occurs.

createIntervalMonth

```
public static GoldilocksInterval createIntervalMonth(int monthPrecision, boolean sign, int
month) throws SQLException
```

- Operation: It creates GoldilocksInterval object with the given month value. The monthPrecision refers to the number of digits which the month can have. The value of month should be an integer equal to or bigger than 0. If the time is a positive number, sign is true. If it is a negative number, sign is false.
- Exception: If the given month value exceeds monthPrecision, an error occurs.

```
public static GoldilocksInterval createIntervalMonth(int monthPrecision, String month) throws
SQLException
```

- Operation: It creates GoldilocksInterval object with the given month value. The monthPrecision refers to the number of digits which the month can have. The value of month should be an integer equal to or bigger than 0.
- Exception: If the given month value exceeds monthPrecision, an error occurs.

createIntervalYearToMonth

```
public static GoldilocksInterval createIntervalYearToMonth(int yearPrecision, boolean sign,
int year, int month) throws SQLException
```

- Operation: It creates GoldilocksInterval object with the given year, month value. The yearPrecision refers to the number of digits which the year can have. The value of year, month should be equal to or bigger than 0. If the time is a positive number, sign is true. If it is a negative number, sign is false.
- Exception: If the given year value exceeds yearPrecision, an error occurs.

```
public static GoldilocksInterval createIntervalYearToMonth(int yearPrecision, String
yearToMonth) throws SQLException
```

- Operation: It creates GoldilocksInterval object with the given yearToMonth value. The yearPrecision refers to the number of digits which the year can have. yearToMonth should satisfy "yy-mm" pattern.
- Exception: If the given year value exceeds yearPrecision, an error occurs.

createIntervalDay

```
public static GoldilocksInterval createIntervalDay(int dayPrecision, boolean sign, int day)
throws SQLException
```

- Operation: It creates GoldilocksInterval object with the given day value. The dayPrecision refers to the number of digits which the day can have. The value of day should be equal to or bigger than 0. If the time is a positive number, sign is true. If it is a negative number, sign is false.
- Exception: If the given day value exceeds dayPrecision, an error occurs.

```
public static GoldilocksInterval createIntervalDay(int dayPrecision, String day) throws
SQLException
```

- Operation: It creates GoldilocksInterval object with the given day value. The dayPrecision refers to the number of digits which the day can have. The value of day should be an integer equal to or bigger than 0.
- Exception: If the given day value exceeds dayPrecision, an error occurs.

createIntervalHour

```
public static GoldilocksInterval createIntervalHour(int hourPrecision, boolean sign, int hour)
throws SQLException
```

- Operation: It creates GoldilocksInterval object with the given hour value. The hourPrecision refers to the number of digits which the hour can have. The value of hour should be equal to or bigger than 0.

If the time is a positive number, sign is true. If it is a negative number, sign is false.

- Exception: If the given hour value exceeds hourPrecision, an error occurs.

```
public static GoldilocksInterval createIntervalHour(int hourPrecision, String hour) throws
SQLException
```

- Operation: It creates GoldilocksInterval object with the given hour value. The hourPrecision refers to t he number of digits which the hour can have. The value of hour should be an integer equal to or big ger than 0.
- Exception: If the given hour value exceeds hourPrecision, an error occurs.

createIntervalMinute

```
public static GoldilocksInterval createIntervalMinute(int minutePrecision, boolean sign, int
minute) throws SQLException
```

- Operation: It creates GoldilocksInterval object with the given minute value. The minutePrecision refer s to the number of digits which the minute can have. The value of minute should be equal to or bigg er than 0. If the time is a positive number, sign is true. If it is a negative number, sign is false.
- Exception: If the given minute value exceeds minutePrecision, an error occurs.

```
public static GoldilocksInterval createIntervalMinute(int minutePrecision, String minute)
throws SQLException
```

- Operation: It creates GoldilocksInterval object with the given minute value. The minutePrecision refer s to the number of digits which the minute can have. The value of minute should be an integer equal to or bigger than 0.
- Exception: If the given minute value exceeds minutePrecision, an error occurs.

createIntervalSecond

```
public static GoldilocksInterval createIntervalSecond(int secondPrecision, int
fractionalPrecision, boolean sign, int second, int microsecond) throws SQLException
```

- Operation: It creates GoldilocksInterval object with the given second value. The secondPrecision refer s to the number of digits of second, and the fractionalPrecision refers to the number of digits of micr osecond. The value of second and microsecond should be equal to or bigger than 0. If the time is a p ositive number, sign is true. If it is a negative number, sign is false.
- Exception: If the given second value exceeds secondPrecision or the given microSecond value exceed s fractionalPrecision, an error occurs.

```
public static GoldilocksInterval createIntervalSecond(int secondPrecision, int
fractionalPrecision, boolean sign, int day, int hour, int minute, int second, int microsecond)
throws SQLException
```

- Operation: It creates GoldilocksInterval object with the given day, hour, minute, second, microsecond value. The secondPrecision refers to the number of digits of second when day, hour, minute, second i s converted to second. The fractionalPrecision refers to the number of digits of microsecond. The valu e of day, hour, minute, second, microsecond should be equal to or bigger than 0. If the time is a posi tive number, sign is true. If it is a negative number, sign is false.
- Exception: If the converted second value exceeds the secondPrecision or the microSecond value exce eds the fractionalPrecision, an error occurs.

```
public static GoldilocksInterval createIntervalSecond(int secondPrecision, int
fractionalPrecision, String second) throws SQLException
```

- Operation: It creates GoldilocksInterval object with the given second string. The secondPrecision refer s to the number of digits of second. The fractionalPrecision refers to the number of digits of microsec ond. The second string should be one of "dd hh:mm:ss.ffffff" or "dd hh:mm:ss", "dd hh:mm", "dd hh" , "hh:mm", "ss.ffffff", "ss" patterns.
- Exception: If the converted second value exceeds the secondPrecision or the microSecond value exce eds the fractionalPrecision, an error occurs. If the string does not conform to the prescribed format, a n error occurs.

createIntervalDayToHour

```
public static GoldilocksInterval createIntervalDayToHour(int dayPrecision, boolean sign, int
day, int hour) throws SQLException
```

- Operation: It creates GoldilocksInterval object with the given day, hour value. The dayPrecision refers to the number of digits of day. The value of day, hour should be equal to or bigger than 0. If the time is a positive number, sign is true. If it is a negative number, sign is false.
- Exception: If the day value exceeds dayPrecision, an error occurs.

```
public static GoldilocksInterval createIntervalDayToHour(int dayPrecision, String dayToHour)
throws SQLException
```

- Operation: It creates GoldilocksInterval object with the given dayToHour string. The dayPrecision refe rs to the number of digits of day. dayToHour should satisfy "dd hh" format. Or, if its format is "dd hh: mm:ss", all should be 0 except for dd, hh.
- Exception: If the day value exceeds dayPrecision or the given string does not satisfy the format, an err or occurs.

createIntervalDayToMinute

```
public static GoldilocksInterval createIntervalDayToMinute(int dayPrecision, boolean sign, int day, int hour, int minute) throws SQLException
```

- Operation: It creates GoldilocksInterval object with the given day, hour, minute value. The dayPrecision refers to the number of digits of day. The value of day, hour, minute should be equal to or bigger than 0. If the time is a positive number, sign is true. If it is a negative number, sign is false.
- Exception: If the day value exceeds dayPrecision, an error occurs.

```
public static GoldilocksInterval createIntervalDayToMinute(int dayPrecision, String dayToMinute) throws SQLException
```

- Operation: It creates GoldilocksInterval object with the given dayToMinute string. The dayPrecision refers to the number of digits of day. dayToMinute should satisfy "dd hh:mm" format. Or, if its format is "dd hh:mm:ss", all should be 0 except for dd, hh, mm.
- Exception: If the day value exceeds dayPrecision or the given string does not satisfy the format, an error occurs.

createIntervalDayToSecond

```
public static GoldilocksInterval createIntervalDayToSecond(int dayPrecision, int fractionalPrecision, boolean sign, int day, int hour, int minute, int second, int microsecond) throws SQLException
```

- Operation: It creates GoldilocksInterval object with the given day, hour, minute, second, microsecond value. The dayPrecision refers to the number of digits of day. The fractionalPrecision refers to the number of digits of microsecond. The value of day, hour, minute, second, microsecond should be equal to or bigger than 0. If the time is a positive number, sign is true. If it is a negative number, sign is false.
- Exception: If the day value exceeds dayPrecision or the microsecond value exceeds the fractionalPrecision, an error occurs.

```
public static GoldilocksInterval createIntervalDayToSecond(int dayPrecision, String dayToSecond) throws SQLException
```

- Operation: It creates GoldilocksInterval object with the given dayToSecond string. The dayPrecision refers to the number of digits of day. The fractionalPrecision refers to the number of digits of microsecond. dayToSecond should satisfy one of "dd hh:mm:ss.ffffff", "dd hh:mm:ss", "hh:mm:ss", "hh:mm" formats.
- Exception: If the day value or the converted day value exceeds dayPrecision or the microsecond value exceeds the fractionalPrecision or the given string does not satisfy the format, an error occurs.

createIntervalHourToMinute

```
public static GoldilocksInterval createIntervalHourToMinute(int hourPrecision, boolean sign,
int hour, int minute) throws SQLException
```

- Operation: It creates GoldilocksInterval object with the given hour, minute value. The hourPrecision r efers to the number of digits of hour. The value of hour, minute should be equal to or bigger than 0. If the time is a positive number, sign is true. If it is a negative number, sign is false.
- Exception: If the hour value exceeds hourPrecision, an error occurs.

```
public static GoldilocksInterval createIntervalHourToMinute(int hourPrecision, boolean sign,
int day, int hour, int minute) throws SQLException
```

- Operation: It creates GoldilocksInterval object with the given day, hour, minute value. The hourPrecisi on refers to the number of digits of hour. The value of day, hour, minute should be equal to or bigge r than 0. If the time is a positive number, sign is true. If it is a negative number, sign is false.
- Exception: If the hour value or the converted hour value exceeds hourPrecision, an error occurs.

```
public static GoldilocksInterval createIntervalHourToMinute(int hourPrecision, String
hourToMinute) throws SQLException
```

- Operation: It creates GoldilocksInterval object with the given hourToMinute string. The hourPrecision refers to the number of digits of hour. hourToMinute must satisfy "hh:mm" format. The value of ss or ffffff should be 0 for other formats.
- Exception: If the hour value or the converted hour value exceeds hourPrecision, an error occurs. If the given string does not satisfy the format, an error occurs.

createIntervalHourToSecond

```
public static GoldilocksInterval createIntervalHourToSecond(int hourPrecision, int
fractionalPrecision, boolean sign, int hour, int minute, int second, int microsecond) throws
SQLException
```

- Operation: It creates GoldilocksInterval object with the given hour, minute, second, microsecond valu e. The hourPrecision refers to the number of digits of hour. The fractionalPrecision refers to the num ber of digits of microsecond. The value of hour, minute, second, microsecond should be equal to or b igger than 0. If the time is a positive number, sign is true. If it is a negative number, sign is false.
- Exception: If the hour value exceeds hourPrecision or the microsecond value exceeds the fractionalPr ecision, an error occurs.

```
public static GoldilocksInterval createIntervalHourToSecond(int hourPrecision, int
fractionalPrecision, boolean sign, int day, int hour, int minute, int second, int microsecond)
throws SQLException
```

- Operation: It creates GoldilocksInterval object with the given day, hour, minute, second, microsecond value. The hourPrecision refers to the number of digits of hour. The fractionalPrecision refers to the number of digits of microsecond. The value of day, hour, minute, second, microsecond should be equal to or bigger than 0. If the time is a positive number, sign is true. If it is a negative number, sign is false.
- Exception: If the hour value or the converted hour value exceeds hourPrecision or the microsecond value exceeds the fractionalPrecision, an error occurs.

```
public static GoldilocksInterval createIntervalHourToSecond(int hourPrecision, int
fractionalPrecision, String hourToSecond) throws SQLException
```

- Operation: It creates GoldilocksInterval object with the given hourToSecond string. The hourPrecision refers to the number of digits of hour. The fractionalPrecision refers to the number of digits of microsecond. hourToSecond should satisfy "hh:mm:ss.ffffff", "hh:mm:ss", or "hh:mm" format. If dd is included, the value of converted hour should not exceed hourPrecision.
- Exception: If the hour value or the converted hour value exceeds hourPrecision or the microsecond value exceeds the fractionalPrecision, or the given string does not satisfy the format, an error occurs.

createIntervalMinuteToSecond

```
public static GoldilocksInterval createIntervalMinuteToSecond(int minutePrecision, int
fractionalPrecision, boolean sign, int minute, int second, int microsecond) throws
SQLException
```

- Operation: It creates GoldilocksInterval object with the given minute, second, microsecond value. The minutePrecision refers to the number of digits of minute. The fractionalPrecision refers to the number of digits of microsecond. The value of minute, second, microsecond should be equal to or bigger than 0. If the time is a positive number, sign is true. If it is a negative number, sign is false.
- Exception: If the minute value exceeds the minutePrecision or the microsecond value exceeds the fractionalPrecision, an error occurs.

```
public static GoldilocksInterval createIntervalMinuteToSecond(int minutePrecision, int
fractionalPrecision, boolean sign, int day, int hour, int minute, int second, int microsecond)
throws SQLException
```

- Operation: It creates GoldilocksInterval object with the given day, hour, minute, second, microsecond value. The minutePrecision refers to the number of digits of minute. The fractionalPrecision refers to the number of digits of microsecond. The value of day, hour, minute, second, microsecond should be equal to or bigger than 0. If the time is a positive number, sign is true. If it is a negative number, sign is false.
- Exception: If the minute value or the converted minute value exceeds minutePrecision or the microsecond value exceeds the fractionalPrecision, an error occurs.

```
public static GoldilocksInterval createIntervalMinuteToSecond(int minutePrecision, int
fractionalPrecision, String minuteToSecond) throws SQLException
```

- Operation: It creates GoldilocksInterval object with the given minuteToSecond string. The minutePrecision refers to the number of digits of minute. The fractionalPrecision refers to the number of digits of microsecond. minuteToSecond should satisfy "mm:ss.ffffff" or "mm:ss" format. If dd or hh is included, the value of the converted minute should not exceed minutePrecision.
- Exception: If the minute value or the converted minute value exceeds the minutePrecision or the microsecond value exceeds the fractionalPrecision or the given string does not satisfy the format, an error occurs.

getSign

```
public int getSign()
```

- Operation: If the time is a positive number, 1 is returned. If the time is a negative number, -1 is returned.
- Exception: It does not occur.

getYear

```
public int getYear()
```

- Operation: It returns the year value. Whether the interval object is a negative number is not returned through getYear().
- Exception: It does not occur.

getMonth

```
public int getMonth()
```

- Operation: It returns the month value. Whether the interval object is a negative number is not returned through getMonth().
- Exception: It does not occur.

getAccumulatedMonth

```
public int getAccumulatedMonth()
```

- Operation: It converts the value of year and month to the value of month, and returns the result.

- Exception: It does not occur.

getDay

```
public int getDay()
```

- Operation: It returns the day value. Whether the interval object is a negative number is not returned t hrough getDay().
- Exception: It does not occur.

getHour

```
public int getHour()
```

- Operation: It returns the hour value. Whether the interval object is a negative number is not returned through getHour().
- Exception: It does not occur.

getAccumulatedHour

```
public int getAccumulatedHour()
```

- Operation: It converts the value of day and hour to the value of hour, and returns the result.
- Exception: It does not occur.

getMinute

```
public int getMinute()
```

- Operation: It returns the minute value. Whether the interval object is a negative number is not return ed through getMinute().
- Exception: It does not occur.

getAccumulatedMinute

```
public int getAccumulatedMinute()
```

- Operation: It converts the value of day, hour and minute to the value of minute, and returns the resul t.
- Exception: It does not occur.

getSecond

```
public int getSecond()
```

- Operation: It returns the second value. Whether the interval object is a negative number is not return ed through getSecond().
- Exception: It does not occur.

getAccumulatedSecond

```
public int getAccumulatedSecond()
```

- Operation: It converts the value of day, hour, minute and second to the value of second, and returns the result.
- Exception: It does not occur.

getMicroSecond

```
public int getMicroSecond()
```

- Operation: It returns the microsecond value. Whether the interval object is a negative number is not r eturned through getMicroSecond().
- Exception: It does not occur.

getAccumulatedMicroSecond

```
public long getAccumulatedMicroSecond()
```

- Operation: It converts the value of day, hour, minute, second and microsecond to the value of micros econd, and returns the result.
- Exception: It does not occur.

getTypeName

```
public String getTypeName()
```

- Operation: The type name is returned.
- Exception: It does not occur.

getSqlType

```
public int getSqlType()
```

- Operation: The type of this object is returned as the type constant defined in GoldilocksTypes.
- Exception: It does not occur.

toString

```
public String toString()
```

- Operation: The interval value which is indicated by this object is returned as a string value.
- Exception: It does not occur.

GOLDILOCKS Type

Constant Definition

```
public static final int INTERVAL_YEAR;
public static final int INTERVAL_MONTH;
public static final int INTERVAL_DAY;
public static final int INTERVAL_HOUR;
public static final int INTERVAL_MINUTE;
public static final int INTERVAL_SECOND;
public static final int INTERVAL_YEAR_TO_MONTH;
public static final int INTERVAL_DAY_TO_HOUR;
public static final int INTERVAL_DAY_TO_MINUTE;
public static final int INTERVAL_DAY_TO_SECOND;
public static final int INTERVAL_HOUR_TO_MINUTE;
public static final int INTERVAL_HOUR_TO_SECOND;
public static final int INTERVAL_MINUTE_TO_SECOND;
public static final int TIME_WITH_TIME_ZONE;
public static final int TIMESTAMP_WITH_TIME_ZONE;
public static final int REF_CURSOR;
```

These constants are used as like the constants of java.sql.Types. In other words, they are used when speci fying the types in setObject of PreparedStatement or the types in getObject of ResultSet. These types are separately provided by GoldilocksTypes because they are not defined in the JDBC standard.

Type Conversion

The following tables describe how to convert types.

Table 32-3 SQL types → GOLDILOCKS types

SQL type	GOLDILOCKS type
GoldilocksTypes.INTERVAL_DAY	INTERVAL DAY
GoldilocksTypes.INTERVAL_DAY_TO_HOUR	INTERVAL DAY TO HOUR
GoldilocksTypes.INTERVAL_DAY_TO_MINUTE	INTERVAL DAY TO MINUTE
GoldilocksTypes.INTERVAL_DAY_TO_SECOND	INTERVAL DAY TO SECOND
GoldilocksTypes.INTERVAL_HOUR	INTERVAL HOUR
GoldilocksTypes.INTERVAL_HOUR_TO_MINUTE	INTERVAL HOUR TO MINUTE
GoldilocksTypes.INTERVAL_HOUR_TO_SECOND	INTERVAL HOUR TO SECOND

SQL type	GOLDILOCKS type
GoldilocksTypes.INTERVAL_MINUTE	INTERVAL MINUTE
GoldilocksTypes.INTERVAL_MINUTE_TO_SECOND	INTERVAL MINUTE TO SECOND
GoldilocksTypes.INTERVAL_MONTH	INTERVAL MONTH
GoldilocksTypes.INTERVAL_SECOND	INTERVAL SECOND
GoldilocksTypes.INTERVAL_YEAR	INTERVAL YEAR
GoldilocksTypes.INTERVAL_YEAR_TO_MONTH	INTERVAL YEAR TO MONTH
GoldilocksTypes.REF_CURSOR	REF CURSOR
GoldilocksTypes.TIME_WITH_TIME_ZONE	TIME WITH TIME ZONE
GoldilocksTypes.TIMESTAMP_WITH_TIME_ZONE	TIMESTAMP WITH TIME ZONE
Types.ARRAY	N/A
Types.BIGINT	NATIVE_BIGINT
Types.BINARY	BINARY(2000)
Types.BIT	BOOLEAN
Types.BLOB	LONG VARBINARY
Types.BOOLEAN	BOOLEAN
Types.CHAR	CHAR(2000)
Types.CLOB	LONG VARCHAR
Types.DATALINK	N/A
Types.DATE	DATE
Types.DECIMAL	DECIMAL
Types.DISTINCT	N/A
Types.DOUBLE	NATIVE_DOUBLE
Types.FLOAT	FLOAT
Types.INTEGER	NATIVE_INTEGER
Types.JAVA_OBJECT	N/A
Types.LONGNVARCHAR	N/A
Types.LONGVARBINARY	LONG VARBINARY
Types.LONGVARCHAR	LONG VARCHAR
Types.NCHAR	N/A
Types.NCLOB	N/A
Types.NUMERIC	NUMBER
Types.NVARCHAR	N/A
Types.OTHER	N/A
Types.REAL	NATIVE_REAL
Types.REF	N/A
Types.REF_CURSOR	REF CURSOR
Types.ROWID	ROWID
Types.SMALLINT	NATIVE_SMALLINT
Types.SQLXML	N/A
Types.STRUCT	N/A

SQL type	GOLDILOCKS type
Types.TIME	TIME
Types.TIME_WITH_TIMEZONE	TIME WITH TIME ZONE
Types.TIMESTAMP	TIMESTAMP
Types.TIMESTAMP_WITH_TIMEZONE	TIMESTAMP WITH TIME ZONE
Types.TINYINT	NATIVE_SMALLINT
Types.VARBINARY	VARBINARY(4000)
Types.VARCHAR	VARCHAR(4000)

Table 32-4 Whether supporting getter method for GOLDILOCKS type - 1

	NATIVE_SMALLINT	NATIVE_INTEGER	NATIVE_BIGINT	NATIVE_REAL	NATIVE_DOUBLE
getByte	O	O	O	O	O
getShort	O	O	O	O	O
getInt	O	O	O	O	O
getLong	O	O	O	O	O
getFloat	O	O	O	O	O
getDouble	O	O	O	O	O
getBigDecimal	O	O	O	O	O
getBoolean	Avaliable only for 0,1	Avaliable only for 0,1	Avaliable only for 0,1	Avaliable only for 0,1	Avaliable only for 0,1
getString	O	O	O	O	O
getBytes	raw data	raw data	raw data	raw data	raw data
getDate	X	X	X	X	X
getTime	X	X	X	X	X
getTimestamp	X	X	X	X	X
getAsciiStream	raw data	raw data	raw data	raw data	raw data
getBinaryStream	raw data	raw data	raw data	raw data	raw data
getCharacterStream	X	X	X	X	X
getClob	O	O	O	O	O
getBlob	raw data	raw data	raw data	raw data	raw data
getArray	X	X	X	X	X
getRef	X	X	X	X	X
getURL	X	X	X	X	X
getObject	Short	Integer	Long	Float	Double
getRowId	X	X	X	X	X

Table 32-5 Whether supporting getter method for GOLDILOCKS Type - 2

	BOOLEAN	FLOAT/ NUMBER	CHAR/ VARCHAR/ LONG VARCHAR	BINARY/ VARBINARY/LONG VARBINARY	ROWID
getByte	0 or 1	O	Available only for numeric	X	X
getShort	0 or 1	O	Available only for numeric	X	X
getInt	0 or 1	O	Available only for numeric	X	X
getLong	0 or 1	O	Available only for numeric	X	X
getFloat	0 or 1	O	Available only for numeric	X	X
getDouble	0 or 1	O	Available only for numeric	X	X
getBigDecimal	0 or 1	O	Available only for numeric	X	X
getBoolean	O	Available only for 0,1	Available only for "t", "f", "true", "false", "y", "n", "yes", "no", "on", "off", "1", "0" (case-insensitive)	X	X
getString	"TRUE" or "FALSE"	O	O	O	O
getBytes	raw data	raw data	raw data	O	raw data
getDate	X	X	Available only for date format	X	X
getTime	X	X	Available only for time format	X	X
getTimestamp	X	X	Available only for timestamp format	X	X
getAsciiStream	raw data	raw data	raw data	O	raw data
getBinaryStream	raw data	raw data	raw data	O	raw data
getCharacterStream	X	X	O	X	X
getClob	"TRUE" or "FALSE"	O	O	O	O
getBlob	raw data	raw data	raw data	raw data	raw data
getArray	X	X	X	X	X
getRef	X	X	X	X	X

	BOOLEAN	FLOAT/ NUMBER	CHAR/ VARCHAR/ LONG VARCHAR	BINARY/ VARBINARY/LONG VARBINARY	ROWID
getURL	X	X	X	X	X
getObject	Boolean	BigDecimal	String	byte[]	RowId
getRowId	X	X	X	X	O

Table 32-6 Whether supporting getter method for GOLDILOCKS Type - 3

	DATE	TIME/ TIME WITH TIME ZONE	TIMESTAMP/ TIMESTAMP WITH TIME ZONE	INTERVAL	REF CURSOR
getByte	X	X	X	Available only for a single item type	X
getShort	X	X	X	Available only for a single item type	X
getInt	X	X	X	Available only for a single item type	X
getLong	X	X	X	Available only for a single item type	X
getFloat	X	X	X	Available only for a single item type	X
getDouble	X	X	X	Available only for a single item type	X
getBigDecimal	X	X	X	Available only for a single item type	X
getBoolean	X	X	X	X	X
getString	O	O	O	O	X
getBytes	raw data	raw data	raw data	raw data	X
getDate	O	O	O	X	X
getTime	O	O	O	X	X
getTimestamp	O	O	O	X	X
getAsciiStream	raw data	raw data	raw data	raw data	X
getBinaryStream	raw data	raw data	raw data	raw data	X
getCharacterStream	X	X	X	X	X
getClob	O	O	O	O	X
getBlob	raw data	raw data	raw data	raw data	X
getArray	X	X	X	X	X
getRef	X	X	X	X	X
getURL	X	X	X	X	X
getObject	Date	Time	Timestamp	GoldilocksInterval	ResultSet

	DATE	TIME/ TIME WITH TIME ZONE	TIMESTAMP/ TIMESTAMP WITH TIME ZONE	INTERVAL	REF CURSOR
getRowId	X	X	X	X	X

33.

Embedded SQL

33.1 Precompiler

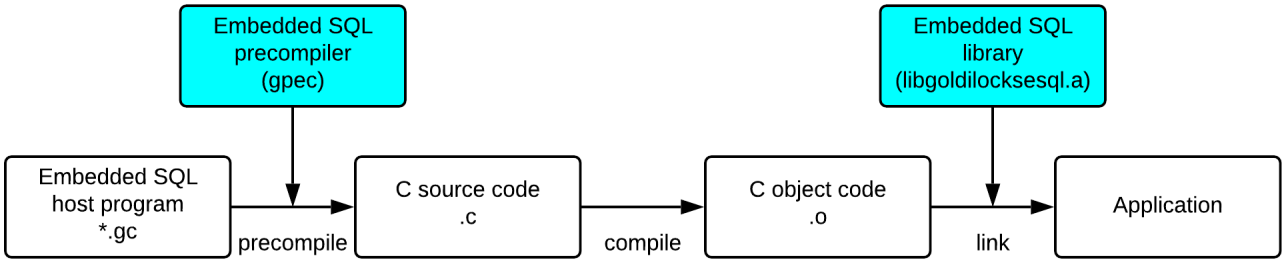
Overview

GOLDILOCKS precompiler is a programming development tool which enables a user to use embedded SQL in the high-level programming language. Currently, GOLDILOCKS supports the precompiler only for the language such as C/ C++, and it is called as gpec.

Developing Embedded SQL Applications

As described in **Figure 1 Developing the embedded SQL applications**, the user creates a C source program including embedded SQL, and converts it via gpec precompiler. Then, the pure C code is created of which the embedded SQL on the source code is converted to the contents calling the library of GOLDILOCKS. This C code uses C compiler of the system to perform compile as object code and link it with libgoldilocksesql.a which is an embedded SQL library provided by GOLDILOCKS, then completes the final application.

Figure 1 Developing the embedded SQL applications



Configuring Embedded SQL Application Development Tool

Embedded SQL application of GOLDILOCKS consists of the following elements.

Directory or File	Description
\bin\gpec	GOLDILOCKS precompiler embedded SQL for C
\include\goldilocksesql.h	It is an embedded SQL library header file. Precompiler automatically inserts it, so the user does not have to do any extra work.
\include\sqlca.h	It is the SQLCA data structure related header file.
\lib\libgoldilocksesql.a, \lib\libgoldilocksesqls.so	It is the embedded SQL run-time library.
\lib\libgoldilocks.a, \lib\libgoldilockss.so	It is the GOLDILOCKS DA/ CS mixed mode library.
\lib\libgoldilocksa.a, \lib\libgoldilocksas.so	It is the GOLDILOCKS DA mode library.

Directory or File	Description
\lib\libgoldilocksc.a, \lib\libgoldilockscs.so	It is the GOLDILOCKS CS mode library.
\sample\EmbeddedSQL	It is a sample program.

Building Application

This chapter describes the process to build an embedded SQL source program of GOLDILOCKS and make the application.

Precompile

Description

The user precompiles the C/ C ++ source code which was written by using embedded SQL, and generates the pure C/ C ++ source code. The core of this process is converting the user-created embedded SQL to t he library call provided by GOLDILOCKS, and the C/ C++ source code is not modified except for embedde d SQL.

Usage

The name of GOLDILOCKS precompiler is gpec, and it is located in $GOLDILOCKS_HOME/bin/.
gpec is used as follows.

```
$ gpec [OPTION]... <input file>
```

<input file> is input to gpec, and gpec generates the C/ C++ source code after precompile process. The e xtension of <input file> is *.gc, and it can be omitted. If <input file> does not have *.gc extension, the file name with the extension should be required.
For more information about options given to gpec, refer to **Precompiler Options**.

Example

```
$ gpec sample1

FileName: sample1
Pre-compile sample1.gc -> sample1.c
```

Compile

C/ C++ source code is generated after precompile process. The source code generates the object code by using the C/ C++ compiler provided in the platform. For more information about this process refer to the C/ C++ compiler manual which is provided by the user platform.

Link

The application is generated by linking the object codes generated through the process above, and GOLD ILOCKS supports libgoldilocksesql.a for an embedded SQL. The library includes the GOLDILOCKS APIs whi ch is converted from the embedded SQL by the precompiler, so it is prerequisite when making the embed ded SQL application.

Additionally, the required library varies upon the various operation modes of GOLDILOCKS, and the requi red library is selected and linked according to the operation mode of the current application as follows.

Operation mode	Static library	Shared object
DA dedicated	libgoldilocksa.a	libgoldilocksas.so
CS dedicated	libgoldilocksc.a	libgoldilockscs.so
DA/ CS mixed	libgoldilocks.a	libgoldilockss.so

Other link processes are as same as the process of generating the common C/ C++ application so refer to the linker manual provided by the user platform.

Example

make is used a lot to easily perform the precompile, compile, link process above. The following is an exa mple of Makefile to create the sample program. Refer to the following example to make the Makefile wh ich is suitable for the user environment.

```
GPEC = gpec
GPECFLAGS =
#GPECFLAGS = --unsafe-null --no-prompt
CC = gcc
CFLAGS = -g -Wall
INC = -I$(GOLDILOCKS_HOME)/include
LFLAGS = -L$(GOLDILOCKS_HOME)/lib
LIB = -lgoldilocksesql -lgoldilocksc -lpthread -lm -lrt

BINS = overview sample1 sample2 sample3 sample4 sample5 dyn1 dyn2 number date_time thread1
fetch_struct_array
```

```
ifneq ($(MAKECMDGOALS), clean)
ifneq ($(MAKECMDGOALS), all)
TARGET = $(MAKECMDGOALS)
OBJECT = $(TARGET).o
C_SRC  = $(TARGET).c
endif
endif
```

- Implicit rules

```
.SUFFIXES: .gc .c .o
.gc.c:
    $(GPEC) $(GPECFLAGS) $^          ❶ Precompile
.c.o:
    $(CC) $(CFLAGS) -c $(INC) $^     ❷ Compile
```

- Build rules

```
NoTarget :
    @echo "Syntax : make {all | sample_name | clean}"
    @echo "sample_name is one of '$(BINS)'"

all :
    for target in $(BINS); do \
        $(MAKE) $$target;     \
    done

$(OBJECT) : $(C_SRC)
$(TARGET) : $(OBJECT)
    $(CC) -o $@ $^ $(LFLAGS) $(LIB)    ❸ Link

clean :
    rm -rf $(BINS) *.o *.c *~ core
```

Sample

GOLDILOCKS provides a simple embedded SQL sample code to help to create an embedded SQL applicati on. The sample code is in $GOLDILOCKS_HOME/sample/EmbeddedSQL directory, and sample.sql which i s in the same directory should be executed first to execute the samples.

```
$ cd $GOLDILOCKS/sample/EmbeddedSQL
$ gsql test test -i sample.sql
$ make
Syntax : make {all ¦ sample_name ¦ clean}
sample_name is one of 'overview sample1 sample2 sample3 sample4 sample5 dyn1 dyn2 number
date_time thread1 fetch_struct_array'
```

If *make all* is executed, all samples are built. Execute make <sample_name> to build a particular sample al one. For <Sample_name>, refer to the message above. If sample2 is built and executed, and the result is as follows.

```
$ make sample2
gpec  sample2.gc
FileName: sample2.gc
Pre-compile sample2.gc -> sample2.c
gcc -g -Wall -c -I/home/mycomman/work/product/Gliese/home/include sample2.c
gcc -o sample2 sample2.o -L/home/mycomman/work/product/Gliese/home/lib -lgoldilocksesql
-lpthread -lm -lrt -lgoldilocksa
$ ./sample2
Connect goldilocks ...
 EMPNO    ENAME                JOB       SALARY
====== ==================== ========== ========
  2854                 Park         RND       800
  2098                  Kim    SALESMAN      1600
  2175                 Choi    SALESMAN      1250
  2306                  Lee     SUPPORT      2975
  2122                  Lyu    SALESMAN      1250
  2999                  Ohn     SUPPORT      2850
  2012                Cheon     SUPPORT      2450
  2168                 Sohn         RND      3000
  2836                  Seo         CEO      5000
  2022                 Song    SALESMAN      1500
  2232                Jeong         RND      1100
  2676                 Kang         RND       950
  2714                  Cho         RND      3000
  2441                 Yoon         RND      1300
====== ==================== ========== ========
Record Count = 14
====== ==================== ========== ========
SUCCESS
############################
```

Precompiler Options

This chapter describes the options of gpec.

--no-prompt, -n

Description

It does not output the version information.

Example

```
$ gpec --no-prompt sample2
FileName: sample2
Pre-compile sample2.gc -> sample2.c
$
```

--version, -v

Description

It outputs only the version information, then exits.

Example

```
$ gpec --version
$
```

--help, -h

Description

It outputs the help message. It performs the same operation even when an option or <input file> does no t exist in gpec.

Example

```
$ gpec --help
gpec is the GOLDILOCKS embedded SQL precompiler for C programs.
```

```
Usage:
  gpec [OPTION]... <input file>

Options:
  --no-prompt    No Print version information
  --version      Print version information and exit
  --help         Print help message
  --output       Describe output filename
  --unsafe-null  Allow a NULL fetch without indicator variable
  --include-path Describe header file path
  --no-lineinfo  Exclude line information
  --char_map     Mapping of character arrays ( CHARZ ¦ STRING )
  --cumulative   SQLERRD[2] records cumulative sum of rows processed for FETCH CURSOR.
  --autocommit   Set autocommit TRUE to all connection.
$
```

--output, -o

Description

It specifies the name of precompile result file. If this option is not given, the file name as same as <input file> is created with the extension of .c.

Example

- When an output is not used.

```
$ gpec sample2.gc
FileName: sample2.gc
Pre-compile sample2.gc -> sample2.c
$ ls
sample2.c  sample2.gc
```

- When an output is used.

```
$ gpec --output outfile.cpp sample2.gc
FileName: sample2.gc
Pre-compile sample2.gc -> outfile.cpp
$ ls
outfile.cpp  sample2.gc
```

--unsafe-null

Description

Even if it does not use the host indicator variable, It succeeds when NULL fetch occurs. It means that the operation is successful, but it does not mean that the NULL value can be fetched.

Example

```
$ gpec --unsafe-null sample2
FileName: sample2
Option : --unsafe-null
Pre-compile sample2.gc -> sample2.c
$
```

--include-path, -I

Description

It describes the path of the header file to refer when performing the precompile. Other header files are fo und by using EXEC SQL INCLUDE statement when performing the precompile. First, it searches in the dire ctory where the current file is located. If it can not find any file, then it searches in the directories where t he option is described in turn.

Example

The following is an example of when the header file is in include directory.

- When an error occurs.

```
$ gpec sample.gc
FileName: sample.gc
Pre-compile sample.gc -> sample.c
ERR-42000(41000): syntax error
Error at line 12, in file sample.gc
ERR-42000(41004): "decl.h": file not exist
ERR-42000(41000): syntax error
rsEmpRecord gRecord[] = {
^
Error at line 15, in file sample.gc
```

- When -I option is used.

```
$ gpec -Iinclude sample.gc
FileName: sample.gc
Pre-compile sample.gc -> sample.c
$
```

--no-lineinfo

Description

When gc file is converted to c file by default, GPEC adds #liine information to enable debugging with gc f ile. However, if this option is used, It does not add the line information through #line preprocessor when creating c file.

Example

```
$ gpec --no-lineinfo sample2
FileName: sample2
Pre-compile sample2.gc -> sample2.c
$
```

--char_map, -c

Description

It sets to which type the char type data declared in a DECLARE SECTION is mapped. The default value is 'STRING', and it is null-terminated data type. 'CHARZ' is space padded and null-terminated data type.

Example

```
$ gpec --char_map=STRING overview
FileName: overview
Pre-compile overview.gc -> overview.c
$ gpec --char_map=CHARZ overview
FileName: overview
Pre-compile overview.gc -> overview.c
```

--define, -D

Description

It is a define name used in gpec and it is set to 1.

Example

```
$ gpec --define=AAA preprocess
FileName: preprocess
Pre-compile preprocess.gc -> preprocess.c
$ gpec -D BBB preprocess
FileName: preprocess
Pre-compile preprocess.gc -> preprocess.c
ERR-42000(41028): 'BBB' macro is already defined at line 45, in file preprocess.gc
```

--cumulative

Description

It processes sqlerrd[2] in FETCH CURSOR statement as the accumulated total.

Example

```
$ gpec --cumulative sample.gc
FileName: sample Option : --cumulative
Pre-compile sample.gc -> sample.c
$
```

--autocommit

Description

It sets *auto commit* of all connections to TRUE.

Example

```
$ gpec --autocommit sample.gc
FileName: sample Option : --autocommit
Pre-compile sample.gc -> sample.c
$
```

33.2 Embedded SQL

Preprocessing

Overview

It performs the preprocessing before precompiling in gpec.

gpec supports the preprocess statements such as #if, #ifdef, #if defined, #ifndef #else, #elif, #endif, #define, #undef.

It can use a predefine by using a --define option of gpec. It is defined as 1 when it is predefined with a gpec option.
e.g. gpec --define=_DEV_ Test.gc is as same as #define _DEV_ (1) in Test.gc file.

Applicable Range

The host variable of SQL precompiler feature for gpec is applied only within a declare section and the feature is used in the EXEC SQL statement. (The host variable outside of the declare section can not be used in the host EXEC SQL statement.)
gpec preprocessor are applied all over the source.
Only the header file declared with EXEC SQL INCLUDE is applied to an include file.

```
#define _DEV1_
EXEC SQL BEGIN DECLARE SECTION;
#define _DEV2_
char username[10];
char password[10];
#ifdef _DEV1_
VARCHAR conn_str[20];  ❶
#elif defined _DEV2_
VARCHAR conn_str[30];  ❷
#endif
EXEC SQL END DECLARE SECTION;
```

❶ _DEV1_ is defined irrespective of the position of DECLARE SECTION, so it is processed as a host variable in gpec.
❷ _DEV1_ is defined, so its location becomes false, so is processed as a white space in gpec.

When creating c file from the file above using gpec, conn_str[20] is converted and #ifdef, #elif, #endif pr eprocessor statements and conn_str[30] are converted into white spaces as follows.

```
/* EXEC SQL BEGIN DECLARE SECTION; */
#define _DEV2_
char username[10];
char password[10];
/* VARCHAR conn_str[20]; */
struct { int len; char arr[20]; } conn_str; ❶

                              ❷

/* EXEC SQL END DECLARE SECTION; */
```

❶ _DEV1_ is defined irrespective of the position of DECLARE SECTION, so it is processed as a h ost variable in gpec.
❷ _DEV1_ is defined, so its location becomes false, so is processed as a white space in gpec.

Types

#if

- Syntax

```
#if constant
```

or

```
#if defined identifier
```

or

```
#if !defined identifier
```

- Example

```
#if 0
int sVar1;
#endif
#if 3-2   ❶ An operation is available.
int sVar2;
```

```
#endif
#if defined _DEV_
int sVar3;
#endif
#if !defined (_DEV_)
int sVar4;
#endif
```

#ifdef, #ifndef

- Syntax

```
#ifdef identifier
```

or

```
#ifndef identifier
```

- Example

```
#ifdef _DEV_
int sVar1;
#endif
#ifndef _DEV_
int sVar2;
#else
int sVar3;
#endif
```

#else, #elif, #endif

- Syntax

```
#else
```

or

```
#endif
```

or

```
#elif constant
```

or

```
#elif defined identifier
```

- Example

```
#if 1
int sVar1;
#else
int sVar2;
#endif
#if 0
int sVar3;
#elif 1
int sVar4;
#else
int sVar5;
#endif
#ifdef _DEV1_
int sVar6;
#elif defined _DEV2_
int sVar7;
#elif !defined _DEV3_
int sVar8;
#endif
```

#define, #undef

- Syntax

```
#define identifier
```

or

```
#define identifier constant
```

or

```
#undef identifier
```

- Example

```
#define _DEV1_
EXEC SQL BEGIN DECLARE SECTION;
#define _DEV2_
char username[10];
char password[10];
#ifdef _DEV1_
VARCHAR conn_str[20]; ❶
#elif defined _DEV2_
VARCHAR conn_str[30]; ❷
#endif
#undef _DEV2_
#ifdef _DEV2_
VARCHAR sDept[10];  ❸
#endif
EXEC SQL END DECLARE SECTION;
```

❶ _DEV1_ is defined, so it is processed as a host variable in gpec.
❷ #ifdef _DEV1_is true, so _DEV2_is processed as a white space in gpec.
❷ _DEV2_ is undefed, so it is processed as a white space in gpec.

An annotation can be used in #define. However, if #define are written along multiple lines, the n the annotation can not be correctly processed.

```
#define _DEF1_  1 \ ❶
    + 1
#define _DEF2_   1 \ /* this ❷
is comment */ + 1
#define _DEF3_  1  /* this is comment */ + 1  ❸
```

❶ _DEF1_ is processed as 1 + 1.
❷ _DEF2_is processed as 1.
❸ _DEF3_is processed as 1 + 1.

Constraints

Even though it is defined within a declare section, it is not extended to an EXEC SQL statement.

```
EXEC SQL BEGIN DECLARE SECTION;
#define C_EMP_NO   14
char username[10];
EXEC SQL END DECLARE SECTION;
EXEC SQL
    SELECT USERNAME INTO :username
    FROM EMP
    WHERE EMPNO = C_EMP_NO; ❶ A wrong MACRO is used.
```

Expansion

MACRO can be used in the middle of c statement or in the middle of EXEC SQL statement.

```
#define _DEV_
EXEC SQL BEGIN DECLARE SECTION;
char
#ifdef _DEV_
sTrue[10];
#else
sFalse[10];
#endif
EXEC SQL END DECLARE SECTION;
EXEC SQL SELECT
#ifdef _DEV_
         "true" INTO :sTrue
#else
         "false" INTO :sFalse
#endif
         FROM DUAL;
```

gpec can execute the preprocessor to create the following c code. The following is an example in which t he converting from SQL statement to c code is omitted.

```
#define _DEV_
EXEC SQL BEGIN DECLARE SECTION;
char


sTrue[10];
```

```
EXEC SQL END DECLARE SECTION;
EXEC SQL SELECT

         "true" INTO :sTrue



         FROM DUAL;
```

Connection

Connecting to Database

A process of connecting to the database server is required to perform the operation by connecting to the database in the embedded SQL program.
The syntax for connecting to the database is expressed in GOLDILOCKS as follows.

```
EXEC SQL [ AT <db_name> ] CONNECT <user_name> IDENTIFIED BY <password> [ AT <db_name> ] [
USING <conn_string> ]
<db_name> := dbname ¦ :hostvar
<user_name> := username ¦ :hostvar
<password> := password ¦ :hostvar
<conn_string> := connection_string ¦ :hostvar
```

An example of the most basic way to connect to database is as follows.

```
EXEC SQL BEGIN DECLARE SECTION;
char username[10];
char password[10];
EXEC SQL END DECLARE SECTION;
strcpy( username, "test" );
strcpy( password, "test" );
...
EXEC SQL CONNECT :username IDENTIFIED BY :password;
```

GOLDILOCKS supports both D/A mode and C/S mode. The D/A mode is operated by attaching directly to the shared memory, and C/S mode is operated by connecting to the database by using TCP communicati

on. When operated in D/A mode, it is not necessary to include a separate server information as described above because the direct access from the same host of the database is performed. When operated in C/S mode, Data Source Name (DSN) should be specified to access. For more information about DSN, refer to **Data Source Configuration**.

connection_string information should be given to use DSN. USING clause is used to use connection_strin g information. An example of the connection statement whose DSN is "GOLDILOCKS", and using USING clause is as follows.

```
EXEC SQL BEGIN DECLARE SECTION;
char username[10];
char password[10];
char conn_str[20];
EXEC SQL END DECLARE SECTION;
strcpy( username, "test" );
strcpy( password, "test" );
strcpy( conn_str, "DSN=GOLDILOCKS" );
...
EXEC SQL CONNECT :username IDENTIFIED BY :password USING :conn_str;
```

When developing an application, sometimes it is required to uniquely identify the respective connection. I t is the case of which each connection is performed in the multi-thread program in D/A mode. Or it is the case of which several connections are performed in C/S mode. The name is given to each connection by u sing AT clause.

AT clause may be positioned in front of CONNECT statement or USING clause. The example of AT clause i n CONNECT statement is as follows.

```
EXEC SQL BEGIN DECLARE SECTION;
char username[10];
char password[10];
char conn_str[20];
char conn_name[10];
EXEC SQL END DECLARE SECTION;
strcpy( username, "test" );
strcpy( password, "test" );
strcpy( conn_str, "DSN=GOLDILOCKS" );
strcpy( conn_name, "DBCONN1" );
...
EXEC SQL CONNECT :username IDENTIFIED BY :password AT :conn_name USING :conn_str;
```

Disconnecting Database

It disconnects the connection to the database in the application. Like as the CONNECT statement, the dis connection is performed for the default connection or it is performed by giving a connection name. The s yntax for disconnecting all connections performed in the application is also provided.

Single Disconnection

There are two ways of disconnecting a single connection, which are explicit and implicit method.
The explicit method uses DISCONNECT statement as follows.

```
EXEC SQL [ AT <db_name> ] DISCONNECT;
```

The transaction is terminated via commit or rollback. The implicit disconnection is performed by adding R ELEASE option after the transaction termination statement.

```
EXEC SQL [ AT <db_name> ] { COMMIT/ROLLBACK } [ WORK ] RELEASE;
```

All Disconnection

All connections of the current application are disconnected all together as follows.

```
EXEC SQL DISCONNECT ALL;
```

Transaction

Database application consists of transaction units. Therefore, an embedded SQL program should be able to manipulate the transaction. This chapter describes how to manipulate the transaction.

Start and End of Transaction

A transaction starts in the SQL which is performed first after it is connected. The transaction is maintained until the end command explicitly occurs. There are two end commands, which are COMMIT and ROLLBA CK.

COMMIT

The transaction commit command causes the following actions.

- All data changes after the current transaction started are permanently reflected in the database.
- The reflected changes are visibly applied to all transactions which started after the changes or to the s ensitive cursor.

- All savepoints which are created in the current transaction are deleted.
- All locks which are obtained in the current transaction are released.
- All cursors which are opened in the current transaction are closed. (The holdable cursor is excluded.)
- The transaction is terminated.

The commit command is used as follows.

```
EXEC SQL COMMIT [ WORK ];
```

ROLLBACK

The rollback command types are as follows.

- Transaction full rollback
- Transaction partial rollback
- Statement-level rollback

The transaction rollback command causes the following actions.

- All data changes after the current transaction started are undone and it returns to the state of before the transaction starts.
- All savepoints which are created in the current transaction are deleted.
- All locks which are obtained in the current transaction are released.
- All cursors which are opened in the current transaction are closed. (The holdable cursor is excluded.)
- The transaction is terminated.

The rollback command is used as follows.

```
EXEC SQL ROLLBACK [ WORK ];
```

Transaction partial rollback uses the savepoint. The application developer explicitly specifies the savepoint, and the transaction partial rollback is implemented by undoing all operations up to the savepoint. The tra nsaction partial rollback command causes the following actions.

- All changes after the rollback savepoint are undone.
- All savepoints created after the rollback savepoint are deleted.
- All locks obtained after the rollback savepoint are released.

Transaction partial rollback command is used as follows.

```
EXEC SQL ROLLBACK TO SAVEPOINT <savepoint_name>;
```

Statement-level rollback undoes only the statement which is currently executed. For example, as if rows a re inserted to a table and the unique violation occurs so the current statement can not be performed, the

n only the currently executed statement should be undone so that the next operation may continue.

In this case, there is not a command to be explicitly specified by the user because GOLDILOCKS internally undoes the statement.

Auto Commit

If the embedded SQL of GOLDILOCKS generally performs the connection, transaction is operated in non auto-commit mode.

However, it may be required to adjust the auto-commit mode for the convenience of application develop ment or for the logical environment of the application. The embedded SQL precompiler of GOLDILOCKS may turn on or off the auto-commit mode with the following syntax.

```
EXEC SQL [ AT <db_name> ] ATUTOCOMMIT { ON ¦ OFF };
```

RELEASE Option

The currently used connection may be turned off by using the RELEASE option when ending the transacti on (commit/ rollback).
The option is allowed to be applied only for the transaction full commit rollback. It can not be used for th e transaction partially rollback (ROLLBACK TO SAVEPOINT).

Host Variables and Datatypes

The embedded SQL application aims at manipulating the data and querying to obtain the desired results by interworking with the database server.

A method for the application data to be transferred to the database server, or vise versa, is required for t his operation. The method to perform this role is defined as a host variable.

The application can use the host variables in the same way as C variables because the host variable is decl ared as a variable of C language. A host variable is treated as a part of the SQL statement and is responsi ble for the input/output of value between the database server and applications.

Declaring Host Variable

Host variable should be declared in the embedded SQL directive as follows.

```
EXEC SQL BEGIN DECLARE SECTION;
```

- Declaring a host variable

```
EXEC SQL END DECLARE SECTION;
```

The section above is called as a declare section, and host variables can be declared within the declare sect ion in the same way as C variables. The following is an example of declaring a part of host variables.

```
EXEC SQL BEGIN DECLARE SECTION;
    int     empno;
    char    ename[20];
    double  salary;
EXEC SQL END DECLARE SECTION;
```

Declaring Function Argument

When using the function argument as a host variable, it is required to provide the information about the function argument. This information should be declared within the next embedded SQL directive.

```
EXEC SQL BEGIN ARGUMENT SECTION;
```

- Declaring a host variable

```
EXEC SQL END ARGUMENT SECTION;
```

The section above is called as an argument section, and the host variable should be declared as same as t he function argument's name and its type within the argument section.

```
void func( int empno, char ename[20], double salary )
{
EXEC SQL BEGIN ARGUMENT SECTION;
     int     empno;
     char    ename[20];
     double  salary;
EXEC SQL END ARGUMENT SECTION;
...
```

A pseudo type can not be used literally as a function argument, but it can be used by using the declare se ction and typedef keyword.

```
EXEC SQL BEGIN DECLARE SECTION;
typedef VARCHAR domainName[20];
EXEC SQL END DECLARE SECTION;
```

```
void func( int * no, domainName * name )
{
    EXEC SQL BEGIN ARGUMENT SECTION;
    int        no[20];
    domainName name[20];
    EXEC SQL END ARGUMENT SECTION;
    EXEC SQL INSERT INTO EMP VALUES ( :no, :name );
}
```

If the host variable is declared as a pointer type, then the array length is unknown. Even when t he function argument is a pointer type, then the array length of the host variable should be spe cified in the argument section.

C Data Type for Host Variable

There are two types of C data used as the host variables, and which are a native type provided by the C la nguage and a data type additionally provided by GOLDILOCKS. The following table describes the data typ es provided by the embedded SQL of GOLDILOCKS.

C Native Datatype

C native datatype is the default data type provided by the C language, and its range and size are entirely dependent on the application development platform.

Table 33-1 C native datatype

C datatype	Description
char	It is a single character.
char[n]	It is a string with maximum length n.
short	It is a small integer (2 bytes).
int	It is an integer (4 bytes).
long	It is a large integer (4/8 bytes).
long long	It is a very large integer (8 bytes).
float	It is a single precision floating-point number.
double	It is a double precision floating-point number.

- char

char type represents a single character.

- char[n]

char[n] represents a string with maximum length n.
The following is an example.

```
EXEC SQL BEGIN DECLARE SECTION;
char strName[20];
EXEC SQL BEGIN DECLARE SECTION;
```

If it is declared as above, strName represents to the string data with the maximum length 20.
If strName is used as an output of ESQL, then it is space padded.


The maximum length of a char[] can not exceed 2001.

- short

It represents 2 bytes integer datatype.

- int

It represents 4 bytes integer datatype.

- long

Long type means a large integer, but the range of the actual value is determined in accordance with the platform. Long is 8 bytes integer in 64 bits Unix/Linux platform, but it is 4 bytes integer in 32 bits Unix/Linux platform.

- long long

It means very large integer, and it represents the 8 bytes integer.

- float

It is the float type of C language, and it represents the 4 bytes single-precision floating-point number.

- double

It is the double type of C language, and it represents the double-precision floating-point number.

Pseudo Datatype

Pseudo type supports the various forms of GOLDILOCKS type, and it is the type provided by GOLDILOCKS embedded SQL precompiler for the convenience of development. Most of its contents are implemented by using the structure of C.

Table 33-2 GOLDILOCKS embedded SQL pseudo type

Pseudo type	Description
VARCHAR[n]	It is a variable-length string with the maximum length n.
LONG VARCHAR	It is a variable-length string with the maximum length 100M (104857600).
BINARY[n]	It is a binary data with the maximum length n.
VARBINARY[n]	It is a variable-length binary data with the maximum length n.
LONG VARBINARY	It is a variable-length binary data with the maximum length 100M (104857600).
NUMBER	It is an integer whose number of significant digits is 38.
NUMBER(p)	It is an integer whose number of significant digits is p.
NUMBER(p, s)	It is a real number whose number of significant digits is p and whose scale is s.
BOOLEAN	It is a boolean type.
DATE	It is a date type data.
TIME	It is a time type data.
TIME WITH TIMEZONE	It is a time type data with timezone.
TIMESTAMP	It is a datetime type data.
TIMESTAMP WITH TIMEZONE	It is a time type data with timezone.
INTERVAL YEAR	It is an interval data type.
INTERVAL MONTH	
INTERVAL DAY	
INTERVAL HOUR	
INTERVAL MINUTE	
INTERVAL SECOND	
INTERVAL YEAR TO MONTH	
INTERVAL DAY TO HOUR	
INTERVAL DAY TO MINUTE	
INTERVAL DAY TO SECOND	
INTERVAL HOUR TO MINUTE	
INTERVAL HOUR TO SECOND	
INTERVAL MINUTE TO SECOND	

VARCHAR

VARCHAR type is the datatype in which the variable length string can be stored, and it consists of the foll owing structure.

```
struct {
    int  len;
    char arr[n];
}
```

n refers to the maximum length of VARCHAR type.

```
EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR varstr[100];
EXEC SQL END DECLARE SECTION;
```

For example, if it is declared as above, it is converted via precompile process as follows.

```
struct VARCHAR_varstr {
    int  len;
    char arr[100];
} varstr;
```

The application can use VARCHAR type as follows.

```
strcpy( varstr.arr, "abcde" );
varstr.len = strlen( varstr.arr );

EXEC SQL INSERT INTO TEST_T1 VALUES ( :varstr );
```

The length of VARCHAR can not exceed 4000.

LONG VARCHAR

LONG VARCHAR type is the datatype in which a long variable-length strings can be stored, and it consists of the following structure.

```
typedef struct SQL_LONG_VARIABLE_LENGTH_STRUCT
{
    SQLBIGINT   len;
    SQLCHAR   * arr;
} SQL_LONG_VARIABLE_LENGTH_STRUCT;
```

LONG VARCHAR type can be declared as follows.

```
EXEC SQL BEGIN DECLARE SECTION;
    LONGVARCHAR long_text[1048576];
EXEC SQL END DECLARE SECTION;
```

LONG VARCHAR is different from VARCHAR. LONG VARCHAR can have the length up to 100M (104857 600), so it does not pre-allocate the space for the string when declaring. After declaring LONG VARCHAR, the memory space should be allocated in long_text.arr before actual use, and the application should relea se the allocated memory space. An example of using LONG VARCHAR is as follows.

```
long_text.arr = malloc( 1048576 );

gets( long_text.arr );
long_text.len = strlen( long_text.arr );

EXEC SQL INSERT INTO TEST_T1 VALUES ( :long_text );
...
free( long_text.arr );
```

LONG VARCHAR type can be declared by specifying the current length up to 100M (104857600).

BINARY

BINARY type is the datatype to literally handle the non-formal raw data, and it consists of the following structure. The example of declaring BINARY type is as follows.

```
EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR binary[100];
EXEC SQL END DECLARE SECTION;
```

If it is declared as above, it is converted via precompile process as follows.

```
char binary[100];
```

Its form is as same as the character string storage form. char[n] is defined to store the string of database, but BINARY type is for storing the binary data. Therefore, the applications can use BINARY type in the same way as dealing with the character string data.

The maximum length of BINARY can not exceed 2000.

VARBINARY

VARBINARY type is the datatype in which the variable length binary data can be stored, and it consists of the following structure.

```
struct {
    int  len;
    char arr[n];
}
```

n refers to the maximum length of VARCHAR type.

```
EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR varbin[100];
EXEC SQL END DECLARE SECTION;
```

For example, if it is declared as above, it is converted via precompile process as follows.

```
struct VARBINARY_varbin {
    int  len;
    char arr[100];
} varbin;
```

The forms of VARBINARY type and VARCHAR type are same. VARCHAR is for storing the string data, but VARBINARY is for storing the binary data. VARCHAR and VARBINARY are same in all other aspects except for the aspect above. Therefore, the application can use both VARCHAR and VARBINARY types as follows.

```
memcpy( varbin.arr, binary_data, 50 );
varbin.len = 50;

EXEC SQL INSERT INTO TEST_T1 VALUES ( :varbin );
```

The length of VARBINARY can not exceed 4000.

LONG VARBINARY

LONG VARBINARY type is the datatype in which the long variable length binary data can be stored, and it consists of the following structure.

```
typedef struct SQL_LONG_VARIABLE_LENGTH_STRUCT
{
    SQLBIGINT   len;
    SQLCHAR   * arr;
} SQL_LONG_VARIABLE_LENGTH_STRUCT;
```

LONG VARBINARY type can be declared as follows.

```
EXEC SQL BEGIN DECLARE SECTION;
    LONGVARBINARY long_bin[1048576];
EXEC SQL END DECLARE SECTION;
```

The difference between LONG VARBINARY and VARBINARY is that the space is not pre-allocated to store binary data when declaring LONG VARBINARY type.(It is as same as the difference between LONG VARCHAR and VARCHAR). After declaring LONG VARBINARY, the memory space should be allocated in long_bin.arr before actual use, and the application should release the allocated memory space after use. An example of using LONG VARBINARY is as follows.

```
long_bin.arr = malloc( 1048576 );

memcpy( long_bin.arr, long_binary_data, 1048576);
long_bin.len = 1048576;

EXEC SQL INSERT INTO TEST_T1 VALUES ( :long_bin );
...
free( long_bin.arr );
```

LONG VARBINARY type can be declared by specifying the current length up to 100M(104857600).

NUMBER

NUMBER type is the datatype which provides an SQL_NUMERIC_STRUCT defined in ODBC to make it to be used in the application.

```
#define SQL_MAX_NUMERIC_LEN 16
typedef struct tagSQL_NUMERIC_STRUCT
{
    SQLCHAR precision;
    SQLSCHAR scale;
    SQLCHAR sign; /* 1=pos 0=neg */
    SQLCHAR val[SQL_MAX_NUMERIC_LEN];
} SQL_NUMERIC_STRUCT;
```

NUMBER type is configured to express the real number type data with precision and scale. When declaring NUMBER type, precision and scale can also be declared. Scale and precision can be omitted as follows in some cases.

Table 33-3 Scale and precision of number types

Declaring NUMBER type	Description
NUMBER	It is as same as NUMBER(38, 0).
NUMBER(p)	It is as same as NUMBER(p, 0)
NUMBER(p, s)	It is a real number type data with precision p, scale s.

```
EXEC SQL BEGIN DECLARE SECTION;
    NUMBER        number_default;
    NUMBER(20)    number_20;
    NUMBER(30,10) number_30_10;
EXEC SQL END DECLARE SECTION;
```

For example, if the variable is declared as above, number_default variable is as same as the declaration of NUMBER(38, 0), and number_20 variable is as same as the declaration of NUMBER(20, 0). NUMBER type uses SQL_NUMERIC_STRUCT which is a ODBC type. The following is a sample of using NUMBER type.

```
/*
 * number.gc
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
EXEC SQL INCLUDE SQLCA;
#define  SUCCESS  0
#define  FAILURE  -1
#define  PRINT_SQL_ERROR(aMsg)                                    \
    {                                                             \
        printf("\n");                                             \
        printf(aMsg);                                             \
        printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n",   \
               sqlca.sqlcode,                                     \
               SQLSTATE,                                          \
               sqlca.sqlerrm.sqlerrmc );                          \
    }

int Connect(char *aHostInfo, char *aUserID, char *sPassword);
int CreateTable();
int DropTable();

unsigned long long ConvertMantisaToDecimal(SQLCHAR *aNumStrValue)
{
    unsigned long long sResult = 0;
    unsigned long long sLast=1;
    unsigned int sCurrent;
    unsigned int sLSD = 0;
    unsigned int sMSD = 0;
```

```
    int          i    = 1;
    for(i = 0; i < SQL_MAX_NUMERIC_LEN; i ++)
    {
        sCurrent = (unsigned char) aNumStrValue[i];
        sLSD = sCurrent % 16; //Obtain LSD
        sMSD = sCurrent / 16; //Obtain MSD
        sResult += sLast * sLSD;
        sLast = sLast * 16;
        sResult += sLast * sMSD;
        sLast = sLast * 16;
    }
    return sResult;
}

void PrintNumber(SQL_NUMERIC_STRUCT *aNumber)
{
    unsigned long long  sDigit;
    unsigned long long  sFraction;
    unsigned long long  sMantisa;
    unsigned long long  sFactor;
    int     i;
    sMantisa = ConvertMantisaToDecimal( aNumber->val );
    sFactor = 1;
    for( i = 0; i < aNumber->scale; i ++ )
    {
        sFactor *= 10;
    }
    sDigit = sMantisa / sFactor;
    sFraction = sMantisa % sFactor;
    if( sFraction != 0 )
    {
        printf("%llu.%-3llu", sDigit, sFraction);
    }
    else
    {
        printf("%llu", sDigit);
    }
}

int main(int     argc,
         char  **argv)
```

```
{
    EXEC SQL BEGIN DECLARE SECTION;
    NUMBER        sNumber;
    NUMBER(10,5)  sResultNumber1;
    NUMBER(10,5)  sResultNumber2;
    char          sCharNumber[20];
    int           sNo;
    int           sResultNo;
    EXEC SQL END DECLARE SECTION;
    int   i;
    int   sState = 0;
    printf("#### Number Datatype Test ####\n");
    printf("Connect GOLDILOCKS ...\n");
    if(Connect("DSN=GOLDILOCKS", "test", "test") != SUCCESS)
    {
        goto fail_exit;
    }
    printf("Create table ...\n");
    if(CreateTable() != SUCCESS)
    {
        goto fail_exit;
    }
    sState = 1;
    printf("Insert record ...\n");
    for(i = 0; i < 20; i ++)
    {
        sNo = i + 1;
        memset( &sNumber, 0x00, sizeof(SQL_NUMERIC_STRUCT) );
        sNumber.precision = 38;
        sNumber.scale = 3;
        sNumber.sign = 1;
        /*
         * 0x627d = 25213
         */
        sNumber.val[0] = 0x7d + i;
        sNumber.val[1] = 0x62;
        snprintf( sCharNumber, 20, "25.2%02d", 13 + i );
        EXEC SQL
            INSERT INTO TEST_T1(C1, C2, C3)
            VALUES(:sNo, :sNumber, :sCharNumber);
        if(sqlca.sqlcode != 0)
```

```
        {
            goto fail_exit;
        }
    }
    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    printf("Retrive record\n");
    EXEC SQL
        DECLARE CUR1 CURSOR FOR
        SELECT C1, C2, C3
        FROM   TEST_T1;
    EXEC SQL OPEN CUR1;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    printf(" NO   Number1  Number2\n");
    printf("==== ======== ========\n");
    memset( &sResultNumber1, 0x00, sizeof(SQL_NUMERIC_STRUCT) );
    memset( &sResultNumber2, 0x00, sizeof(SQL_NUMERIC_STRUCT) );
    while( 1 )
    {
        EXEC SQL
            FETCH FROM CUR1
            INTO :sResultNo, :sResultNumber1, :sResultNumber2;
        if(sqlca.sqlcode == SQL_NO_DATA)
        {
            /*
             * No more data
             */
            break;
        }
        if(sqlca.sqlcode != 0)
        {
            goto fail_exit;
        }
        printf("%3d   ", sResultNo);
        PrintNumber( &sResultNumber1 );
```

```
        printf("   ");
        PrintNumber( &sResultNumber2 );
        printf("\n");
    }
    printf("==== ======== ========\n");
    EXEC SQL CLOSE CUR1;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    sState = 0;
    printf("Drop table ...\n");
    if(DropTable() != SUCCESS)
    {
        goto fail_exit;
    }
    printf("Disconnect GOLDILOCKS ...\n");
    EXEC SQL COMMIT WORK RELEASE;
    printf("SUCCESS\n");
    printf("############################\n");
    return 0;
  fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    printf("FAILURE\n");
    printf("############################\n\n");
    switch(sState)
    {
        case 1:
            printf("Drop table ...\n");
            (void)DropTable();
        default:
            break;
    }
    EXEC SQL ROLLBACK WORK RELEASE;
    return 0;
}
```

```
int CreateTable()
{
    EXEC SQL DROP TABLE IF EXISTS TEST_T1;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
```

- Create table

```
EXEC SQL CREATE TABLE TEST_T1 ( C1  INTEGER,
                                C2  NUMERIC(38,4),
                                C3  VARCHAR(20) );
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    return SUCCESS;
  fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL ROLLBACK WORK;
    return FAILURE;
}
```

- Drop table

```
int DropTable()
{
    EXEC SQL DROP TABLE TEST_T1;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
```

```
        goto fail_exit;
    }
    return SUCCESS;
  fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL ROLLBACK WORK;
    return FAILURE;
}

int Connect(char *aHostInfo, char *aUserID, char *sPassword)
{
    EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR  sUid[80];
    VARCHAR  sPwd[20];
    VARCHAR  sConnStr[1024];
    EXEC SQL END DECLARE SECTION;
```

- Log on GOLDILOCKS

```
strcpy((char *)sUid.arr, aUserID);
    sUid.len = (short)strlen((char *)sUid.arr);
    strcpy((char *)sPwd.arr, sPassword);
    sPwd.len = (short)strlen((char *)sPwd.arr);
    strcpy((char *)sConnStr.arr, aHostInfo);
    sConnStr.len = (short)strlen((char *)sConnStr.arr);
```

- DB connection

```
EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    return SUCCESS;
  fail_exit:
    PRINT_SQL_ERROR("[ERROR] Connection Failure!");
    return FAILURE;
}
```

BOOLEAN

BOOLEAN type has the value of TRUE or FALSE. Actually, the variable is used as the variable of C languag e, so if the variable is 1, it refers to TRUE. If the variable is 0, it refers to FALSE.
The following is an example of using a simple BOOLEAN type variable.

```
EXEC SQL BEGIN DECLARE SECTION;
    BOOLEAN boolean;
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT IsEnable INTO :boolean FROM STATUS WHERE ID = 100;

if( boolean != 0 )
{
    print( "Enable Status : TRUE\n" );
}
else
{
    print( "Enable Status : FALSE\n" );
}
```

DATE

Date type can deal with dates and times. Date type has SQL_TIMESTAMP_STRUCT structure of ODBC.

```
typedef struct tagTIMESTAMP_STRUCT
{
        SQLSMALLINT    year;
        SQLUSMALLINT   month;
        SQLUSMALLINT   day;
        SQLUSMALLINT   hour;
        SQLUSMALLINT   minute;
        SQLUSMALLINT   second;
        SQLUINTEGER    fraction;
} TIMESTAMP_STRUCT;
typedef TIMESTAMP_STRUCT SQL_TIMESTAMP_STRUCT;
```

If the variable of DATE type is declared, the precompiler converts it to the structure above, and the meani ng of each field as same as SQL_TIMESTAMP_STRUCT of ODBC.

TIME

TIME type is the datatype which deals with time, and it has SQL_TIME_STRUCT structure of ODBC.

```
typedef struct tagTIME_STRUCT
{
        SQLUSMALLINT   hour;
        SQLUSMALLINT   minute;
        SQLUSMALLINT   second;
} TIME_STRUCT;
typedef TIME_STRUCT SQL_TIME_STRUCT;
```

If the variable of TIME type is declared, the precompiler converts it to the structure above, and the meaning of each field is as same as SQL_TIME_STRUCT of ODBC.

TIME WITH TIMEZONE

TIME WITH TIMEZONE type is the datatype which deals with the time with timezone, and it has the structure as follows.

```
typedef struct tagTIME_WITH_TIMEZONE_STRUCT
{
   SQLUSMALLINT hour;
   SQLUSMALLINT minute;
   SQLUSMALLINT second;
   SQLUINTEGER  fraction;
   SQLSMALLINT  timezone_hour;
   SQLSMALLINT  timezone_minute;
} TIME_WITH_TIMEZONE_STRUCT;
typedef TIME_WITH_TIMEZONE_STRUCT SQL_TIME_WITH_TIMEZONE_STRUCT;
```

If the variable of TIME WITH TIMEZONE type is declared, precompiler converts it to the structure above, and the meaning of each field is as follows.

Table 33-4 Fields of TIME WITH TIMEZONE

Field name	Description
hour	hour
minute	minute
second	second
fraction	the seconds below the decimal point
timezone_hour	hour of timezone
timezone_minute	minute of timezone

TIMESTAMP

TIMESTAMP type is the datatype which deals with date ~ time, and it has SQL_TIMESTAMP_STRUCT stru cture of ODBC.

```
typedef struct tagTIMESTAMP_STRUCT
{
        SQLSMALLINT    year;
        SQLUSMALLINT   month;
        SQLUSMALLINT   day;
        SQLUSMALLINT   hour;
        SQLUSMALLINT   minute;
        SQLUSMALLINT   second;
        SQLUINTEGER    fraction;
} TIMESTAMP_STRUCT;
typedef TIMESTAMP_STRUCT SQL_TIMESTAMP_STRUCT;
```

If the variable of TIMESTAMP type is declared, precompiler converts it to the structure above, and the me aning of each field as same as SQL_TIMESTAMP_STRUCT structure of ODBC.

TIMESTAMP WITH TIMEZONE

TIMESTAMP WITH TIMEZONE type is the datatype which deals with timestamp with timezone, and it has the following structure.

```
typedef struct tagTIMESTAMP_WITH_TIMEZONE_STRUCT
{
   SQLSMALLINT  year;
   SQLUSMALLINT month;
   SQLUSMALLINT day;
   SQLUSMALLINT hour;
   SQLUSMALLINT minute;
   SQLUSMALLINT second;
   SQLUINTEGER  fraction;
   SQLSMALLINT  timezone_hour;
   SQLSMALLINT  timezone_minute;
} TIMESTAMP_WITH_TIMEZONE_STRUCT;
typedef TIMESTAMP_WITH_TIMEZONE_STRUCT SQL_TIMESTAMP_WITH_TIMEZONE_STRUCT;
```

If the variable of TIMESTAMP WITH TIMEZONE type is declared, precompiler converts it to the structure a bove, and the meaning of each field is as follows.

Table 33-5 Fields of TIMESTAMP WITH TIMEZONE

Field name	Description
year	year
month	month
day	day
hour	hour
minute	minute
second	second
fraction	the seconds below the decimal point
timezone_hour	hour of timezone
timezone_minute	minute of timezone

The following is an example of dealing with DATE, TIME, TIMESTAMP, TIME WITH TIMEZONE, TIMESTAMP WITH TIMEZONE types.

```
/*
 * date_time.gc
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

EXEC SQL INCLUDE SQLCA;

#define  SUCCESS  0
#define  FAILURE  -1

#define  PRINT_SQL_ERROR(aMsg)                                          \
    {                                                                   \
        printf("\n");                                                   \
        printf(aMsg);                                                   \
        printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n",       \
               sqlca.sqlcode,                                           \
               SQLSTATE,                                                \
               sqlca.sqlerrm.sqlerrmc );                                \
    }

int Connect(char *aHostInfo, char *aUserID, char *sPassword);
int CreateTable();
int DropTable();
```

```
int main(int     argc,
         char  **argv)
{
    EXEC SQL BEGIN DECLARE SECTION;
    int                     sNo;
    int                     sResultNo;
    DATE                    sDate, sResultDate;
    TIME                    sTime, sResultTime;
    TIME WITH TIMEZONE      sTimeTz, sResultTimeTz;
    TIMESTAMP               sTimestamp, sResultTimestamp;
    TIMESTAMP WITH TIMEZONE  sTimestampTz, sResultTimestampTz;
    EXEC SQL END DECLARE SECTION;
    int  sState = 0;

    printf("#### Datatype Insert Test ####\n");
    printf("Connect GOLDILOCKS ...\n");
    if(Connect("DSN=GOLDILOCKS", "test", "test") != SUCCESS)
    {
        goto fail_exit;
    }
    sState = 1;
    printf("Create table ...\n");
    if(CreateTable() != SUCCESS)
    {
        goto fail_exit;
    }
    sState = 2;
    printf("Insert record ...\n");

    sNo = 1;
    /**
     * Date : 2014-7-14 21:29:30
     */
    sDate.year  = 2014;
    sDate.month = 7;
    sDate.day   = 14;
    sDate.hour  = 21;
    sDate.minute = 29;
    sDate.second = 30;
    sDate.fraction = 0;
```

```
/**
 * Time : 17:46:35
 */
sTime.hour   = 17;
sTime.minute = 46;
sTime.second = 35;

/**
 * Time With Timezone : 17:46:35.6789(+9:00)
 */
sTimeTz.hour             = 17;
sTimeTz.minute           = 46;
sTimeTz.second           = 35;
sTimeTz.fraction         = 678900000;
sTimeTz.timezone_hour    = 9;
sTimeTz.timezone_minute  = 0;

/**
 * Timestamp : 2014-02-13 17:46:28.123
 */
sTimestamp.year     = 2014;
sTimestamp.month    = 2;
sTimestamp.day      = 13;
sTimestamp.hour     = 17;
sTimestamp.minute   = 46;
sTimestamp.second   = 28;
sTimestamp.fraction = 123000000;

/**
 * Time With Timezone : 2014-05-18 17:46:35.001(+9:00)
 */
sTimestampTz.year     = 2014;
sTimestampTz.month    = 5;
sTimestampTz.day      = 18;
sTimestampTz.hour     = 17;
sTimestampTz.minute   = 46;
sTimestampTz.second   = 35;
sTimestampTz.fraction = 1000000;
sTimestampTz.timezone_hour    = 9;
sTimestampTz.timezone_minute  = 0;
```

```
    /**
     * Insert record
     */
    EXEC SQL
        INSERT INTO TEST_T1(C1, C2, C3, C4, C5, C6)
        VALUES(:sNo, :sDate, :sTime, :sTimeTz, :sTimestamp, :sTimestampTz);
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    printf("Retrive record\n");
    EXEC SQL
        SELECT C1, C2, C3, C4, C5, C6
        INTO   :sResultNo, :sResultDate, :sResultTime, :sResultTimeTz, :sResultTimestamp,
:sResultTimestampTz
        FROM   TEST_T1
        WHERE  C1 = 1;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    printf("================================================================\n");
    printf( "DATE                   : %04d-%02d-%02d %02d:%02d:%02d\n",
            sResultDate.year,
            sResultDate.month,
            sResultDate.day,
            sResultDate.hour,
            sResultDate.minute,
            sResultDate.second );

    printf( "TIME                   : %02d:%02d:%02d\n",
            sResultTime.hour,
```

```
        sResultTime.minute,
        sResultTime.second );

printf( "TIME WITH TIMEZONE     : %02d:%02d:%02d.%09u(GMT %+02d:%02d)\n",
        sResultTimeTz.hour,
        sResultTimeTz.minute,
        sResultTimeTz.second,
        sResultTimeTz.fraction,
        sResultTimeTz.timezone_hour,
        sResultTimeTz.timezone_minute );

printf( "TIMESTAMP              : %04d-%02d-%02d %02d:%02d:%02d.%09u\n",
        sResultTimestamp.year,
        sResultTimestamp.month,
        sResultTimestamp.day,
        sResultTimestamp.hour,
        sResultTimestamp.minute,
        sResultTimestamp.second,
        sResultTimestamp.fraction );

printf( "TIMESTAMP WITH TIMEZONE: %04d-%02d-%02d %02d:%02d:%02d.%09u(GMT %+02d:%02d)\n",
        sResultTimestampTz.year,
        sResultTimestampTz.month,
        sResultTimestampTz.day,
        sResultTimestampTz.hour,
        sResultTimestampTz.minute,
        sResultTimestampTz.second,
        sResultTimestampTz.fraction,
        sResultTimestampTz.timezone_hour,
        sResultTimestampTz.timezone_minute );

printf("================================================================\n");

sState = 0;
printf("Drop table ...\n");
if(DropTable() != SUCCESS)
{
    goto fail_exit;
}

sState = 0;
```

```
    printf("Disconnect GOLDILOCKS ...\n");
    EXEC SQL COMMIT WORK RELEASE;

    printf("SUCCESS\n");
    printf("############################\n");

    return 0;

  fail_exit:
    printf("\n");
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");

    printf("FAILURE\n");
    printf("############################\n\n");

    switch(sState)
    {
        case 1:
            printf("Drop table ...\n");
            (void)DropTable();
        default:
            break;
    }

    EXEC SQL ROLLBACK WORK RELEASE;

    return 0;
}

int CreateTable()
{
    EXEC SQL DROP TABLE IF EXISTS TEST_T1;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
```

- Create table

```
EXEC SQL CREATE TABLE TEST_T1 ( C1  INTEGER,
                                C2  DATE,
```

```
                                    C3  TIME,
                                    C4  TIME WITH TIME ZONE,
                                    C5  TIMESTAMP,
                                    C6  TIMESTAMP WITH TIME ZONE );
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");

    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}
```

- Drop table

```
int DropTable()
{
    EXEC SQL DROP TABLE TEST_T1;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
```

```
    return SUCCESS;

  fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");

    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}

int Connect(char *aHostInfo, char *aUserID, char *sPassword)
{
    EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR  sUid[80];
    VARCHAR  sPwd[20];
    VARCHAR  sConnStr[1024];
    EXEC SQL END DECLARE SECTION;
```

- Log on GOLDILOCKS

```
strcpy((char *)sUid.arr, aUserID);
    sUid.len = (short)strlen((char *)sUid.arr);
    strcpy((char *)sPwd.arr, sPassword);
    sPwd.len = (short)strlen((char *)sPwd.arr);
    strcpy((char *)sConnStr.arr, aHostInfo);
    sConnStr.len = (short)strlen((char *)sConnStr.arr);
```

- DB connection

```
EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:
    PRINT_SQL_ERROR("[ERROR] Connection Failure!");

    return FAILURE;
}
```

INTERVAL Types

INTERAVAL types are the datatypes which represent the time interval between two times. They are divide d into year ~ month family type and day ~ second family type. They are subdivided by specific types of ea ch family.
The following table describes the detailed classification of INTERVAL type.

Table 33-6 Classification of INTERVAL type

Family	Detailed type	Description
YEAR TO MONTH	INTERVAL YEAR	It is an interval in terms of years.
	INTERVAL MONTH	It is an interval in terms of months.
	INTERVAL YEAR TO MONTH	It is an interval in terms of years to months.
DAY TO SECOND	INTERVAL DAY	It is an interval in terms of days.
	INTERVAL HOUR	It is an interval in terms of hours.
	INTERVAL MINUTE	It is an interval in terms of minutes.
	INTERVAL SECOND	It is an interval in terms of seconds.
	INTERVAL DAY TO HOUR	It is an interval in terms of days to hours.
	INTERVAL DAY TO MINUTE	It is an interval in terms of days to minutes.
	INTERVAL DAY TO SECOND	It is an interval in terms of days to seconds.
	INTERVAL HOUR TO MINUTE	It is an interval in terms of hours to minutes.
	INTERVAL HOUR TO SECOND	It is an interval in terms of hours to seconds.
	INTERVAL MINUTE TO SECOND	It is an interval in terms of minutes to seconds.

INTERVAL type has the structure of SQL_INTERVAL_STRUCT of ODBC as follows.

```
typedef enum
{
    SQL_IS_YEAR      = 1,
    SQL_IS_MONTH     = 2,
    SQL_IS_DAY      = 3,
    SQL_IS_HOUR      = 4,
    SQL_IS_MINUTE     = 5,
    SQL_IS_SECOND     = 6,
    SQL_IS_YEAR_TO_MONTH   = 7,
    SQL_IS_DAY_TO_HOUR    = 8,
    SQL_IS_DAY_TO_MINUTE   = 9,
    SQL_IS_DAY_TO_SECOND   = 10,
    SQL_IS_HOUR_TO_MINUTE   = 11,
    SQL_IS_HOUR_TO_SECOND   = 12,
    SQL_IS_MINUTE_TO_SECOND   = 13
} SQLINTERVAL;
```

```
typedef struct tagSQL_YEAR_MONTH
{
    SQLUINTEGER  year;
    SQLUINTEGER  month;
} SQL_YEAR_MONTH_STRUCT;
typedef struct tagSQL_DAY_SECOND
{
    SQLUINTEGER  day;
    SQLUINTEGER  hour;
    SQLUINTEGER  minute;
    SQLUINTEGER  second;
    SQLUINTEGER  fraction;
} SQL_DAY_SECOND_STRUCT;
typedef struct tagSQL_INTERVAL_STRUCT
{
    SQLINTERVAL  interval_type;
    SQLSMALLINT  interval_sign;
    union {
        SQL_YEAR_MONTH_STRUCT  year_month;
        SQL_DAY_SECOND_STRUCT  day_second;
   } intval;
} SQL_INTERVAL_STRUCT;
```

All INTERVAL types listed above have the same structure. For each INTERVAL type, a valid field is separatel y distinguished within the structure. INTERVAL type is distinguished by interval_type field within the struct ure, and a real INTERVAL value is represented to intval union in the structure. The field which is used in in tval union is different by each type, and the following table describes the valid field depending on the typ e.

Table 33-7 The valid field depending on INTERVAL type

Type	?.interval_type	?.intval valid field
INTERVAL YEAR	SQL_IS_YEAR	*.year_month.year
INTERVAL MONTH	SQL_IS_MONTH	*.year_month.month
INTERVAL YEAR TO MONTH	SQL_IS_YEAR_TO_MONTH	*.year_month.year *.year_month.month
INTERVAL DAY	SQL_IS_DAY	*.day_second.day
INTERVAL HOUR	SQL_IS_HOUR	*.day_second.hour
INTERVAL MINUTE	SQL_IS_MINUTE	*.day_second.minute
INTERVAL SECOND	SQL_IS_SECOND	*.day_second.second *.day_second.fraction
INTERVAL DAY TO HOUR	SQL_IS_DAY_TO_HOUR	*.day_second.day

Type	?.interval_type	?.intval valid field
		*.day_second.hour
INTERVAL DAY TO MINUTE	SQL_IS_DAY_TO_MINUTE	*.day_second.day *.day_second.hour *.day_second.minute
INTERVAL DAY TO SECOND	SQL_IS_DAY_TO_SECOND	*.day_second.day *.day_second.hour *.day_second.minute *.day_second.second *.day_second.fraction
INTERVAL HOUR TO MINUTE	SQL_IS_HOUR_TO_MINUTE	*.day_second.hour *.day_second.minute
INTERVAL HOUR TO SECOND	SQL_IS_HOUR_TO_SECOND	*.day_second.hour *.day_second.minute *.day_second.second *.day_second.fraction
INTERVAL MINUTE TO SECOND	SQL_IS_MINUTE_TO_SECOND	*.day_second.minute *.day_second.second *.day_second.fraction

Fraction refers to the seconds below the decimal point. (The fraction field is valid only for INTERVAL type which includes SECOND.)

Special Type

The special data type provides the additional functionality and convenience for development of applicatio n rather than handling the data on its own. The following table describes the special types.

Table 33-8 Special types

Type	Description
SQL_CONTEXT	It manages run-time context within the multi-connection structure.
Struct	It constitutes a set of column as a structure, and it is used when dealing with ro w.
Typedef	The previously defined type is redefined as another name.

SQL_CONTEXT

SQL_CONTEXT is a special data type for managing the run-time context. Run-time context is used for the purpose of managing the connection and the individual data related to the connection on the run-time w hile the application is running.

SQL_CONTEXT variable is declared as follows.

```
EXEC SQL BEGIN DECLARE SECTION;
SQL_CONTEXT my_context;
EXEC SQL BEGIN DECLARE SECTION;
```

ALLOCATE clause is used after declaring SQL_CONTEXT variable as follows.

```
EXEC SQL CONTEXT ALLOCATE :my_context;
```

USE clause is used for SQL_CONTEXT variable.

```
EXEC SQL CONTEXT USE :my_context;
```

USE clause specifies the context to be used, and the following describes how to go back to the default co ntext not the context declared by the application.

```
EXEC SQL CONTEXT USE DEFAULT;
```

The disused SQL_CONTEXT variable is released as follows.

```
EXEC SQL CONTEXT FREE :my_context;
```

The following is an example of using SQL_CONTEXT.

```
/*
 * thread1.gc
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>

EXEC SQL INCLUDE SQLCA;

#define  SUCCESS  0
#define  FAILURE  -1

#define  PRINT_SQL_ERROR(aMsg)                                          \
    {                                                                   \
        printf("\n");                                                   \
        printf(aMsg);                                                   \
        printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n",   \
```

```
                sqlca.sqlcode,                                          \
                SQLSTATE,                                               \
                sqlca.sqlerrm.sqlerrmc );                               \
    }

int Connect(sql_context aCtx, char *aHostInfo, char *aUserID, char *sPassword);
int CreateEmpTempTable();
int DropEmpTempTable();
void *clientThread(void *args);

typedef struct thread_param
{
    int    mNo;
    char  *mJobName;
} thread_param;

#define  THREAD_COUNT    2
char gJobName[THREAD_COUNT][20]= {
    "RND",
    "SUPPORT"
};

int main(int argc, char **argv)
{
    EXEC SQL BEGIN DECLARE SECTION;
    int          sEmpNo;
    varchar      sEName[20 + 1];
    char         sJob[20];
    long         sSalary;
    EXEC SQL END DECLARE SECTION;
    int          sRecordCount = 0;
    pthread_t    thread_id[THREAD_COUNT];
    thread_param param[THREAD_COUNT];
    int          i;

    printf("Connect GOLDILOCKS ...\n");
    if(Connect(NULL, "DSN=GOLDILOCKS", "test", "test") != SUCCESS)
    {
        goto fail_exit;
    }
```

```
    if(CreateEmpTempTable() != SUCCESS)
    {
        goto fail_exit;
    }
```

- Create client thread

```
for( i = 0; i < THREAD_COUNT; i ++ )
    {
        param[i].mNo = i;
        param[i].mJobName = gJobName[i];
        if( pthread_create(&thread_id[i],
                           NULL,
                           clientThread,
                           &param[i]) != 0 )
        {
            printf( "Can't create thread %d!\n", i );
        }
        else
        {
            printf( "Create thread %d!\n", i );
        }
    }

    for( i = 0; i < THREAD_COUNT; i ++ )
    {
        if( pthread_join(thread_id[i],
                         NULL) != 0 )
        {
            printf( "Error when waiting for thread %d to terminate!\n", i );
        }
        else
        {
            printf( "Stopped thread %d!\n", i );
        }
    }
```

- Retrieve employee

```
EXEC SQL
        DECLARE EMP_CUR CURSOR FOR
        SELECT empno, ename, job, sal
```

```
    FROM   EMP_TEMP
    ORDER BY empno;

EXEC SQL OPEN EMP_CUR;
if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
}

printf(" EMPNO    ENAME                JOB      SALARY\n");
printf("====== ==================== ========== ========\n");
while( 1 )
{
    EXEC SQL
        FETCH EMP_CUR
        INTO  :sEmpNo, :sEName, :sJob, :sSalary;
    if(sqlca.sqlcode == SQL_NO_DATA)
    {
        break;
    }
    else if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    sRecordCount ++;

    printf("%6d %20s %10s %8ld\n",
           sEmpNo, sEName.arr, sJob, sSalary);
}

printf("====== ==================== ========== ========\n");
printf("Record Count = %d\n", sRecordCount);
printf("====== ==================== ========== ========\n");

EXEC SQL CLOSE EMP_CUR;
if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
```

```
        goto fail_exit;
    }

    if(DropEmpTempTable() != SUCCESS)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK RELEASE;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    printf("SUCCESS\n");
    printf("############################\n");

    return 0;

  fail_exit:

    printf("FAILURE\n");
    printf("############################\n\n");
    EXEC SQL ROLLBACK WORK RELEASE;

    return 0;
}

int Connect(sql_context aCtx, char *aHostInfo, char *aUserID, char *sPassword)
{
    EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR  sUid[80];
    VARCHAR  sPwd[20];
    VARCHAR  sConnStr[1024];
    EXEC SQL END DECLARE SECTION;
    struct sqlca sqlca;
```

- Log on GOLDILOCKS

```
strcpy((char *)sUid.arr, aUserID);
    sUid.len = (short)strlen((char *)sUid.arr);
    strcpy((char *)sPwd.arr, sPassword);
    sPwd.len = (short)strlen((char *)sPwd.arr);
    strcpy((char *)sConnStr.arr, aHostInfo);
    sConnStr.len = (short)strlen((char *)sConnStr.arr);
```

- DB connection

```
if( aCtx != NULL )
    {
        EXEC SQL CONTEXT USE :aCtx;
        EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
    }
    else
    {
        EXEC SQL CONTEXT USE DEFAULT;
        EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
    }
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] Connection Failure!");

    return FAILURE;
}

int Disconnect(sql_context aCtx)
{
    struct sqlca sqlca;
```

- DB disconnection

```
if( aCtx != NULL )
    {
```

```
        EXEC SQL CONTEXT USE :aCtx;
        EXEC SQL DISCONNECT;
    }
    else
    {
        EXEC SQL CONTEXT USE DEFAULT;
        EXEC SQL DISCONNECT;
    }
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] Connection Failure!");

    return FAILURE;
}
```

- Create table

```
int CreateEmpTempTable()
{
   EXEC SQL DROP TABLE IF EXISTS EMP_TEMP;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL
        CREATE TABLE EMP_TEMP (
            EMPNO NUMBER(4) CONSTRAINT PK_EMP_TEMP PRIMARY KEY,
            ENAME VARCHAR2(10),
            JOB VARCHAR2(9),
            SAL NUMBER(7,2),
            DEPTNO NUMBER(2) );
    if(sqlca.sqlcode != 0)
    {
```

```
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}
```

- Drop table

```
int DropEmpTempTable()
{
    EXEC SQL DROP TABLE EMP_TEMP;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL ROLLBACK WORK;
```

```
    return FAILURE;
}

void *clientThread(void *args)
{
    EXEC SQL BEGIN DECLARE SECTION;
    SQL_CONTEXT   my_context;
    char          job_name[20 + 1];
    EXEC SQL END DECLARE SECTION;
    int           state = 0;
    thread_param *param = (thread_param *)args;

    EXEC SQL CONTEXT ALLOCATE :my_context;
    state = 1;

    EXEC SQL CONTEXT USE :my_context;
    if(Connect(my_context, "DSN=GOLDILOCKS", "test", "test") != SUCCESS)
    {
        goto fail_exit;
    }
    state = 2;

    strcpy( job_name, param->mJobName );

    EXEC SQL
        INSERT INTO EMP_TEMP
        SELECT *
        FROM   EMP
        WHERE  JOB = :job_name;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    state = 1;
    if(Disconnect(my_context) != SUCCESS)
```

```
    {
        goto fail_exit;
    }
    state = 0;
    EXEC SQL CONTEXT FREE :my_context;
    pthread_exit(0);
    return NULL;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    switch(state)
    {
        case 2:
            (void)Disconnect(my_context);
        case 1:
            EXEC SQL CONTEXT FREE :my_context;
            break;
        default:
            break;
    }

    pthread_exit(0);
    return NULL;
}
```

SQL_CURSOR

The cursor variable is available for the query in the embedded SQL program. The cursor variable is a handle for the cursor which should be defined and opened by using PL/SQL in GOLDILOCKS server.

Declare the cursor variable in the embedded SQL by using SQL_CURSOR type. The following is an example of declaring the cursor variable.

```
EXEC SQL BEGIN DECLARE SECTION;
sql_cursor   emp_cursor;
SQL_CURSOR   sal_cursor;
sql_cursor * cur_ptr;
EXEC SQL END DECLARE SECTION;
cur_ptr = &emp_cursor;
```

The cursor variable is available by declaring with sql_cursor or SQL_CURSOR. The cursor variable follows the C range rules like as other host variables. It is transferable as the function parameter. It can also define

the cursor variable or the function which returns the pointer for the cursor variable. The following is an example of using the cursor variable as the function parameter and return.

```
SQL_CURSOR * alloc_cursor( sql_cursor * emp_cursor )
{
EXEC SQL BEGIN DECLARE SECTION;
    sql_cursor * ret_cursor;
EXEC SQL END DECLARE SECTION;
    ret_cursor = emp_cursor;
....
    return ret_cursor;
}
```

The cursor variable should be allocated before using it. Use **ALLOCATE** command after declaring the cursor variable to perform this operation. The following is an example of allocating ret_cursor which was used in alloc_cursor function in the example above.

```
EXEC SQL ALLOCATE :ret_cursor;
```

If the cursor variable is allocated, then the memory is allocated during the run-time, so it should be freed by using **FREE** command. The memory which was created when allocating the cursor variable is freed when CLOSE command is explicitly used or the connection is closed.

```
EXEC SQL FREE :ret_cursor;
```

The cursor variable should be opened in GOLDILOCKS database server. The embedded SQL command OPEN can not be used to opes the cursor variable. The cursor variable can be opened by calling the storing procedure or the function of PL/SQL opening the cursor, or by defining the anonymous block opening the cursor.

The following is an example of opening the cursor variable by calling the storing procedure of PL/SQL.

```
CREATE OR REPLACE PROCEDURE empProc( curs IN OUT SYS_REFCURSOR,
                                     dept IN INTEGER )
AS
BEGIN
   OPEN curs FOR SELECT ename FROM emp
                         WHERE deptno = dept ORDER BY ename;
END;
/
```

Define empProc storing procedure and call empProc storing procedure in the embedded SQL to open and FETCH the cursor variable as the example above.

The following is an example.

```
EXEC SQL BEGIN DECLARE SECTION;
SQL_CURSOR empCursor;
int  dept;
char ename[128];
EXEC SQL END DECLARE SECTION;
EXEC SQL ALLOCATE :empCursor;
EXEC SQL CALL empProc( :empCursor,  :dept );
while( 1 )
{
    EXEC SQL FETCH :empCursor INTO :ename;
    if( sqlca.sqlcode == 100 ) break;
    printf( "%s\n", ename );
}
```

The following is an example of defining the storing procedure and opening the cursor variable in the emb edded SQL.

```
EXEC SQL BEGIN DECLARE SECTION;
SQL_CURSOR empCursor;
int  dept;
char ename[128];
EXEC SQL END DECLARE SECTION;
EXEC SQL ALLOCATE :empCursor;
EXEC SQL EXECUTE
    BEGIN
        OPEN :empCursor FOR SELECT ename FROM emp
                                    WHERE dept_no = :dept ORDER BY ename;
    END;
END-EXEC;
while( 1 )
{
    EXEC SQL FETCH :empCursor INTO :ename;
    if( sqlca.sqlcode == 100 ) break;
    printf( "%s\n", ename );
}
```

The following is an example of closing the cursor variable through CLOSE command.

```
EXEC SQL CLOSE :empCursor;
```

When closing the cursor variable, the cursor is closed but it does not return the memory. The memory for the cursor variable can be returned through FREE command.

Allocating the cursor variable internally allocates the client's statement, so it is related to the co ntext. Therefore, open the cursor after allocating the cursor variable, then FETCH, CLOSE and F REE are operable within the same context only. For example, if the cursor variable is allocated in the default context and used in another named context then an error occurs.

Host Structure

The C structure can be used as a host variable in an embedded SQL of GOLDILOCKS. A typical scalar varia ble represents a single column, but using the structure enables to simply express multiple column sets.

Host structure can only be used only in INTO clause of SELECT INTO or FETCH INTO statement and in VAL UES clause of INSERT statement, but it can not be used in WHERE clause or UPDATE SET clause, because it has the same effect as when listing its member variables in sequence.

The structure is defined in the declare section to use the host structure, and it can be used as a host varia ble after declaring the structure variable. The structure is defined in the same way as defining the C struct, and it can also be used after defining the type via typedef.

The following is an example of using the host structure.

```
/*
 * sample4.gc
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

EXEC SQL INCLUDE SQLCA;

#define  SUCCESS  0
#define  FAILURE  -1
#define  PRINT_SQL_ERROR(aMsg)                                    \
    {                                                             \
        printf("\n");                                             \
        printf(aMsg);                                             \
        printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n",  \
               sqlca.sqlcode,                                     \
```

```
                SQLSTATE,                                               \
                sqlca.sqlerrm.sqlerrmc );                               \
    }
EXEC SQL BEGIN DECLARE SECTION;
typedef struct rsRecord
{
    int          mEmpNo;
    varchar      mEName[20 + 1];
    char         mJob[20 + 1];
    long         mSalary;
} rsRecord;
EXEC SQL END DECLARE SECTION;

int Connect(char *aHostInfo, char *aUserID, char *sPassword);

int main(int argc, char **argv)
{
    EXEC SQL BEGIN DECLARE SECTION;
    rsRecord     sRecord;
    EXEC SQL END DECLARE SECTION;
    int  sRecordCount = 0;

    printf("Connect GOLDILOCKS ...\n");
    if(Connect("DSN=GOLDILOCKS", "test", "test") != SUCCESS)
    {
        goto fail_exit;
    }
```

- Retrieve employee

```
EXEC SQL
        DECLARE EMP_CUR CURSOR FOR
        SELECT empno, ename, job, sal
        FROM   EMP
        ORDER BY EMPNO;

    EXEC SQL OPEN EMP_CUR;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
```

```
    }

    printf(" EMPNO    ENAME                JOB      SALARY\n");
    printf("====== ==================== ========== ========\n");
    while( 1 )
    {
        EXEC SQL
            FETCH EMP_CUR
            INTO  :sRecord;
        if(sqlca.sqlcode == SQL_NO_DATA)
        {
            break;
        }
        else if(sqlca.sqlcode != 0)
        {
            PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
            goto fail_exit;
        }

        sRecordCount ++;

        printf("%6d %20s %10s %8ld\n",
               sRecord.mEmpNo, sRecord.mEName.arr, sRecord.mJob, sRecord.mSalary);
    }

    printf("====== ==================== ========== ========\n");
    printf("Record Count = %d\n", sRecordCount);
    printf("====== ==================== ========== ========\n");

    EXEC SQL CLOSE EMP_CUR;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK RELEASE;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
```

```
    }

    printf("\n\nSUCCESS\n");
    printf("############################\n");

    return 0;

  fail_exit:

    printf("\n\nFAILURE\n");
    printf("############################\n\n");

    EXEC SQL ROLLBACK WORK RELEASE;

    return 0;
}

int Connect(char *aHostInfo, char *aUserID, char *sPassword)
{
    EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR  sUid[80];
    VARCHAR  sPwd[20];
    VARCHAR  sConnStr[1024];
    EXEC SQL END DECLARE SECTION;
```

- Log on GOLDILOCKS

```
strcpy((char *)sUid.arr, aUserID);
    sUid.len = (short)strlen((char *)sUid.arr);
    strcpy((char *)sPwd.arr, sPassword);
    sPwd.len = (short)strlen((char *)sPwd.arr);
    strcpy((char *)sConnStr.arr, aHostInfo);
    sConnStr.len = (short)strlen((char *)sConnStr.arr);
```

- DB connection

```
EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
```

```
    return SUCCESS;

  fail_exit:
    PRINT_SQL_ERROR("[ERROR] Connection Failure!");

    return FAILURE;
}
```

When a user declares a structure for using the host variable, the restriction is as follows. The nested structure is not allowed. If another structure exists within the structure declaration as shown below, it can not be used as the host variable.

```
EXEC SQL BEGIN DECLARE SECTION;
typedef struct rsRecord
{
    struct person {
        int          mEmpNo;
        varchar      mEName[20 + 1];
    } person;
    char         mJob[20 + 1];
    long         mSalary;
} rsRecord;
EXEC SQL END DECLARE SECTION;
```

Indicator Variable

Scalar Indicator

Host variables can be used together with the indicator variables combined with it. An indicator variable determines whether the current value of the host variable is NULL. The indicator variable is declared in the same way as the host variable, and it is declared only as an integer type of C (short, int, long, long long). Indicator is used as follows.

```
:hostvar INDICATOR :hostind
:hostvar :hostind (INDICATOR keyword can be omitted.)
```

The value of indicator variable means the followings.

Table 33-9 Value of input indicator

Value	Meaning
-1	NULL

Value	Meaning
>= 0	It inputs a host variable value.

Table 33-10 Value of output indicator

Value	Meaning
-1	NULL
0	All values are stored in a host variable.
> 0	It is the length of DB data when all values are not stored in a host variable due to an insufficient buffer size of host variable.

The following is an example of using the indicator.

```
/*
 * sample5.gc
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

EXEC SQL INCLUDE SQLCA;

#define  SUCCESS  0
#define  FAILURE  -1
#define  PRINT_SQL_ERROR(aMsg)                                          \
    {                                                                   \
        printf("\n");                                                   \
        printf(aMsg);                                                   \
        printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n",       \
               sqlca.sqlcode,                                           \
               SQLSTATE,                                                \
               sqlca.sqlerrm.sqlerrmc );                                \
    }

int Connect(char *aHostInfo, char *aUserID, char *sPassword);
int CreateEmpTempTable();
int DropEmpTempTable();

int main(int argc, char **argv)
{
    EXEC SQL BEGIN DECLARE SECTION;
```

```
    int          sEmpNo;
    varchar      sEName[20 + 1];
    char         sJob[20];
    long         sSalary;
    int          sDeptNo;
    int          sDeptNoInd;
    EXEC SQL END DECLARE SECTION;
    int          sRecordCount = 0;
    int          state = 0;

    printf("Connect GOLDILOCKS ...\n");
    if(Connect("DSN=GOLDILOCKS", "test", "test") != SUCCESS)
    {
        goto fail_exit;
    }

    if(CreateEmpTempTable() != SUCCESS)
    {
        goto fail_exit;
    }
    state = 1;
```

- Update Dept NULL where deptno = 10

```
EXEC SQL
        UPDATE EMP_TEMP
        SET    DEPTNO = NULL
        WHERE  DEPTNO = 10;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }
```

- Retrieve employee

```
EXEC SQL
        DECLARE EMP_CUR CURSOR FOR
        SELECT empno, ename, job, sal, deptno
        FROM   EMP_TEMP
        ORDER BY empno;
```

```
EXEC SQL OPEN EMP_CUR;
if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
}

printf(" EMPNO         ENAME           JOB      SALARY  DEPTNO\n");
printf("====== ==================== ========== ======== ======\n");
while( 1 )
{
    EXEC SQL
        FETCH EMP_CUR
        INTO  :sEmpNo, :sEName, :sJob, :sSalary, :sDeptNo :sDeptNoInd;
    if(sqlca.sqlcode == SQL_NO_DATA)
    {
        break;
    }
    else if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    sRecordCount ++;

    if(sDeptNoInd == -1)
    {
        printf("%6d %20s %10s %8ld (null)\n",
               sEmpNo, sEName.arr, sJob, sSalary);
    }
    else
    {
        printf("%6d %20s %10s %8ld %4d\n",
               sEmpNo, sEName.arr, sJob, sSalary, sDeptNo);
    }
}

printf("====== ==================== ========== ======== ======\n");
printf("Record Count = %d\n", sRecordCount);
printf("====== ==================== ========== ======== ======\n");
```

```
    EXEC SQL CLOSE EMP_CUR;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    state = 0;
    if(DropEmpTempTable() != SUCCESS)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK RELEASE;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    printf("\n\nSUCCESS\n");
    printf("############################\n");

    return 0;

fail_exit:

    printf("\n\nFAILURE\n");
    printf("############################\n\n");

    switch( state )
    {
        case 1:
            (void)DropEmpTempTable();
            break;
        default:
            break;
    }

    EXEC SQL ROLLBACK WORK RELEASE;
```

```
    return 0;
}

int Connect(char *aHostInfo, char *aUserID, char *sPassword)
{
    EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR  sUid[80];
    VARCHAR  sPwd[20];
    VARCHAR  sConnStr[1024];
    EXEC SQL END DECLARE SECTION;
```

- Log on GOLDILOCKS

```
strcpy((char *)sUid.arr, aUserID);
    sUid.len = (short)strlen((char *)sUid.arr);
    strcpy((char *)sPwd.arr, sPassword);
    sPwd.len = (short)strlen((char *)sPwd.arr);
    strcpy((char *)sConnStr.arr, aHostInfo);
    sConnStr.len = (short)strlen((char *)sConnStr.arr);
```

- DB connection

```
EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] Connection Failure!");

    return FAILURE;
}
```

- Create table

```
int CreateEmpTempTable()
{
    EXEC SQL DROP TABLE IF EXISTS EMP_TEMP;
```

```
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL
        CREATE TABLE EMP_TEMP (
            EMPNO NUMBER(4) CONSTRAINT PK_EMP_TEMP PRIMARY KEY,
            ENAME VARCHAR2(10),
            JOB VARCHAR2(9),
            SAL NUMBER(7,2),
            DEPTNO NUMBER(2) );
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL
        INSERT INTO EMP_TEMP
        SELECT * FROM EMP;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");

    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}
```

- Drop table

```
int DropEmpTempTable()
{
    EXEC SQL DROP TABLE EMP_TEMP;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}
```

Structure Indicator

If a host variable is a scalar variable, an Indicator variable is declared and combined with the host variable, then it is used. If the host variable is the structure, each variable configuring the structure can not be use d with an indicator variable. In this case, the indicator should also be the structure.

When declaring an indicator structure, it should comply with the followings.

- The number of indicator members should be as same as the number of host variable structure memb ers to be combined.
- All members of the indicator structure should have the integer data type.

For example, if the following structure is declared, the member of indicator structure should be four beca use the number of the variables of the structure is four.

```
EXEC SQL BEGIN DECLARE SECTION;
typedef struct rsRecord
{
    int          mEmpNo;
    varchar      mEName[20 + 1];
    char         mJob[20 + 1];
    long         mSalary;
} rsRecord;
EXEC SQL END DECLARE SECTION;
```

Therefore, it should be declared as follows.

```
EXEC SQL BEGIN DECLARE SECTION;
typedef struct rsRecordInd
{
    int          mEmpNoInd;
    int          mENameInd;
    int          mJobInd;
    int          mSalaryInd;
} rsRecordInd;
EXEC SQL END DECLARE SECTION;
```

Each indicator structure member sequentially corresponds one-to-one to the host variable structure mem ber which is combined to the indicator structure. When declaring as above, the combination relationship is as follows.

```
:rsRecordVar INDICATOR :rsRecordIndVar
```

Host variable	Indicator to be combined
rsRecordVar.mEmpNo	rsRecordIndVar.mEmpNoInd
rsRecordVar.mEName	rsRecordIndVar.mENameInd
rsRecordVar.mJob	rsRecordIndVar.mJobInd
rsRecordVar.mSalary	rsRecordIndVar.mSalaryInd

Embedded SQL

Host Variable

The host variable is used as the data input/output medium between the application and GOLDILOCKS. The input host variable transfers the data from the application to GOLDILOCKS, and the output host variable transfers the data from GOLDILOCKS to the application. The declarations of two variables are same and their roles are determined in the SQL statement in use.

The host variable which is located in INTO clause of SELECT or FETCH statement is the output host variable received the data from GOLDILOCKS. Other than that is the input host variable. The input host variable should be set before executing the SQL statement.

Host Indicator

The host variable literally uses the variable of C language, so a particular way to display NULL does not exist. The indicator variable can be used for that, and a single indicator is combined with a single host variable.

An indicator variable means the followings.

Table 33-11 Value of input indicator

Value	Meaning
-1	NULL
>= 0	It inputs a host variable value.

Table 33-12 Value of output indicator

Value	Meaning
-1	NULL
0	All values are stored in a host variable.
> 0	It is the length of DB data when all values are not stored in a host variable due to an insufficient buffer size of host variable.

Insert NULL

NULL is inserted in a column as follows.

```
EXEC SQL INSERT INTO EMP ( EMPNO, DEPTNO ) VALUES ( :empno, NULL );
```

However, if hard coding above is used to create the application, the flexibility becomes very poor. The following is an example of using the indicator variable.

```
deptno_ind = -1;
EXEC SQL INSERT INTO EMP ( EMPNO, DEPTNO ) VALUES ( :empno, :deptno :deptno_ind );
```

If the value of deptno_ind which is the indicator variable is -1, it is recognized as NULL regardless of the v alue of deptno which is the host variable.

Fetch NULL

When GOLDILOCKS receives the data, the indicator variable can be used to determine whether to be NUL L. The following is an example.

```
EXEC SQL DECLARE CUR_1 CURSOR FOR
    SELECT  EMPNO, DEPTNO
    FROM    EMP
    WHERE   EMPNO = :emp_number;
EXEC SQL OPEN CUR_1;

while( 1 )
{
    EXEC SQL FETCH CUR_1 INTO :empno, :deptno :deptno_ind;
    if( sqlca.sqlcode == SQL_NO_DATA )
    {
        break;
    }

    if( deptno_ind == -1 )
    {
        printf( "empno : %d, deptno : (null)\n", empno );
    }
    else
    {
        printf( "empno : %d, deptno : %d\n", empno, deptno );
    }
}

EXEC SQL CLOSE CUR_1;
```

If deptno_ind is -1 after performing FETCH, it determines that deptno is NULL.

Basic SQL Statement

All SQL statements provided in GOLDILOCKS is available in an embedded SQL. For more information abo ut SQL statements, refer to **Part III. SQL Manual**. When using an SQL statement in an embedded SQL, the SQL statement is specified after EXEC SQL keyword. This chapter describes Data Definition Language (DD L) or Data Manipulation Language (DML), and the next chapter describes the statements repeatedly retri

eving data as like a query.

SQLCA is checked to determine whether the SQL statement is successfully executed. For more informatio n, refer to **Handling Run-time Errors**.

DDL Statement

DDL statement is an SQL statement which creates, drops or alters GOLDILOCKS objects such as table, vie w, index. When executing a DDL statement in an embedded SQL application, the SQL statement is specifi ed after EXEC SQL keyword.

```
EXEC SQL DROP TABLE IF EXISTS EMP;
EXEC SQL
    CREATE TABLE EMP (
        EMPNO NUMBER(4) CONSTRAINT PK_EMP PRIMARY KEY,
        ENAME VARCHAR2(10),
        JOB VARCHAR2(9),
        SAL NUMBER(7,2),
        DEPTNO NUMBER(2) );
```

The host variable can not be used in DDL statement. Therefore, the following usage is wrong.

```
strcpy( table_name, "T1" );
EXEC SQL CREATE TABLE :table_name ( C1 INTEGER );
```

If DDL statement is to be used variably as above because the DDL statement is not defined while creating the application, the dynamic SQL statement can be used as follows.

```
strcpy( table_name, "T1" );
sprintf( sql_stmt, "CREATE TABLE %s ( C1 INTEGER )", table_name );
EXEC SQL EXECUTE IMMEDIATE :sql_stmt;
```

For more information, refer to **Embedded Dynamic SQL**.

Select Into Statement

The query is used to retrieve the data from GOLDILOCKS. Generally, the number of retrieved rows are not known, so the cursor object is declared and the process of open, fetch, close may be performed to use th e query statement. For more information, refer to **Cursor**.

However, the number of retrieved rows are known in a special case. For example, when the primary key i s known and the record with the same primary key is retrieved, then it can be predicted that the result eit her does not exist or has up to one record. The select into statement is used when the number of the res ult records is less than one as follows.

```
EXEC SQL
    SELECT ename, job, sal
    INTO :emp_name, :job_title, :salary
    FROM emp
    WHERE empno = :emp_number;
```

Several records can be retrieved by using select into statement when using host array. For more information about host array, refer to **Host Array**.

Insert Statement

The insert statement is used to insert rows into a table. The column value can be determined by using the host variable, and NULL can be inserted by using an indicator.

The following is an example of the insert statement.

```
EXEC SQL
    INSERT INTO EMP ( empno, ename, job, sal )
    VALUES ( :emp_number, :emp_name, :job_name :job_ind, :saraly );
EXEC SQL
    INSERT INTO DEPT ( deptno, dname, loc )
    VALUES ( 1, :dept_name, NULL );
```

When using host structure, data can be inserted in structure units instead of individually using h ost variable.

```
EXEC SQL BEGIN DECLARE SECTION;
typedef struct rsRecord
{
    int          mEmpNo;
    varchar      mEName[20 + 1];
    char         mJob[20 + 1];
    long         mSalary;
} rsRecord;

rsRecord  sInsertRec;
EXEC SQL END DECLARE SECTION;

sInsertRec.mEmpNo = 3000;
strcpy( sInsertRec.mEName.arr, "John" );
```

```
sInsertRec.mEName.len = strlen( sInsertRec.mEName.arr );
strcpy( sInsertRec.mJob , "RND" );
sInsertRec.mSalary = 3500;

EXEC SQL INSERT INTO EMP ( empno, ename, job, sal ) VALUES ( :sInsertRec );
```

Multiple rows can be inserted by using host array at once. For more information, refer to **Host Array**.

Update Statement

The update statement is used to update the column values of the specific rows in a table. The column value can be determined by using the host variable, and NULL can be inserted by using an indicator.
The following is an example of the update statement.

```
EXEC SQL
    UPDATE emp
    SET sal = :salary, deptno = :dept_number :deptno_ind
    WHERE empno = :emp_number;
```

Multiple rows can be updated by using host array at once. For more information, refer to **Host Array**.

Delete Statement

The delete statement removes the specific rows from a table.
The following is an example of the delete statement.

```
EXEC SQL
    DELETE FROM emp
    WHERE empno = :emp_number;
```

Multiple rows can be deleted by using host array at once. For more information, refer to **Host Array**.

PSM Statement

PSM statement creates a procedure or a function within a server and uses it.
For more information about PSM, refer to **PSM manual**.

Generally, it starts with EXEC SQL EXECUTE and ends with END-EXEC;. However, when creating a procedure or a function, EXEC SQL is used instead of EXEC SQL EXECUTE.

The following is an example of a statement creating and calling a procedure.

```
EXEC SQL
    CREATE OR REPLACE PROCEDURE PROC1( A1 INTEGER, A2 INTEGER )
      IS
        V1 INTEGER;
      BEGIN
        SELECT COUNT(*)
          INTO V1
          FROM T1
          WHERE T1.I1 >= A1 AND T1.I1 <= A2;
        DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
      END;
END-EXEC;
EXEC SQL CALL PROC1( 2, 4 );
```

The following is an example of a statement creating and calling a function.

```
EXEC SQL BEGIN DECLARE SECTION;
    int         sV1 = 0;
EXEC SQL END DECLARE SECTION;
EXEC SQL
    CREATE OR REPLACE FUNCTION FUNC1( A1 INTEGER, A2 INTEGER )
      RETURN INTEGER
      IS
        V1 INTEGER;
      BEGIN
        SELECT COUNT(*)
          INTO V1
          FROM T1
          WHERE T1.I1 >= A1 AND T1.I1 <= A2;
        RETURN V1;
      END;
END-EXEC;
EXEC SQL CALL FUNC1( 2, 4 ) INTO :sV1;
```

The following is an example of an anonymous block statement.

```
EXEC SQL EXECUTE
    DECLARE
      V1 INTEGER := 0;
      FUNCTION FUNC1( A1 INTEGER )
        RETURN INTEGER
        IS
        BEGIN
          RETURN A1 * 10;
        END;
    BEGIN
      V1 := FUNC1( 10 );
      DBMS_OUTPUT.PUT_LINE( 'V1 = ' || V1 );
    END;
END-EXEC;
```

```
EXEC SQL EXECUTE
    DECLARE
      PROCEDURE PROC1( A1 INTEGER )
      IS
      BEGIN
        DBMS_OUTPUT.PUT_LINE( 'A1 = ' || A1 );
      END;
    BEGIN
      PROC1( 100 );
    END;
END-EXEC;
```

Cursor

The application executes the query to retrieve data from GOLDILOCKS. Generally, the cursor is used beca use the number of rows retrieved are not known when executing the query. The cursor is an identifier wh ich specifies the position of the current row in the query result set. The cursor can be manipulated via the following operations.

Declare Cursor

It declares a cursor. The cursor name and its query should be specified. The declared cursor name is used for different cursor manipulation commands later.

The following is an example of the cursor declaration.

```
EXEC SQL
    DECLARE RECORD_CUR1 CURSOR FOR
    SELECT   empno, ename, dept
    FROM     SEMP
    ORDER BY empno;
```

The cursor name is an identifier recognized and used by the precompiler, and it has nothing to do with th e variable of C program. The scope of the cursor is a file. Therefore, even when the cursor with the same name is defined in another file, it is considered as a separate cursor. In other words, all statements which perform the operations such as declare/ open/ fetch/ close of the cursor should exist on a single source fil e.

Open Cursor

It opens a cursor.

```
EXEC SQL OPEN <cursor_name>;
Example)
EXEC SQL OPEN EMP_CURSOR;
```

If the cursor is opened, it is prepared to fetch the result set which the query of the declared cursor is exec uted. However, it does not actually fetch the result. **Fetch Cursor** should be performed to actually fetch th e data.
The host variable which is used when executing the query does not affect the result set until the current c ursor is closed.

```
EXEC SQL DECLARE EMP_CURSOR CURSOR FOR
        SELECT    empno, ename, dept
        FROM      EMP
        WHERE     empno < :sNo
        ORDER BY  empno;

    sNo = 100;
    EXEC SQL OPEN EMP_CURSOR;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    while( 1 )
    {
        sNo = 10;
```

```
        EXEC SQL FETCH EMP_CURSOR INTO :emp_number, :emp_name, :dept_name;
        if(sqlca.sqlcode == SQL_NO_DATA)
        {
            break;
        }
        else if(sqlca.sqlcode != 0)
        {
            goto fail_exit;
        }
        ...
    }
```

In the example above, even if the cursor is opened with sNo = 100 and sNo is changed during the proces sing, it does not affect the result set until the cursor is closed.

Fetch Cursor

It fetches the row at the cursor position.

```
EXEC SQL FETCH <cursor_name> INTO <host_variable_list>;
Example)
EXEC SQL FETCH EMP_CURSOR INTO :emp_number, :emp_name, :dept_name;
```

The cursor should be declared first for FETCH operation, and it should be open. If FETCH is performed for the first time, the cursor moves to the first row of the result set and specifies it as a current row. Then, it f etches the current row into the host variable of INTO clause and returns. After then, if FETCH is repeatedl y performed, the cursor updates the current row to the next row, and then it repeatedly fetches the curre nt row into the host variable of INTO clause and returns. If the result does not exist even after performing FETCH, SQL_NO_DATA code is returned in sqlca.sqlcode, and the application checks this code to determi ne whether this operation is terminated.

Close Cursor

It closes a cursor.

```
EXEC SQL CLOSE <cursor_name>;
Example)
EXEC SQL CLOSE EMP_CURSOR;
```

The cursor should already be open to close it. The FETCH can not be performed in this cursor after closing the cursor. If the cursor is opened to use it again after it is close, then it becomes a new cursor. Therefore, the result set of the new cursor may be different from the result set of the previously closed cursor.

Reusing Cursor

The cursor name which was already declared is repeatedly reusable. This paragraph describes the sequence to reuse the same cursor name.

Sequence of Cursor Statement

- DECLARE CURSOR
  It can be performed after performing cursor CLOSE, COMMIT and ROLLBACK statements.
- OPEN
  It can be performed when the cursor is completely fetched or after performing CLOSE statement.
- FETCH
  It can be performed after performing cursor OPEN statement.
- CLOSE
  It can be performed after performing cursor OPEN statement or FETCH statement.

Cursor Statement and Host Variable

Cursor statement can be located in any place within a file. However, the usage of the cursor statement varies depending on whether the input host variable used in DECLARE statement is a global variable or a local variable.

- If the input host variable used in DECLARE statement is a local variable, then OPEN statement should be located in the function where DECLARE statement is located.
- If the input host variable used in DECLARE statement is a global variable, then OPEN statement can be located in the function which is different from where DECLARE statement is located.
- If the input host variable does not exist in DECLARE statement, then OPEN statement can be located in the function which is different from where DECLARE statement is located.

The reason is that the pointer of the host variable used in cursor declare statement is implicitly stored, and the pointer is used in cursor open statement, so if two statements are located in different functions and used host variables are local variables, then open statement refers to the invalid pointer. Therefore, it is recommended to locate both declare statement and open statement within the same function.

Example of Using Cursor

The following is an example of using DDL, DML and cursor.

```
/*
 * overview.gc
 *
 * Connect / Disconnect
 * DDL(Create/Drop table)
```

```
 * Basic DML(Insert, Delete, Update)
 * Standing Cursor
 */
EXEC SQL INCLUDE SQLCA;

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define  SUCCESS  0
#define  FAILURE  -1
#define  PRINT_SQL_ERROR(aMsg)                                          \
    {                                                                   \
        printf("\n");                                                   \
        printf(aMsg);                                                   \
        printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n",   \
               sqlca.sqlcode,                                           \
               SQLSTATE,                                                \
               sqlca.sqlerrm.sqlerrmc );                                \
    }

EXEC SQL BEGIN DECLARE SECTION;
typedef struct rsEmpRecord
{
    int      mEmpNo;
    char     mEName[20];
    char     mDept[10];
} rsEmpRecord;

rsEmpRecord gRecord[10] = {
    { 1, "Park", "RND" },
    { 2, "Kim", "CEO" },
    { 3, "Choi", "SALES" },
    { 4, "Lee", "CTO" },
    { 5, "Lyu", "RND" },
    { 6, "Ohn", "SUPPORT" },
    { 7, "Cheon", "RND" },
    { 8, "Sohn", "SALES" },
    { 9, "Smith", "WAIT" },
    { 10, "mycomman", "WAIT" }
};
```

```
EXEC SQL END DECLARE SECTION;

int CreateEmpTable();
int DropEmpTable();
int Connect(char *aHostInfo, char *aUserID, char *sPassword);
int Disconnect();
int PrintRecord();

int main(int argc, char **argv)
{
    EXEC SQL BEGIN DECLARE SECTION;
    int          sEmpNo;
    char         sDept[10 + 1];
    EXEC SQL END DECLARE SECTION;
    int  sState = 0;

    printf("Connect GOLDILOCKS ...\n");
    if(Connect("DSN=GOLDILOCKS", "test", "test") != SUCCESS)
    {
        goto fail_exit;
    }
    sState = 1;

    printf("Create SEMP table ...\n");
    if(CreateEmpTable() != SUCCESS)
    {
        goto fail_exit;
    }
    sState = 2;

    printf("Insert record ...\n");
    EXEC SQL
        INSERT INTO SEMP(empno, ename, dept)
        VALUES(:gRecord);
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }
    printf("====== ==================== ==========\n");
    printf("%d Record Inserted\n", sqlca.sqlerrd[2]);
```

```
printf("====== ==================== ==========\n");

EXEC SQL COMMIT WORK;
if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
}

printf("Current record\n");
PrintRecord();

sEmpNo = 9;
printf("Delete record WHERE empno == %d\n", sEmpNo);
EXEC SQL
    DELETE FROM SEMP
    WHERE  empno = :sEmpNo;
if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
}

printf("After Delete record\n");
PrintRecord();

EXEC SQL COMMIT WORK;
if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
}

strcpy( sDept, "RND" );
printf("Update record WHERE dept == 'WAIT'\n");
EXEC SQL
    UPDATE SEMP
    SET    dept = :sDept
    WHERE  dept = 'WAIT';
if(sqlca.sqlcode != 0)
{
```

```
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    printf("After Update record\n");
    PrintRecord();

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    sState = 1;
    printf("Drop SEMP table ...\n");
    if(DropEmpTable() != SUCCESS)
    {
        goto fail_exit;
    }

    sState = 0;
    printf("Disconnect GOLDILOCKS ...\n");
    if(Disconnect() != SUCCESS)
    {
        goto fail_exit;
    }

    printf("SUCCESS\n");
    printf("############################\n");

    return 0;

fail_exit:

    printf("FAILURE\n");
    printf("############################\n\n");

    EXEC SQL ROLLBACK WORK;
    switch(sState)
    {
```

```
        case 2:
            printf("Drop SEMP table ...\n");
            (void)DropEmpTable();
        case 1:
            printf("Disconnect GOLDILOCKS ...\n");
            (void)Disconnect();
            break;
        default:
            break;
    }

    return 0;
}
```

- Create table

```
int CreateEmpTable()
{
    EXEC SQL DROP TABLE IF EXISTS SEMP;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL CREATE TABLE SEMP ( empno        INTEGER,
                                 ename        VARCHAR(20),
                                 dept         VARCHAR(10),
                                 PRIMARY KEY (empno) );
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;
```

```
  fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");

    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}
```

- Drop table

```
int DropEmpTable()
{
    EXEC SQL DROP TABLE SEMP;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");

    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}

int Connect(char *aHostInfo, char *aUserID, char *sPassword)
{
    EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR  sUid[80];
    VARCHAR  sPwd[20];
```

```
    VARCHAR  sConnStr[1024];
    EXEC SQL END DECLARE SECTION;
```

- Log on GOLDILOCKS

```
strcpy((char *)sUid.arr, aUserID);
    sUid.len = (short)strlen((char *)sUid.arr);
    strcpy((char *)sPwd.arr, sPassword);
    sPwd.len = (short)strlen((char *)sPwd.arr);
    strcpy((char *)sConnStr.arr, aHostInfo);
    sConnStr.len = (short)strlen((char *)sConnStr.arr);
```

- DB connection

```
EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] Connection Failure!");

    return FAILURE;
}

int Disconnect()
{
    EXEC SQL DISCONNECT;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] [ERROR] Disconnect Failure!");
```

```
    return FAILURE;
}

int PrintRecord()
{
    EXEC SQL BEGIN DECLARE SECTION;
    rsEmpRecord sResultRecord;
    EXEC SQL END DECLARE SECTION;
    int   sRecordCount = 0;

    EXEC SQL
        DECLARE RECORD_CUR1 CURSOR FOR
        SELECT   empno, ename, dept
        FROM     SEMP
        ORDER BY empno;

    EXEC SQL OPEN RECORD_CUR1;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    printf(" EMPNO    ENAME               DEPT\n");
    printf("====== ==================== ==========\n");
    while( 1 )
    {
        EXEC SQL FETCH RECORD_CUR1 INTO :sResultRecord;
        if(sqlca.sqlcode == SQL_NO_DATA)
        {
            break;
        }
        else if(sqlca.sqlcode != 0)
        {
            goto fail_exit;
        }

        sRecordCount ++;

        printf("%6d %20s %10s\n",
               sResultRecord.mEmpNo,
```

```
                sResultRecord.mEName,
                sResultRecord.mDept);
    }

    printf("====== ==================== ==========\n");
    printf("Record Count = %d\n", sRecordCount);
    printf("====== ==================== ==========\n");

    EXEC SQL CLOSE RECORD_CUR1;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");

    return FAILURE;
}
```

Cursor Property

The previous chapter described only the most basic cursor. However, the cursor may have various properti es, and it can perform various functions depending on the properties. An embedded SQL of GOLDILOCKS provides the cursor properties provided in ISO/IEC-9075-2 SQL Foundation and ODBC.
For more information about the cursor definition syntax and the property, refer to **DECLARE cursor_name** .

Scrollable Cursor

SCROLL is the cursor property of ISO type, and it determines whether the cursor is scrollable. The scrollabl e cursor can receive the position option in the FETCH statement, and the row at the position according to this option is fetched. If scrolling is impossible, it can FETCH only the row in the result set sequentially.

When declaring the scroll cursor, SCROLL option is used. The following is an example of the declaration.

```
EXEC SQL DECLARE cur_scroll SCROLL CURSOR FOR SELECT c1, c2 FROM t1;
```

With NO SCROLL option, the cursor which is unable to scroll is declared. When the scroll option is not given, the default value is NO SCROLL.
For the scrollable cursor, it is possible to specify the position information when fetching. This position information is called as fetch orientation, and it is as follows.

Table 33-13 Fetch orientation

Fetch orientation	Description
NEXT	It fetches the next row of the current position.
PRIOR	It fetches the previous row of the current position.
FIRST	It fetches the first row in the result set.
LAST	It fetches the last row in the result set.
CURRENT	It fetches the row of the current position.
ABSOLUTE <position>	• It fetches the row corresponding to *position* in the result set. • If the position value is a negative number, it fetches the row of the previous position from AFTER THE LAST ROW.
RELATIVE <position>	It fetches the row which is away as much as *position* from the current position.

If fetch orientation is ABSOLUTE or RELATIVE, the value of <position> is additionally required. The value of <position> may be an integer literal or the host variable of the integer type. The following is an example of fetch orientation.

```
EXEC SQL FETCH NEXT;
EXEC SQL FETCH PRIOR;
EXEC SQL FETCH FIRST;
EXEC SQL FETCH LAST;
EXEC SQL FETCH CURRENT;
EXEC SQL FETCH ABSOLUTE 100;
EXEC SQL FETCH RELATIVE :position;
```

Sensitive Cursor

Sensitivity is the cursor property of ISO type, and it determines whether the changes can be displayed if the result set is changed during the cursor operation. Sensitivity has the following three options.

Table 33-14 Sensitivity

Option	Description
SENSITIVE	The information which is deleted or updated in another transaction can be viewed
INSENSITIVE	The information which is deleted or updated after the cursor is opened can not be viewed.
ASENSITIVE	SENSITIVE/ INSENSITIVE is selected according to information of query.

If sensitive option is not specified, the default value is INSENSITIVE, and the cursor with the sensitive optio

n is declared as follows.

```
EXEC SQL DECLARE <cur_name> SENSITIVE CURSOR FOR SELECT c1, c2 FROM t1;
EXEC SQL DECLARE <cur_name> INSENSITIVE CURSOR FOR SELECT c1, c2 FROM t1;
EXEC SQL DECLARE <cur_name> ASENSITIVE CURSOR FOR SELECT c1, c2 FROM t1;
```

Holdable Cursor

Holdability is the cursor property of ISO type, and it determines whether the cursor is still held even after t he transaction which opened the current cursor is committed. Holdability has the following two options.

Table 33-15 Holdability

Option	Description
WITH HOLD	• The cursor is held even after the end of the transaction. • It can not be used with FOR UPDATE clause. • It can not be used with INSERT INTO ... RETURNING statement. • It can not be used with UPDATE ... RETURNING statement. • It can not be used with DELETE FROM ... RETURNING statement. • It can not be used in the query including the global temporary table whose table commit action is ON COMMIT DELETE ROWS.
WITHOUT HOLD	The cursor is closed when COMMIT/ ROLLBACK the transaction.
Rollback and cursor	• It closes a cursor which is included in a transaction when rolling back the transac tion. • It closes a cursor which is created after the savepoint when rolling back the trans action to the savepoint.

If holdable option is not specified, the default value is determined according to <cursor updatability>.

- If FOR READ ONLY or <cursor updatability> is not specified, it is WITH HOLD.
- If it is used with FOR UPDATE statement, it is WITHOUT HOLD.

The cursor using holdable option is declared as follows.

```
EXEC SQL DECLARE <cur_name> CURSOR WITH HOLD FOR SELECT c1, c2 FROM t1;
EXEC SQL DECLARE <cur_name> CURSOR WITHOUT HOLD FOR SELECT c1, c2 FROM t1 FOR UPDATE;
```

Static Cursor

Static cursor is the cursor property of ODBC type, and it is as same as INSENSITIVE SCROLL cursor of ISO t ype.

```
EXEC SQL DECLARE cur_static STATIC CURSOR FOR SELECT c1, c2 FROM t1;
```

For more information about how to fetch the static cursor, refer to **Scrollable Cursor**.

Keyset Driven Cursor

Keyset driven cursor is the cursor property of ODBC type, it is as same as SENSITIVE SCROLL cursor of ISO type.

```
EXEC SQL DECLARE cur_static KEYSET CURSOR FOR SELECT c1, c2 FROM t1;
```

Keyset driven cursor also has the scroll feature, so for more information about how to fetch, refer to **Scrol lable Cursor**.

Positioned DML

Positioned DML refers that delete or update statement can be executed for the finally fetched row using CURRENT OF <cursor_name> clause. The cursor should be open and point to the row by performing the F etch at least once to execute the positioned DML.

The following is an example of executing the positioned DML.

```
EXEC SQL DECLARE emp_cursor CURSOR FOR
     SELECT ename, sal FROM emp WHERE job = 'SALES'
     FOR UPDATE;
...
EXEC SQL OPEN emp_cursor;
EXEC SQL WHENEVER NOT FOUND GOTO ...
while( 1 )
{
    EXEC SQL FETCH emp_cursor INTO :emp_name, :salary;
    ...
    EXEC SQL UPDATE emp SET sal = :new_salary
         WHERE CURRENT OF emp_cursor;
}
```

Options

This chapter describes the applicable options when an embedded SQL source code is precompiled.

Precompiled Header File

When developing the embedded SQL program, it is efficient that the information referenced in common by multiple source codes is stored in the separated header file and the header file is included in the sourc

e code. C language supports this feature by using #include statement. However, when the header file is i ncluded by using #include of C language the target file is not precompliled but it is interpreted by the co mpiler of C language, so the declare section within the file is not converted by the precompliler.

EXEC SQL INCLUDE statement is used to convert and insert this header file into the source code by the pr ecompiler when creating the header file. The syntax is as follows.

```
EXEC SQL INCLUDE <filename>;
```

The statement above precompiles the given file and inserts it into the source code.

Specifying Header File Path

Generally, it preferentially searches for the directory in which the current source code is stored when sear ching for the header file. However, header files are often separately stored in many cases or they exist in other paths for many reasons.

In this case, the directory searching for the header file is given separately as an option, and the syntax is a s follows.

```
EXEC SQL OPTION( INCLUDE = <directory path> );
```

Several options can be listed and the directories are searched in the order given by the options when sear ching for the header file via EXEC SQL INCLUDE.

This option can be given via command-line option of precompiler. For more information, refer t o **--include-path, -I** of **Precompiler Options**.

Host Array

So far, only the scalar variables having only a single value as the host variable were described. This chapte r describes how to use the array with the host variable.
When using the host array, the program source code becomes simple, and the performance is improved, but It should be used carefully because there are restrictions.

Declaring Host Array

Declaration of the host array is as same as the declaration of the scalar variable. A variable is required onl y to be declared as an array itself. The following is an example of declaring the host array with its size 10.

```
EXEC SQL BEGIN DECLARE SECTION;
    int    empno[10];
    char   ename[10][20];
    double salary[10];
EXEC SQL END DECLARE SECTION;
```

The host array declaration has the following limitations.

- More than two-dimensional array is not allowed.
- Exceptionally, the string family (char, varchar) and binary family (binary, varbinary) can be used as the two-dimensional array because their array consists of an array subscript representing the array size an d data size.
- An array of pointer is not allowed.

Using Host Array

Accessing Host Array

How to use the host array in the SQL statement is as same as to use the scalar host variable.
The following is a simple example of the host array.

```
EXEC SQL BEGIN DECLARE SECTION;
int    emp_number[20];
char   emp_name[20][10];
int    dept_number[20];
EXEC SQL END DECLARE SECTION;
```

- Setting the values of emp_number, emp_name, dept_number

```
...
```

- Inserting the values of emp_number, emp_name, dept_number

```
EXEC SQL INSERT INTO emp (empno, ename, deptno)
    VALUES (:emp_number, :emp_name, :dept_number);
```

The example above has the same feature as the following code.

```
EXEC SQL BEGIN DECLARE SECTION;
int    emp_number[20];
char   emp_name[20][10];
int    dept_number[20];
```

```
EXEC SQL END DECLARE SECTION;
```

- Setting the values of emp_number, emp_name, dept_number

```
...
```

- Inserting the values of emp_number, emp_name, dept_number

```
for( i = 0; i < 20; i ++ )
{
    EXEC SQL INSERT INTO emp (empno, ename, deptno)
        VALUES (:emp_number[i], :emp_name[i], :dept_number[i]);
}
```

When using several host variable arrays, the operation is performed for the array with the smallest array size among the host arrays. In the example above, 20 rows are inserted because the size of every host variable is 20. However, 10 rows are inserted as a result in the example below, if the array size of dept_number is specified as 10.

```
EXEC SQL BEGIN DECLARE SECTION;
int    emp_number[20];
char   emp_name[20][10];
int    dept_number[10];
EXEC SQL END DECLARE SECTION;
```

- Setting the values of emp_number, emp_name, dept_number

```
...
```

- Inserting the values of emp_number, emp_name, dept_number

```
EXEC SQL INSERT INTO emp (empno, ename, deptno)
    VALUES (:emp_number, :emp_name, :dept_number);
```

Using Host Indicator Array

If the host variable is an array, the indicator variable to be combined to it should also be an array. Also, the indicator variable and the host variable should have the same array size. The following is an example of adding the indicator variable to the example above.

```
EXEC SQL BEGIN DECLARE SECTION;
int    emp_number[20];
int    emp_number_ind[20];
```

```
char   emp_name[20][10];
int    emp_name_ind[20];
int    dept_number[20];
int      dept_number_ind[20];
EXEC SQL END DECLARE SECTION;
```

- Setting the values of emp_number, emp_name, dept_number
- Setting each indicator value of emp_number_ind, emp_name_ind, dept_number_ind

```
...
```

- Inserting the values of emp_number, emp_name, dept_number

```
EXEC SQL INSERT INTO emp (empno, ename, deptno)
    VALUES (:emp_number :emp_number_ind,
            :emp_name :emp_name_ind,
            :dept_number :dept_number_ind);
```

Restrictions

- When using several host variables, it is not allowed to use a mixture of scalar variables and array varia bles.
- The host array can not be used in WHERE clause of the select statement.
- The host array can not be used in CURRENT OF clause of the update, delete statements.

Array in INTO Clause

In GOLDILOCKS, rows are fetched by using SELECT INTO statement or cursor. An embedded SQL uses INT O clause in common for both two methods. Multiple rows are fetched by using the host array in INTO cla use.

Array in SELECT INTO

If the number of rows to be fetched is explicitly known, it is implemented by using the host array in **Select Into Statement**. Only the case of when row does not exist or a single row is fetched is described in **Select I nto Statement**. However, multiple rows can be fetched by using the host array in it.

The usage is as same as using scalar variables, but the Select Into statement with the array can be created just by declaring the host variable with the array.

```
EXEC SQL BEGIN DECLARE SECTION;
char   emp_name[50][20];
int    emp_number[50];
```

```
float  salary[50];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT ENAME, EMPNO, SAL
    INTO :emp_name, :emp_number, :salary
    FROM EMP
    WHERE SAL > 1000;
```

In the example above, SELECT INTO statement is used to fetch 50 rows by declaring the host variable wit h an array. This statement fetches only the first 50 rows which meet the query criteria because it is an ind ependent execution unit.

Even if more than 50 rows meet the query criteria, it can not fetch the rear rows starting from t he 51th row by using the SELECT INTO statement. Cursor should be used when continuously fet ching rows.

Array When Using Cursor

If the number of rows in the result set is unknown for the current query, the cursor should be used. For m ore information about how to use the cursor, refer to **Cursor**. After declaring the cursor, multiple rows ca n be fetched at once by using the host variable of INTO clause as an array in FETCH statement.

```
EXEC SQL BEGIN DECLARE SECTION;
    int   emp_number[50];
    char  emp_name[50][20];
    char  dept_name[50][20];
EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE EMP_CURSOR CURSOR FOR
        SELECT    empno, ename, dept
        FROM      EMP
        WHERE     empno < :sNo
        ORDER BY  empno;

    sNo = 100;
    EXEC SQL OPEN EMP_CURSOR;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
```

```
    while( 1 )
    {
        EXEC SQL FETCH EMP_CURSOR INTO :emp_number, :emp_name, :dept_name;
        if(sqlca.sqlcode == SQL_NO_DATA)
        {
            break;
        }
        else if(sqlca.sqlcode != 0)
        {
            goto fail_exit;
        }
        ...
    }
```

50 rows can be fetched at once by using the host array in FETCH statement.

sqlca.sqlerrd[2]

The rows are fetched as many as the declared array size when using an array, but sometimes it can not fe tch as many as the declared array size because there is not a row any more. For example, if the array size i s declared as 50 and the number of rows in the result set are 30, then only 30 rows can be fetched. To so lve this problem, an embedded SQL of GOLDILOCKS provides information about the number of rows pro cessed by the current statement in sqlca.sqlerrd[2].
The number of rows processed in INSERT, UPDATE, DELETE, SELECT INTO, FETCH statements are returne d in sqlca.sqlerrd[2].

```
EXEC SQL BEGIN DECLARE SECTION;
    int   emp_number[50];
    char  emp_name[50][20];
    char  dept_name[50][20];
EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE EMP_CURSOR CURSOR FOR
        SELECT    empno, ename, dept
        FROM      EMP
        WHERE     empno < :sNo
        ORDER BY  empno;

    sNo = 100;
    EXEC SQL OPEN EMP_CURSOR;
    if(sqlca.sqlcode != 0)
    {
```

```
        goto fail_exit;
    }

    while( 1 )
    {
        EXEC SQL FETCH EMP_CURSOR INTO :emp_number, :emp_name, :dept_name;
        if(sqlca.sqlcode == SQL_NO_DATA)
        {
            break;
        }
        else if(sqlca.sqlcode != 0)
        {
            goto fail_exit;
        }

        for( i = 0; i < sqlca.sqlerrd[2]; i ++ )
        {
            printf( "%d %s %s\n", emp_number[i], emp_name[i], dept_name[i] );
        }
        ...
    }
```

For more information about sqlca, refer to **Handling Run-time Errors**.

Array in Insert Statement

If the host variable is declared as an array in INSERT statement, the array insert is implemented.

```
EXEC SQL BEGIN DECLARE SECTION;
int    emp_number[20];
char   emp_name[20][10];
int    dept_number[20];
EXEC SQL END DECLARE SECTION;
```

- Setting the values of emp_number, emp_name, dept_number

```
...
```

- Inserting the values of emp_number, emp_name, dept_number

```
EXEC SQL INSERT INTO emp (empno, ename, deptno)
    VALUES (:emp_number, :emp_name, :dept_number);
```

The example above has the same feature as the following code.

```
EXEC SQL BEGIN DECLARE SECTION;
int    emp_number[20];
char   emp_name[20][10];
int    dept_number[20];
EXEC SQL END DECLARE SECTION;
```

- Setting the values of emp_number, emp_name, dept_number

```
...
```

- Inserting the values of emp_number, emp_name, dept_number

```
for( i = 0; i < 20; i ++ )
{
    EXEC SQL INSERT INTO emp (empno, ename, deptno)
        VALUES (:emp_number[i], :emp_name[i], :dept_number[i]);
}
```

- When using the array insert, all host variables should be array variables or all of them should be scalar variables.
- The number of inserted rows can be checked by using sqlca.sqlerrd[2].

Atomic Insert

Atomic insert is a special form of the array insert, and it has the following two characteristics.

- If any of the rows to be inserted is failed, the entire insert operation fails.
- The performance is outstanding because the command is transferred only once from the application t o GOLDILOCKS.

ATOMIC keyword is used for the atomic insert as follows.

```
EXEC SQL BEGIN DECLARE SECTION;
int    emp_number[20];
char   emp_name[20][10];
int    dept_number[20];
```

```
EXEC SQL END DECLARE SECTION;
```

- Setting the values of emp_number, emp_name, dept_number

```
...
```

- Inserting the values of emp_number, emp_name, dept_number

```
EXEC SQL ATOMIC INSERT INTO emp (empno, ename, deptno)
    VALUES (:emp_number, :emp_name, :dept_number);
```

Array in Update Statement

The following is an example of using the array in update statement.

```
EXEC SQL BEGIN DECLARE SECTION;
char  job_title [10][20];
float commission[10];
EXEC SQL END DECLARE SECTION;

...

EXEC SQL UPDATE emp SET comm = :commission
    WHERE job = :job_title;
```

The example above has the same feature as the following code.

```
EXEC SQL BEGIN DECLARE SECTION;
char  job_title [10][20];
float commission[10];
EXEC SQL END DECLARE SECTION;

...
for( i = 0; i < 10; i ++ )
{
    EXEC SQL UPDATE emp SET comm = :commission[i]
        WHERE job = :job_title[i];
}
```

- When using the array update, all host variables should be array variables or all of them shou ld be scalar variables.

- The number of updated rows can be checked by using sqlca.sqlerrd[2].
- Array can not be used in CURRENT OF clause of UPDATE statement.

Array in Delete Statement

Array can be used in DELETE statement as follows.

```
EXEC SQL BEGIN DECLARE SECTION;
char job_title[10][20];
EXEC SQL BEGIN DECLARE SECTION;
...
EXEC SQL DELETE FROM emp
    WHERE job = :job_title;
```

The example above has the same feature as the following code.

```
EXEC SQL BEGIN DECLARE SECTION;
char job_title[10][20];
EXEC SQL BEGIN DECLARE SECTION;
...
for( i = 0; i < 10; i ++ )
{
    EXEC SQL DELETE FROM emp
        WHERE job = :job_title[i];
}
```

- When using the array delete, all host variables should be array variables or all of them shoul d be scalar variables.
- The number of deleted rows can be checked by using sqlca.sqlerrd[2].
- Array can not be used in CURRENT OF clause of DELETE statement.

Using FOR Clause

FOR clause is used to specify the array size when executing the SQL statement. FOR clause can be used in the following statements.

- SELECT INTO
- FETCH
- INSERT
- UPDATE

- DELETE

For clause can be used as follows.

```
EXEC SQL FOR :host_variable <sql_stmt>
EXEC SQL FOR <integer_constant> <sql_stmt>
```

The following is an example of using the for clause.

```
EXEC SQL BEGIN DECLARE SECTION;
int    emp_number[20];
char   emp_name[20][10];
int    dept_number[20];
int      record_cnt;
EXEC SQL END DECLARE SECTION;
```

- Setting the values of emp_number, emp_name, dept_number

```
...
```

- Inserting the values of emp_number, emp_name, dept_number

```
record_cnt = 10;
EXEC SQL FOR :record_cnt INSERT INTO emp (empno, ename, deptno)
    VALUES (:emp_number, :emp_name, :dept_number);
```

In the example above, the actual number of inserted rows are 10 because the size of the host array is give n as 20 but it is specified to perform as much as record_cnt by using the FOR clause.

FOR clause can not be used with CURRENT OF clause in UPDATE/ DELETE statements.

Structure Array

Using the general scalar variables as an array has the advantage of processing multiple rows at once, but i t has a limit that a variable can represent only a single column.
For more information about how to process multiple columns with a single host variable, refer to **Host Str ucture**. Multiple rows with multiple columns can be processed at once by declaring a host variable as a str ucture and using the structure as the array.

A structure array can be used in the following cases.

- Output host variable array: SELECT INTO, FETCH INTO statements
- Input host variable array: VALUES item of INSERT statement

Restrictions

A structure array can not be used in the following cases.

- It can not be used in WHERE clause or FROM clause.
- It can not be used in SET clause of UPDATE statement.

Declaring Structure Array

A structure is declared in an way of a common C language structure declaration. A structural variable can be directly declared, or it can be declared as a host variable after declaring type via typedef.

```
EXEC SQL BEGIN DECLARE SECTION;
typedef struct rsRecord
{
    int          mEmpNo;
    varchar      mEName[20 + 1];
    char         mJob[20 + 1];
    long         mSalary;
} rsRecord;
rsRecord     sRecord[10];

struct {
    int          mEmpNo;
    varchar      mEName[20 + 1];
    char         mJob[20 + 1];
    long         mSalary;
} sResultRecord[10];
EXEC SQL END DECLARE SECTION;
```

When declaring the structure for using the host variable, the nested structure can not be used. If another structure is inside the structure declaration as follows, it can not be used as a host variable.

```
EXEC SQL BEGIN DECLARE SECTION;
typedef struct rsRecord
{
    struct person {
        int          mEmpNo;
        varchar      mEName[20 + 1];
    } person;
```

```
    char         mJob[20 + 1];
    long         mSalary;
} rsRecord;
EXEC SQL END DECLARE SECTION;
```

Indicators of Structure Array

As described in **Structure Indicator**, if a host variable is the structure, the Indicator variable should also be the structure. For the same reason, if the host variable is the structure array, the indicator variable should also be the corresponding structure array.

The declaration of the Indicator structure array should comply with the followings.

- It should have the number of members as many as the host variable structure to be combined by the i ndicator.
- All members of the Indicator structure should have a integer data type.
- The size of the indicator structure array should be as same as that of the host variable structure array. If size of the Indicator structure array is smaller, the number of array executions should be limited by using FOR clause when executing SQL statement.

Mixed Use of Structure and Scalar Variable

When the host structure is transferred to GOLDILOCKS, its structure members are sequentially listed. The following is an example of mixed use of the host structure and the scalar variable.

```
EXEC SQL BEGIN DECLARE SECTION;
    typedef struct rsEmp
    {
        int          mEmpNo;
        varchar      mEName[20 + 1];
    } rsEmp;
    rsEmp     sEmp[5];
    char      sJob[5][20 + 1];
    long      sSalary[5];
EXEC SQL END DECLARE SECTION;

EXEC SQL
    DECLARE EMP_CUR CURSOR FOR
    SELECT empno, ename, job, sal
    FROM   EMP
    ORDER BY EMPNO;
EXEC SQL OPEN EMP_CUR;
```

```
EXEC SQL
    FETCH EMP_CUR
    INTO  :sEmp, :sJob, :sSalary;
```

The structure rsEmp and the scalar variables sJob, sSalary are used together in FETCH statement. This stat ement is internally interpreted as four variables sEmp.mEmpNo, sEmp.mEName, sJob and sSalary. The ho st structure and the host scalar variable can be used together in this way.

The following is an example of mixed use of the host structure array, structure array indicator, structure a nd scalar variable.

```
/*
 * fetch_struct_array.gc
 *  : structure array fetch
 *  : structure indicators
 *  : mix structure, scalar variable
 *  : sqlca.sqlerrd[2]
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

EXEC SQL INCLUDE SQLCA;

#define  SUCCESS  0
#define  FAILURE  -1

#define  PRINT_SQL_ERROR(aMsg)                                  \
    {                                                           \
        printf("\n");                                           \
        printf(aMsg);                                           \
        printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n",   \
               sqlca.sqlcode,                                   \
               SQLSTATE,                                        \
               sqlca.sqlerrm.sqlerrmc );                        \
    }
EXEC SQL BEGIN DECLARE SECTION;
typedef struct rsEmp
{
    int          mEmpNo;
    varchar      mEName[20 + 1];
```

```
} rsEmp;
EXEC SQL END DECLARE SECTION;

int Connect(char *aHostInfo, char *aUserID, char *sPassword);

int main(int argc, char **argv)
{
    EXEC SQL BEGIN DECLARE SECTION;
    rsEmp     sEmp[5];
    char      sJob[5][20 + 1];
    long      sSalary[5];
    EXEC SQL END DECLARE SECTION;
    int  sRecordCount = 0;
    int  i;

    printf("Connect GOLDILOCKS ...\n");
    if(Connect("DSN=GOLDILOCKS", "test", "test") != SUCCESS)
    {
        goto fail_exit;
    }
```

- Retrieve employee

```
EXEC SQL
        DECLARE EMP_CUR CURSOR FOR
        SELECT empno, ename, job, sal
        FROM   EMP
        ORDER BY EMPNO;

    EXEC SQL OPEN EMP_CUR;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    printf(" EMPNO    ENAME                JOB       SALARY\n");
    printf("====== ==================== ========== ========\n");

    while( 1 )
    {
```

```
        EXEC SQL
            FETCH EMP_CUR
            INTO  :sEmp, :sJob, :sSalary;
        if(sqlca.sqlcode == SQL_NO_DATA)
        {
            break;
        }
        else if(sqlca.sqlcode != 0)
        {
            PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
            goto fail_exit;
        }

        sRecordCount += sqlca.sqlerrd[2];

        for( i = 0; i < sqlca.sqlerrd[2]; i ++ )
        {
            printf("%6d %20s %10s %8ld\n",
                   sEmp[i].mEmpNo, sEmp[i].mEName.arr, sJob[i], sSalary[i]);
        }
    }
    printf("====== ==================== ========== ========\n");
    printf("Record Count = %d\n", sRecordCount);
    printf("====== ==================== ========== ========\n");

    EXEC SQL CLOSE EMP_CUR;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK RELEASE;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    printf("\n\nSUCCESS\n");
    printf("############################\n");
```

```
    return 0;

  fail_exit:

    printf("\n\nFAILURE\n");
    printf("############################\n\n");

    EXEC SQL ROLLBACK WORK RELEASE;

    return 0;
}

int Connect(char *aHostInfo, char *aUserID, char *sPassword)
{
    EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR  sUid[80];
    VARCHAR  sPwd[20];
    VARCHAR  sConnStr[1024];
    EXEC SQL END DECLARE SECTION;
```

- Log on GOLDILOCKS

```
strcpy((char *)sUid.arr, aUserID);
sUid.len = (short)strlen((char *)sUid.arr);
strcpy((char *)sPwd.arr, sPassword);
sPwd.len = (short)strlen((char *)sPwd.arr);
strcpy((char *)sConnStr.arr, aHostInfo);
sConnStr.len = (short)strlen((char *)sConnStr.arr);
```

- DB connection

```
EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:
```

```
    PRINT_SQL_ERROR("[ERROR] Connection Failure!");

    return FAILURE;
}
```

Handling Run-time Errors

Overview

In case when the application can not get the expected result during execution, a preparation is required f or this case when developing an embedded SQL application. This chapter explains how to detect the resu lts returned after the SQL execution.

Detecting Run-time Error

SQLCA

All types of errors which occur in an embedded SQL are reported in the area called as SQL Communicatio n Area (SQLCA). Therefore, the application can identify the success of the current operation and the error type by checking the information of SQLCA.

SQLCA is a data structure which stores errors, warnings, SQL statement execution states. SQLCA will only have the results of the last SQL execution which the data structure is combined, but it does not have infor mation about the history of executed SQL statement. If an embedded SQL statement is executed, the exis ting contents of SQLCA disappears. Therefore, it is necessary to immediately check SQLCA and to perfor m the exception handling after executing the SQL statement.

Using SQLCA

The following syntax is used to use SQLCA.

```
EXEC SQL INCLUDE SQLCA;
```

The syntax above is replaced with the following statement in the precompile process.

```
#include "sqlca.h"
```

This statement should be used before using the first embedded SQL statement, and it is typically recomm ended to be positioned at the top of the source code.

The embedded SQL application of GOLDILOCKS generally has a global sqlca. In a single thread program, s qlca can be used without any declaration. However, in multi thread program, sqlca is required to be decla

red separately because the simultaneous accesses to sqlca cause the concurrency problems. For more info rmation, refer to **Multithread Application**.

SQLCA Structure

The following describes the structure of sqlca.

```
struct sqlca
{
    char    sqlcaid[8];    ❶ It is initialized to the string SQLCA.
    int     sqlabc;        ❷ It is the size of sqlca structure.

    /*
     * ❸ It is the error code which occurred in the most recent statement execution.
     * ❹ If it is 0, it is successful. If it is a positive number, it is a warning. If it is
a negative number, it is an error.
     */
    int     sqlcode;

    /*
     * ❺ It stores the error message for sqlcode
     * ❻ .sqlerrml is the length of sqlerrmc.
     * ❼ .sqlerrmc stores the error message in a string form.
     */
    struct
    {
        unsigned short sqlerrml;
        char           sqlerrmc[SQLERRMC_LEN];
    } sqlerrm;

    char    sqlerrp[8];   /* unused                          */
    int     sqlerrd[6];
    /* 0: empty                                             */
    /* 1: empty                                             */
    /* 2: ❽ It is the number of rows processed after INSERT, UPDATE, DELETE.        */
    /* 3: empty                                             */
    /* 4: empty                                             */
    /* 5: empty                                             */
    char    sqlwarn[8];
    /* 0: ❾ If any warning occurs, it is 'W'.
     * 1: ❿ If the result string is truncated in SELECT, FETCH, it is 'W'.
     * 2: unused
```

```
     * 3: unused
     * 4: unused
     * 5: unused
     * 6: unused
     * 7: unused
     */
    char    sqlext[8];    /* unused                          */
    char    sqlstate[8];  /* SQLSTATE                        */
    unsigned short  *rowstatus;    /* fetched row status array*/
};
```

The next chapter describes the content of each component.

SQLCODE

SQLCODE is defined as follows.

```
#define SQLCODE          (sqlca.sqlcode)
```

SQLCODE executes an embedded SQL statement and then returns the result code. SQLCODE was proposed early in ISO/IEC-9075, but it is deprecated in SQL-92. However, it is provided for the backward compatibility because it is being used in many applications. The result code is as follows.

Table 33-16 Execution result of SQLCODE

SQLCODE	Result
0	Success
> 0	Warning
SQL_NO_DATA	No result
< 0	Error

sqlcode is checked as follows. sqlca.sqlcode or SQLCODE can be used.

```
EXEC SQL
        DECLARE EMP_CUR CURSOR FOR
        SELECT empno, ename, job, sal
        FROM   EMP;

    EXEC SQL OPEN EMP_CUR;
    if(SQLCODE != 0)
    {
        goto fail_exit;
    }
    while( 1 )
```

```
    {
        EXEC SQL
            FETCH EMP_CUR
            INTO  :sEmpNo, :sEName, :sJob, :sSalary;
        if(sqlca.sqlcode == SQL_NO_DATA)
        {
            break;
        }
        else if(sqlca.sqlcode != 0)
        {
            goto fail_exit;
        }

        ...
    }
```

SQLSTATE

SQLSTATE is defined as follows.

```
#define SQLSTATE         (sqlca.sqlstate)
```

SQLCODE was deprecated in ISO/IEC-9075, and SQLSTATE is proposed instead. SQLSTATE consists of five characters(number, English uppercase alphabet), and the first two digits are called as class, and the rear 3 digits are called as subclass. The result of SQLSTATE are as follows.

Table 33-17 Execution result of SQLSTATE

SQLSTATE	Result
00000	Success
01xxx	Warning
02000	No result
All other states	Error

The code is modified by checking SQLSTATE instead of SQLCODE in the example above as follows. sqlca.sqlstate or SQLSTATE can be used.

```
EXEC SQL
        DECLARE EMP_CUR CURSOR FOR
        SELECT empno, ename, job, sal
        FROM   EMP;

    EXEC SQL OPEN EMP_CUR;
```

```
    if(strcmp(SQLSTATE, "00000") != 0)
    {
        goto fail_exit;
    }
    while( 1 )
    {
        EXEC SQL
            FETCH EMP_CUR
            INTO  :sEmpNo, :sEName, :sJob, :sSalary;
        if(strcmp(sqlca.sqlstate, "02000") == 0)
        {
            break;
        }
        else if(strcmp(sqlca.sqlstate, "00000") != 0)
        {
            goto fail_exit;
        }

        ...
    }
```

Number of Processed Rows

When executing update, delete statements or insert, select info, fetch statement using array, it informs a user the number of the processed rows. The information is stored in sqlca.sqlerrd[2]. The application refe rs to the field value and finds out the number of the processed rows. The following is an example of usin g sqlca.sqlerrd[2].

```
EXEC SQL BEGIN DECLARE SECTION;
    typedef struct rsEmp
    {
        int          mEmpNo;
        varchar      mEName[20 + 1];
    } rsEmp;
    rsEmp     sEmp[5];
    char      sJob[5][20 + 1];
    long      sSalary[5];
EXEC SQL END DECLARE SECTION;
    EXEC SQL
        DECLARE EMP_CUR CURSOR FOR
        SELECT empno, ename, job, sal
        FROM   EMP;
```

```
EXEC SQL OPEN EMP_CUR;
if(SQLCODE != 0)
{
    goto fail_exit;
}
while( 1 )
{
    EXEC SQL
        FETCH EMP_CUR
        INTO  :sEmpNo, :sEName, :sJob, :sSalary;
    if(sqlca.sqlcode == SQL_NO_DATA)
    {
        break;
    }
    else if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    sRecordCount += sqlca.sqlerrd[2];
    ...
}
```

sqlca.sqlerrd[2] value in fetch statement can be processed as the accumulated number of rows. For more information, refer to --**cumulative** of gpec(Precompiler) option.

Status of Processed Rows

sqlca.row status indicates the row status which is currently processed. When the row is updated or delete d by using the scroll sensitive cursor or Where CURRENT OF, then the row status can be updated. When using SQL statement with array, It has row status as many as the array size.

Table 33-18 Referring to the row status

Number of rows	Row status
1	*sqlca.rowstatus
Array size n	sqlca.rowstats[0] sqlca.rowstats[1] sqlca.rowstats[2] ... sqlca.rowstats[n-1]

Table 33-19 Value of row status

Row status	Description
SQL_ROW_SUCCESS	The row status is normal.
SQL_ROW_DELETED	The row is deleted.
SQL_ROW_UPDATED	The row is updated.
SQL_ROW_NOROW	The row does not exist.
SQL_ROW_ADDED	The row is added.
SQL_ROW_ERROR	The row status is not normal.

The following is an example of referring to the row status.

```
EXEC SQL BEGIN DECLARE SECTION;
    typedef struct rsEmp
    {
        int          mEmpNo;
        varchar      mEName[20 + 1];
    } rsEmp;
    rsEmp     sEmp[5];
    char      sJob[5][20 + 1];
    long      sSalary[5];
EXEC SQL END DECLARE SECTION;
    EXEC SQL
        DECLARE EMP_CUR CURSOR FOR
        SELECT empno, ename, job, sal
        FROM   EMP;

    EXEC SQL OPEN EMP_CUR;
    if(SQLCODE != 0)
    {
        goto fail_exit;
    }
    while( 1 )
    {
        EXEC SQL
            FETCH EMP_CUR
            INTO  :sEmpNo, :sEName, :sJob, :sSalary;
        if(sqlca.sqlcode == SQL_NO_DATA)
        {
            break;
        }
        else if(sqlca.sqlcode != 0)
```

```
        {
            goto fail_exit;
        }

        sRecordCount += sqlca.sqlerrd[2];

        for( i = 0; i < sqlca.sqlerrd[2]; i ++ )
        {
            if( sqlca.rowstatus[i] == SQL_ROW_SUCCESS )
            {
                printf("%6d %20s %10s %8ld\n",
                       sEmp[i].mEmpNo, sEmp[i].mEName.arr, sJob[i], sSalary[i]);
            }
        }

        ...
    }
```

Error Message Text

When an error or warning occurs as a result of the embedded SQL statement, the message can be transf erred in a text form. The error message is stored in sqlca.sqlerrm, and sqlca.sqlerrm.sqlerrml is the length of the text. The actual message is stored in sqlca.sqlerrm.sqlerrmc. The error message text is useful to out put information to a user when anomalies occur in the application. The following is an example of using an error message text.

```
EXEC SQL INSERT INTO EMP VALUES ( :sEmp );
if( SQLCODE != 0 )
{
    printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n",
           SQLCODE,
           SQLSTATE,
           sqlca.sqlerrm.sqlerrmc );
}
```

Warning Flags

When a waring occurs after executing the embedded SQL, sqlca.sqlwarn is used as the flag which marks the warning. It consists of eight char arrays, and the mark of 'W' is used when the warning occurs.

Table 33-20 Warning flags

Warning flag	Description
sqlca.sqlwarn[0]	If any warning occurs, it is 'W'.
sqlca.sqlwarn[1]	If the result string is truncated in select into, fetch, then it is 'W'.
sqlca.sqlwarn[2]	reserved
sqlca.sqlwarn[3]	reserved
sqlca.sqlwarn[4]	reserved
sqlca.sqlwarn[5]	reserved
sqlca.sqlwarn[6]	reserved
sqlca.sqlwarn[7]	reserved

Handling Implicit Error

After executing the embedded SQL statement, a user should check SQLCA and take action into the execution result. However, when treating the same exception occurs after executing the embedded SQL statement, it can be automated by using WHENEVER indicator.

Using WHENEVER Statement

WHENEVER statement has the following syntax.

```
EXEC SQL WHENEVER conditions actions;
```

WHENEVER Condition

Conditions of WHENEVER statement are as follows.

```
<conditions> ::=
      SQLERROR
    ¦ SQLWARNING
    ¦ NOT FOUND
    ¦ SQLSTATE <sqlstate class value>[<sqlstate subclass value>]
    ;

<sqlstate_char> ::= [0-9A-Z];
<sqlstate class value> ::= <sqlstate_char><sqlstate_char>;
<sqlstate subclass value> ::= <sqlstate_char><sqlstate_char><sqlstate_char>;
```

Table 33-21 Conditions of WHENEVER statement

Conditions	Descriptions
SQLERROR	An error occurs while an embedded SQL is executed.
SQLWARNING	A warning occurs while an embedded SQL is executed.

Conditions	Descriptions
NOT FOUND	A result row does not exist.
SQLSTATE <sqlstate>	SQLSTATE <sqlstate> occurs while an embedded SQL is executed.

WHENEVER Action

When actions meet the conditions described above, it describes the action actually performed, and its syn tax is as follows.

```
<actions> ::=
      CONTINUE
    ¦ GOTO <label>
    ¦ STOP
    ¦ DO <c statements>
    ;
```

Table 33-22 Actions of WHENEVER statement

Actions	Description
CONTINUE	An action is not performed. It ignores the given conditions.
GOTO <label>	It branches the program flow with <label>.
STOP	It terminates the program execution.
DO <c statements>	It executes <c statements>.

Scope of WHENEVER Statement

WHENEVER statement describes actions for conditions, and only one action can be described for one con dition. When using WHENEVER statement, the same action is used for all embedded SQL statements unti l the definition of another condition comes after the current condition.

Maximum four WHENEVER statements can be applied for a specific point because WHENEVER statement s are separately managed by each condition. If a new WHENEVER statement is applied for the same condi tion, the existing action is canceled and a new action may be applied afterwards. The following is an exa mple.

```
EXEC SQL WHENEVER SQLERROR STOP;
EXEC SQL INSERT INTO emp VALUES ( :emp_number, :emp_name, :salary ); ❶
...
EXEC SQL WHENEVER SQLERROR CONTINUE;
EXEC SQL UPDATE emp SET sal = sal * 1.1 WHERE sal < :sal_bound; ❷
...
EXEC SQL WHENEVER SQLERROR GOTO exit_label;
EXEC SQL WHENEVER NOT FOUND DO break;
```

```
EXEC SQL DECLARE EMP_CURSOR FOR
         SELECT empno, ename, sal
         FROM   emp;

EXEC SQL OPEN EMP_CURSOR; ❸

EXEC SQL WHENEVER SQLERROR GOTO close_label;
while( 1 )
{
    EXEC SQL FETCH EMP_CURSOR
              INTO   :emp_number, :emp_name, :salary; ❹

    printf( "emp number : %d, emp name : %s, salary : %lf\n",
            emp_number, emp_name, salary );
}
close_label:
EXEC SQL WHENEVER SQLERROR DO sql_error();
EXEC SQL CLOSE EMP_CURSOR; ❺
...
exit_label:
...
```

The application is stopped when an error occurs during executing ❶ because the stop operation is specified for SQLERROR in line 1. SQLERROR action is changed to proceed without performing any action because CONTINUE is indicated for SQLERROR in line 4 . Therefore, it proceeds to the next even when an error occurs during executing ❷. It branches to exit_label when an error occurs during executing ❸ because SQLERROR action is set to branch to exit_label in line 7. It is specified to perform break for NOT FOUND in line 8, and SQLERROR action is reassigned as close_label in line 15. Therefore, two conditions are applied to ❹. When an error occurs during FETCH, it branches to close_label, and if the FETCH result does not exist, then the break statement is executed. sql_error() function is called when an error occurs during executing ❺ because SQLERROR action is specified to execute sql_error() in line 26.

Notice for WHENEVER statement

WHENEVER statement should be used carefully after understanding the operation principle. The followings should be considered when using the WHENEVER statement.

- The position of WHENEVER statement on the source code: WHENEVER statement is an indicator which informs to automatically insert the exception handling code to precompiler, and it is not an logically executable code. WHENEVER should be at the beginning of the embedded SQL statements to be applied on the source code. CONTINUE action should be used to initialize not to affect the following embedded SQL.
- Using the break, continue keywords: If DO break, DO continue are used in an action clause, the loop

scope should be checked and used. The following is an example.

```
EXEC SQL WHENEVER SQLERROR GOTO fail_exit;
EXEC SQL WHENEVER NOT FOUND DO break; ❶
EXEC SQL DECLARE EMP_CURSOR FOR
         SELECT empno, ename, sal
         FROM   emp;

EXEC SQL OPEN EMP_CURSOR;
while( 1 )
{
    EXEC SQL FETCH EMP_CURSOR
             INTO   :emp_number, :emp_name, :salary;  ❷
    printf( "emp number : %d, emp name : %s, salary : %lf\n",
            emp_number, emp_name, salary );
}
EXEC SQL CLOSE EMP_CURSOR;

EXEC SQL
    SELECT MAX(sal)
    INTO   :max_salary
    FROM   emp;          ❸
...
```

break; may be performed for NOT FOUND in subsequent SQL statements because the action for NOT FOUND is defined as DO break; in ❶. When NOT FOUND occurs in ❷ during FETCH, break is performed and it exits the while loop. However, break can not be performed when NOT FOUND occurs in ❸ because it is not a loop. In this case, the compile error occurs when building the source program and developing the application

- Avoiding infinite loop

If it is misused when branching to the action, it causes the infinite loop. The following is an example.

```
EXEC SQL WHENEVER SQLERROR GOTO sql_error;
...
EXEC SQL INSERT INTO emp VALUES ( :emp_number, :emp_name, :salary );
...
sql_error:
    EXEC SQL ROLLBACK WORK RELEASE;
```

When an error occurs during executing the SQL statement, it branches to sql_error label. When an error

occurs during executing the embedded SQL statement, it branches to sql_error. However, when the error s repeatedly occurs during executing another embedded SQL statement in an error processing, the applic ation infinitely loops. In this case, the following is recommended to safely initialize the error handling.

```
EXEC SQL WHENEVER SQLERROR GOTO sql_error;
...
EXEC SQL INSERT INTO emp VALUES ( :emp_number, :emp_name, :salary );
...
sql_error:
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    EXEC SQL ROLLBACK WORK RELEASE;
```

- Scope of branch label

If it is branched in an action, its label should be located in an accessible position. The following is an exa mple.

```
func1()
{

    EXEC SQL WHENEVER SQLERROR GOTO labelA;
    EXEC SQL DELETE FROM emp WHERE deptno = :dept_number;
    ...
labelA:
...
}
func2()
{

    EXEC SQL INSERT INTO emp (job) VALUES (:job_title);
    ...
}
```

In func1(), the branching the labelA is specified for SQLERROR. However, the complile error occurs becau se labelA exists in func1() but it does not exist in func2(). In this case, the same labelA should be created i n func2() or the action should be initilaized as follows.

```
func1()
{

    EXEC SQL WHENEVER SQLERROR GOTO labelA;
    EXEC SQL DELETE FROM emp WHERE deptno = :dept_number;
```

```
    ...
labelA:
...
}
func2()
{
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    EXEC SQL INSERT INTO emp (job) VALUES (:job_title);
    ...
}
```

33.3 Advanced Topic

Embedded Dynamic SQL

Overview

Most of the embedded SQL applications perform the specific operations on GOLDILOCKS. They are to insert, update, delete, retrieve rows and they are specified by using the database language called as SQL. When SQL is directly specified on the embedded SQL source code for this purpose, the precompiler interpretes the SQL, and converts it to an API call available to GOLDILOCKS on the state of knowing all the given SQL and input/output host variables.

However, some applications do not know the SQL in advance when developing the applications. For example, if static SQL is used in an application such as GUI tool when the user wants to query by selecting the operation type, the table name, condition, the user should specify all SQL statement of possible combination in advance. However, it is impossible or it is very inefficient even when it is possible. In this case, if the user generates and executes the SQL statement with an user selectable option, it might be very flexible and efficient. The dynamic SQL is the SQL which is not defined on the source code in advance and is changed at run-time. GOLDILOCKS supports the embedded dynamic SQL feature.

The advantage of embedded dynamic SQL application provides more flexible usage compared to the static SQL, but the disadvantage is that the development of the source code is difficult and it has poor performance compared to the static SQL when performing the same query.

It is recommended to compare the static SQL and the dynamic SQL, then select one of them considering these features carefully when developing applications.

This chapter describes how to develop the embedded dynamic SQL program.

Dynamic SQL Types

The dynamic SQLs are classified according the usage as follows.

Table 33-23 Dynamic SQL type

Method	Description	Support
Method 1	It is a non-query and a host variable does not exist.	O
Method 2	It is a non-query and a host variable may know the number and type.	O
Method 3	It is a query and a host variable may know the number and type.	O
Method 4	It is a query, and the existence of host variable, the number and type are unknown.	X

Currently, GOLDILOCKS supports until the method 3.

Method 1

It is the simplest form of the dynamic SQL, and it is a non-query and can be used when a host variable do es not exist. Typically, it is used for the DDL or for the DML in which the host variable does not exist.

- EXECUTE IMMEDIATE

The SQL statement can be immediately executed because the method1 is a non-query and it does not ha ve the host variable at the same time. The syntax of the immediate execution of SQL is as follows.

```
EXEC SQL EXECUTE IMMEDIATE { :host_variable ¦ <string_literal> };
<string_literal> ::=
      ' <sql_statement> '
    ¦ " <sql_statement> "
    ¦ <sql_statement>
    ;
```

The method 1 is used as follows.

```
sprintf(sSqlStmt, "CREATE TABLE EMP_RND (\n"
            "EMPNO NUMBER(4) CONSTRAINT PK_EMP_RND PRIMARY KEY,\n"
            "ENAME VARCHAR2(10),\n"
            "JOB VARCHAR2(9),\n"
            "SAL NUMBER(7,2),\n"
            "DEPTNO NUMBER(2) )\n" );
    EXEC SQL EXECUTE IMMEDIATE :sSqlStmt;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    sprintf(sSqlStmt, "INSERT INTO EMP_RND\n"
            "SELECT *\n"
            "FROM   EMP\n"
            "WHERE  JOB = 'RND'\n" );
    EXEC SQL EXECUTE IMMEDIATE :sSqlStmt;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
```

Method 2

The method 2 is used for the non-query with an existence of input host variable. The number of input ho st Variables and the data type should be known, and it is performed through preparation step and the ex ecution step.

- Prepare

In the prepare step, the SQL statement is analyzed, and the name is given to this statement. The syntax of prepare is as follows.

```
EXEC SQL PREPARE <statement_name> FROM { :host_variable ¦ <string_literal> };
<string_literal> ::=
      ' <sql_statement> '
    ¦ " <sql_statement> "
    ¦ <sql_statement>
    ;
```

<statement_name> is an identifier notifying to the precompiler. It does not require a separate type or any declaration of a variable because it is not the host variable.

- Execute

The analyzed statement is executed in the execute step. The syntax of execute statement is as follows.

```
EXEC SQL EXECUTE <statement_name> [ USING <host_variable_list> ];
<host_variable_list> ::= <host_variable_entry> [ , <host_variable_list> ];
<host_variable_entry> ::= :host_variable [ [ INDICATOR ] :host_indicator ];
```

If an input host variable exist, USING clause is used. If the host variable does not exist, USING clause is om itted. It is bound to the host variable of the prepared SQL statement according to the sequence of host va riables in USING clause.

When the same SQL statements are repeatedly executed in method 2, execute may be repeatedly perfor med after performing prepare only once. The prepared statement is valid in the current source code until another SQL statement is prepared with the same statement name or it is disconnected.

Method 3

The method 3 is an extended form of the method 2 to support the query. The statements of declare, ope n, fetch, close are added for the cursor because generally a statement is analyzed in the query prepare pr ocess and the cursor should be handled for the query.

- Prepare

The prepare step analyzes the SQL statement as like the method 2, and gives the name to the statement. The syntax of prepare is as follows.

```
EXEC SQL PREPARE <statement_name> FROM { :host_variable ¦ <string_literal> };
<string_literal> ::=
      ' <sql_statement> '
    ¦ " <sql_statement> "
    ¦ <sql_statement>
    ;
```

<statement_name> is an identifier notifying to the precompiler. It does not require a separate type or any declaration of a variable because it is not the host variable.

- Declare cursor

Declare statement declares the cursor for the prepared statement. When declaring the cursor, the cursor property which is as same as the standing cursor can be used. The syntax is as follows.

```
EXEC SQL <dynamic declare cursor>;

<dynamic declare cursor> ::=
    DECLARE <cursor_name> <cursor properties> { FOR ¦ IS } <statement_name>
    ;
<cursor properties> ::=
      [ <cursor sensitivity> ] [ <cursor scrollability>] ] CURSOR [ <cursor holdability> ]
    ¦ [ <odbc cursor type] CURSOR [ <cursor holdability> ]
    ;
<cursor sensitivity> ::=
      INSENSITIVE
    ¦ SENSITIVE
    ¦ ASENSITIVE
    ;
<cursor scrollability> ::=
      NO SCROLL
    ¦ SCROLL
    ;
<cursor holdability> ::=
      WITH HOLD
    ¦ WITHOUT HOLD
    ;
<odbc cursor type> ::=
      STATIC
```

```
    ¦ KEYSET
    ;
```

The meaning and used name of <cursor properties> is as same as the standing cursor.

<statement_name> is the name specified in PREPARE statement. <cursor name> and <statement name> a re the identifiers notifying to the precompiler. It does not require a separate type or any declaration of a v ariable because it is not the host variable.

Only one cursor can be declared per one statement. If another cursor is declared when there is a cursor al ready declared in the statement, then the existing cursor is closed.

- Open cursor

It opens the cursor. It is generally same with the **Open Cursor** of the standing cursor, but there is a signific ant and typical difference in opening the dynamic cursor. The host variable is determined depending on t he declared SQL statement because the dynamic cursor updates and uses the SQL statement freely at the run-time. Therefore, when opening the dynamic cursor, it transfers the host variable by using the USING clause at the time of opening. The cursor should be closed at the time of opening the standing cursor, bu t the dynamic cursor is opened after closing the cursor.

Open syntax of dynamic cursor is as follows.

```
EXEC SQL <dynamic cursor open>;

<dynamic cursor open> ::=
    OPEN <cursor_name> [ USING <host_variable_list> ]
    ;

<host_variable_list> ::= <host_variable_entry> [ , <host_variable_list> ];
<host_variable_entry> ::= :host_variable [ [ INDICATOR ] :host_indicator ];
```

- Fetch cursor

It fetches from the cursor. Fetching dynamic cursor is as same as **fetching standing cursor**.

- Close cursor

It closes the cursor. Closing dynamic cursor is as same as **closing standing cursor**.

Example Program

The following is a sample program of using the dynamic method 1, 2, 3.

```
/*
 * dyn2.gc
 *  : dynamic method 1
 *  : dynamic method 2
 *  : dynamic method 3
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
EXEC SQL INCLUDE SQLCA;

#define  SUCCESS  0
#define  FAILURE  -1
#define  PRINT_SQL_ERROR(aMsg)                                          \
    {                                                                   \
        printf("\n");                                                   \
        printf(aMsg);                                                   \
        printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n",   \
               sqlca.sqlcode,                                           \
               SQLSTATE,                                                \
               sqlca.sqlerrm.sqlerrmc );                                \
    }

EXEC SQL BEGIN DECLARE SECTION;
typedef struct Record
{
    int          mEmpNo;
    varchar      mEName[20 + 1];
    char         mJob[20];
    char         mSalary[10];
} Record;
EXEC SQL END DECLARE SECTION;

int Connect(char *aHostInfo, char *aUserID, char *sPassword);
int CreateEmpTempTable();
int DropEmpTempTable();
int UpdateSalary(char *aJob, int aBound, double aRatio);

int main(int argc, char **argv)
{
```

```
    EXEC SQL BEGIN DECLARE SECTION;
    EXEC SQL END DECLARE SECTION;
    printf("Connect GOLDILOCKS ...\n");
    if(Connect("DSN=GOLDILOCKS", "test", "test") != SUCCESS)
    {
        goto fail_exit;
    }

    if(CreateEmpTempTable() != SUCCESS)
    {
        goto fail_exit;
    }
```

- Print RND employee increate 20% salary where salary < 2000

```
printf("print RND employee increate 20%% salary where salary < 2000\n");
UpdateSalary( "RND", 2000, 1.2 );
printf("\n\n");
```

- Print SUPPORT employee increate 10% salary where salary < 3000

```
printf("print SUPPORT employee increate 10%% salary where salary < 3000\n");
UpdateSalary( "SUPPORT", 3000, 1.1 );
printf("\n\n");

if(DropEmpTempTable() != SUCCESS)
{
     goto fail_exit;
}

printf("Disconnect GOLDILOCKS ...\n");
EXEC SQL COMMIT WORK RELEASE;
if(sqlca.sqlcode != 0)
{
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    goto fail_exit;
}

printf("SUCCESS\n");
printf("############################\n");

return 0;
```

```
  fail_exit:

    printf("\n\nFAILURE\n");
    printf("############################\n\n");
    EXEC SQL ROLLBACK WORK RELEASE;

    return 0;
}

int Connect(char *aHostInfo, char *aUserID, char *sPassword)
{
    EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR  sUid[80];
    VARCHAR  sPwd[20];
    VARCHAR  sConnStr[1024];
    EXEC SQL END DECLARE SECTION;
```

- Log on GOLDILOCKS

```
strcpy((char *)sUid.arr, aUserID);
    sUid.len = (short)strlen((char *)sUid.arr);
    strcpy((char *)sPwd.arr, sPassword);
    sPwd.len = (short)strlen((char *)sPwd.arr);
    strcpy((char *)sConnStr.arr, aHostInfo);
    sConnStr.len = (short)strlen((char *)sConnStr.arr);
```

- DB connection

```
EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] Connection Failure!");

    return FAILURE;
```

```
}
```

- Create table

```
int CreateEmpTempTable()
{
    EXEC SQL BEGIN DECLARE SECTION;
    char   sSqlStmt[8192];
    EXEC SQL END DECLARE SECTION;
    sprintf(sSqlStmt, "DROP TABLE IF EXISTS EMP_RND" );
    EXEC SQL EXECUTE IMMEDIATE :sSqlStmt;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    sprintf(sSqlStmt, "CREATE TABLE EMP_RND (\n"
            "EMPNO NUMBER(4) CONSTRAINT PK_EMP_RND PRIMARY KEY,\n"
            "ENAME VARCHAR2(10),\n"
            "JOB VARCHAR2(9),\n"
            "SAL NUMBER(7,2),\n"
            "DEPTNO NUMBER(2) )\n" );
    EXEC SQL EXECUTE IMMEDIATE :sSqlStmt;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    sprintf(sSqlStmt, "INSERT INTO EMP_RND\n"
            "SELECT *\n"
            "FROM   EMP\n"
            "WHERE  JOB = 'RND'\n" );
    EXEC SQL EXECUTE IMMEDIATE :sSqlStmt;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    sprintf(sSqlStmt, "DROP TABLE IF EXISTS EMP_SUPPORT" );
    EXEC SQL EXECUTE IMMEDIATE :sSqlStmt;
    if(sqlca.sqlcode != 0)
```

```
    {
        goto fail_exit;
    }

    sprintf(sSqlStmt, "CREATE TABLE EMP_SUPPORT (\n"
            "EMPNO NUMBER(4) CONSTRAINT PK_EMP_SUPPORT PRIMARY KEY,\n"
            "ENAME VARCHAR2(10),\n"
            "JOB VARCHAR2(9),\n"
            "SAL NUMBER(7,2),\n"
            "DEPTNO NUMBER(2) )\n" );
    EXEC SQL EXECUTE IMMEDIATE :sSqlStmt;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    sprintf(sSqlStmt, "INSERT INTO EMP_SUPPORT\n"
            "SELECT *\n"
            "FROM   EMP\n"
            "WHERE  JOB = 'SUPPORT'\n" );
    EXEC SQL EXECUTE IMMEDIATE :sSqlStmt;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}
```

- Drop table

```
int DropEmpTempTable()
{
    EXEC SQL DROP TABLE EMP_RND;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL DROP TABLE EMP_SUPPORT;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}

int UpdateSalary(char *aJob, int aBound, double aRatio)
{
    EXEC SQL BEGIN DECLARE SECTION;
    Record       sRecord;
    char         sSelectSql[128];
    char         sUpdateSql[128];
    int          sBound = aBound;
    double       sRatio = aRatio;
    EXEC SQL END DECLARE SECTION;
```

```
    int  sRecordCount = 0;
    int  i;
    int  sIsOpenCur = 0;

    sprintf( sSelectSql, "SELECT EMPNO, ENAME, JOB, SAL FROM EMP_%s WHERE sal < :v1 FOR
UPDATE", aJob );
    sprintf( sUpdateSql, "UPDATE EMP_%s SET sal = sal * :v1 WHERE CURRENT OF DYN_CUR", aJob);

    EXEC SQL PREPARE SELECT_STMT FROM :sSelectSql;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL DECLARE DYN_CUR KEYSET CURSOR FOR SELECT_STMT;
    EXEC SQL OPEN DYN_CUR USING :sBound;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    sIsOpenCur = 1;

    EXEC SQL PREPARE UPDATE_STMT FROM :sUpdateSql;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    while( 1 )
    {
        EXEC SQL
            FETCH NEXT DYN_CUR
            INTO  :sRecord;
        if(sqlca.sqlcode == SQL_NO_DATA)
        {
            break;
        }
        else if(sqlca.sqlcode != 0)
        {
            goto fail_exit;
```

```
        }

        EXEC SQL EXECUTE UPDATE_STMT USING :sRatio;
        if(sqlca.sqlcode != 0)
        {
            goto fail_exit;
        }
    }

    sIsOpenCur = 0;
    EXEC SQL CLOSE DYN_CUR;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    sprintf( sSelectSql, "SELECT EMPNO, ENAME, JOB, SAL FROM EMP_%s ORDER BY SAL DESC", aJob
);
    EXEC SQL PREPARE SELECT_STMT FROM :sSelectSql;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL OPEN DYN_CUR;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    sIsOpenCur = 1;

    printf("%s salary list\n", aJob);
    sRecordCount = 0;
    printf(" EMPNO    ENAME                JOB       SALARY\n");
    printf("====== ==================== ========== ==========\n");
    while( 1 )
    {
        EXEC SQL
            FETCH DYN_CUR
            INTO  :sRecord;
        if(sqlca.sqlcode == SQL_NO_DATA)
```

```
        {
            break;
        }
        else if(sqlca.sqlcode != 0)
        {
            PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
            goto fail_exit;
        }

        for(i = 0; i < sqlca.sqlerrd[2]; i ++)
        {
            sRecordCount ++;
            printf("%6d %20s %10s %10s\n",
                   sRecord.mEmpNo,
                   sRecord.mEName.arr,
                   sRecord.mJob,
                   sRecord.mSalary);
        }
    }

    printf("====== ==================== ========== ==========\n");
    printf("Record Count = %d\n", sRecordCount);
    printf("====== ==================== ========== ==========\n");

    sIsOpenCur = 0;
    EXEC SQL CLOSE DYN_CUR;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    if(sIsOpenCur == 1)
    {
        EXEC SQL CLOSE DYN_CUR;
    }
    return FAILURE;
```

```
}
```

Multithread Application

The multithreaded application is an application including multiple execution units in a single process. The multithreaded application can process multiple tasks such as running multiple applications at the same time in parallel, and it has the advantage of being able to share the same address scope because it is a single process.

Sharing the same address scope means sharing the global variables and static variables. When accessing these variables, the concurrency control for each thread should be considered. Therefore, the application should be developed very carefully.

GOLDILOCKS supports the run-time context. The run-time context has a slight difference between Direct Attach (D/A) mode and Client/Server (C/S) mode. The next chapter describes the run-time context and the guideline for developing the multithreaded application.

Run-time Context

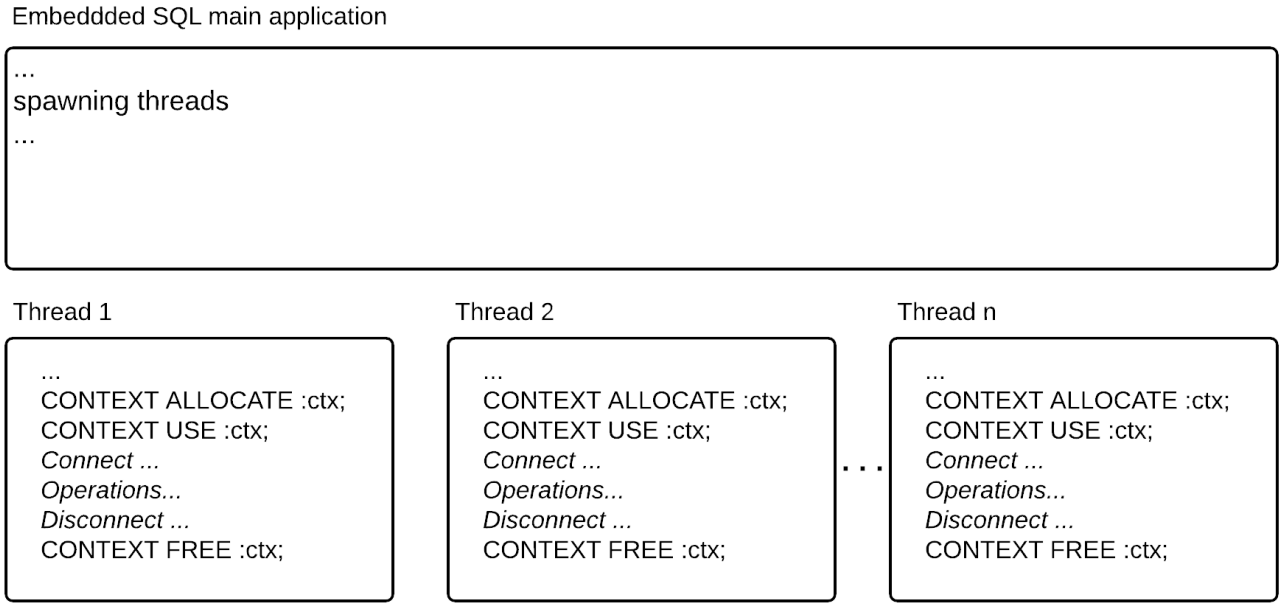
The run-time context in an embedded SQL of GOLDILOCKS is used for the purpose of managing the connection from the application to GOLDILOCKS. The relationship between the run-time context and connection is 1:1, and the run-time context is regarded as the connection itself.

GOLDILOCKS structurally allows only a single connection in a thread in D/A mode. The multithreaded application is required for multiple connections. However, it is the connection with the server via network in C/S mode, so a single thread may have multiple connections.

Direct Attach (D/A) Mode

A single thread has only a single connection in D/A mode. Therefore, n threads are required when developing an application with n connections.

Figure 2 In case when each thread has its own connection in D/A mode



Client/Server (C/S) Mode

An extra restriction for the connection does not exist when operating in C/S mode. A single thread may h ave multiple connections or multiple threads may share a single connection. N threads can share m conne ctions.

Figure 3 In case when multiple threads share a single connection

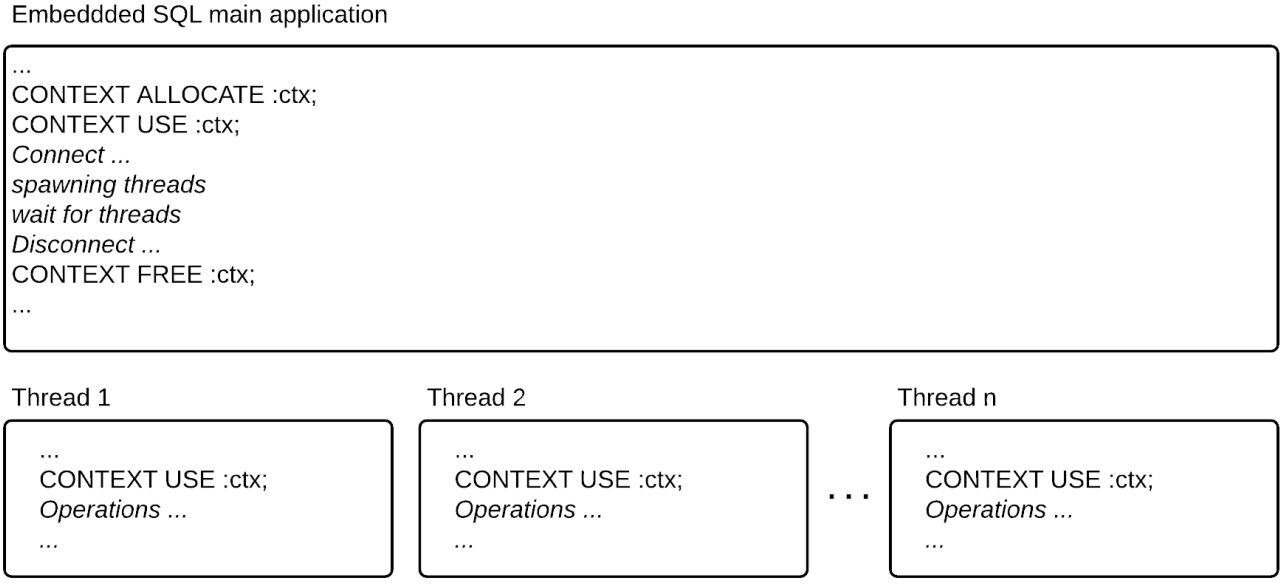
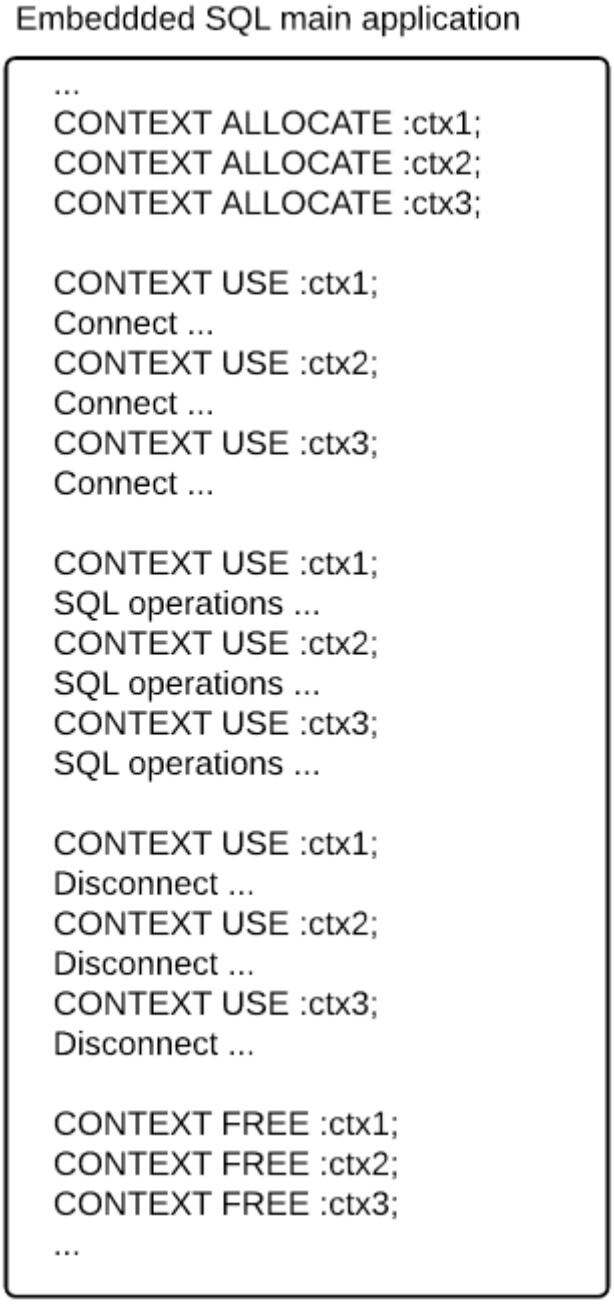


Figure 4 In case when a single thread has multiple connections



Guidelines

The followings should be considered to develop the multithreaded application.

- The SQLCA variable should be declared thread-safely. It is recommended to declare as a stack variabl e in each thread, and to refer to the example program of the next chapter.
- The static variables or the global variables are shared because the multithread has the same address s cope in a single process. The concurrency control should be considered when using these variables.
- A single run-time context should not be allowed to be used at the same time in multiple threads. The concurrency control of run-time context should also be considered.

Example Program

The following is a sample program of a multithreaded application.

```
/*
 * thread1.gc
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>

EXEC SQL INCLUDE SQLCA;

#define  SUCCESS  0
#define  FAILURE  -1

#define  PRINT_SQL_ERROR(aMsg)                                    \
    {                                                             \
        printf("\n");                                             \
        printf(aMsg);                                             \
        printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n",  \
               sqlca.sqlcode,                                     \
               SQLSTATE,                                          \
               sqlca.sqlerrm.sqlerrmc );                          \
    }

int Connect(sql_context aCtx, char *aHostInfo, char *aUserID, char *sPassword);
int CreateEmpTempTable();
int DropEmpTempTable();
void *clientThread(void *args);

typedef struct thread_param
{
    int    mNo;
    char  *mJobName;
} thread_param;

#define  THREAD_COUNT    2
char gJobName[THREAD_COUNT][20]= {
```

```
    "RND",
    "SUPPORT"
};

int main(int argc, char **argv)
{
    EXEC SQL BEGIN DECLARE SECTION;
    int          sEmpNo;
    varchar      sEName[20 + 1];
    char         sJob[20];
    long         sSalary;
    EXEC SQL END DECLARE SECTION;
    int          sRecordCount = 0;
    pthread_t    thread_id[THREAD_COUNT];
    thread_param param[THREAD_COUNT];
    int          i;

    printf("Connect GOLDILOCKS ...\n");
    if(Connect(NULL, "DSN=GOLDILOCKS", "test", "test") != SUCCESS)
    {
        goto fail_exit;
    }

    if(CreateEmpTempTable() != SUCCESS)
    {
        goto fail_exit;
    }
```

- Create client thread

```
for( i = 0; i < THREAD_COUNT; i ++ )
    {
        param[i].mNo = i;
        param[i].mJobName = gJobName[i];
        if( pthread_create(&thread_id[i],
                           NULL,
                           clientThread,
                           &param[i]) != 0 )
        {
            printf( "Can't create thread %d!\n", i );
        }
```

```
        else
        {
            printf( "Create thread %d!\n", i );
        }
    }

    for( i = 0; i < THREAD_COUNT; i ++ )
    {
        if( pthread_join(thread_id[i],
                         NULL) != 0 )
        {
            printf( "Error when waiting for thread %d to terminate!\n", i );
        }
        else
        {
            printf( "Stopped thread %d!\n", i );
        }
    }
```

- Retrieve employee

```
EXEC SQL
        DECLARE EMP_CUR CURSOR FOR
        SELECT empno, ename, job, sal
        FROM   EMP_TEMP
        ORDER BY empno;

    EXEC SQL OPEN EMP_CUR;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    printf(" EMPNO    ENAME                JOB       SALARY\n");
    printf("====== ==================== ========== ========\n");
    while( 1 )
    {
        EXEC SQL
            FETCH EMP_CUR
            INTO  :sEmpNo, :sEName, :sJob, :sSalary;
```

```
        if(sqlca.sqlcode == SQL_NO_DATA)
        {
            break;
        }
        else if(sqlca.sqlcode != 0)
        {
            PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
            goto fail_exit;
        }

        sRecordCount ++;

        printf("%6d %20s %10s %8ld\n",
               sEmpNo, sEName.arr, sJob, sSalary);
    }

    printf("====== ==================== ========== ========\n");
    printf("Record Count = %d\n", sRecordCount);
    printf("====== ==================== ========== ========\n");

    EXEC SQL CLOSE EMP_CUR;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    if(DropEmpTempTable() != SUCCESS)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK RELEASE;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    printf("SUCCESS\n");
    printf("############################\n");
```

```
    return 0;

  fail_exit:

    printf("FAILURE\n");
    printf("###########################\n\n");
    EXEC SQL ROLLBACK WORK RELEASE;

    return 0;
}

int Connect(sql_context aCtx, char *aHostInfo, char *aUserID, char *sPassword)
{
    EXEC SQL BEGIN DECLARE SECTION;
    VARCHAR  sUid[80];
    VARCHAR  sPwd[20];
    VARCHAR  sConnStr[1024];
    EXEC SQL END DECLARE SECTION;
    struct sqlca sqlca;
```

- Log on GOLDILOCKS

```
strcpy((char *)sUid.arr, aUserID);
sUid.len = (short)strlen((char *)sUid.arr);
strcpy((char *)sPwd.arr, sPassword);
sPwd.len = (short)strlen((char *)sPwd.arr);
strcpy((char *)sConnStr.arr, aHostInfo);
sConnStr.len = (short)strlen((char *)sConnStr.arr);
```

- DB connection

```
if( aCtx != NULL )
    {
        EXEC SQL CONTEXT USE :aCtx;
        EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
    }
    else
    {
        EXEC SQL CONTEXT USE DEFAULT;
        EXEC SQL CONNECT :sUid IDENTIFIED BY :sPwd USING :sConnStr;
    }
```

```
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] Connection Failure!");

    return FAILURE;
}
```

- DB disconnection

```
int Disconnect(sql_context aCtx)
{
    struct sqlca sqlca;
    if( aCtx != NULL )
    {
        EXEC SQL CONTEXT USE :aCtx;
        EXEC SQL DISCONNECT;
    }
    else
    {
        EXEC SQL CONTEXT USE DEFAULT;
        EXEC SQL DISCONNECT;
    }
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] Connection Failure!");

    return FAILURE;
```

```
}
```

- Create table

```
int CreateEmpTempTable()
{
    EXEC SQL DROP TABLE IF EXISTS EMP_TEMP;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL
        CREATE TABLE EMP_TEMP (
            EMPNO NUMBER(4) CONSTRAINT PK_EMP_TEMP PRIMARY KEY,
            ENAME VARCHAR2(10),
            JOB VARCHAR2(9),
            SAL NUMBER(7,2),
            DEPTNO NUMBER(2) );
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}
```

- Drop table

```
int DropEmpTempTable()
{
    EXEC SQL DROP TABLE EMP_TEMP;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }

    return SUCCESS;

  fail_exit:

    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL ROLLBACK WORK;

    return FAILURE;
}

void *clientThread(void *args)
{
    EXEC SQL BEGIN DECLARE SECTION;
    SQL_CONTEXT   my_context;
    char          job_name[20 + 1];
    EXEC SQL END DECLARE SECTION;
    int           state = 0;
    thread_param *param = (thread_param *)args;

    EXEC SQL CONTEXT ALLOCATE :my_context;
    state = 1;

    EXEC SQL CONTEXT USE :my_context;
    if(Connect(my_context, "DSN=GOLDILOCKS", "test", "test") != SUCCESS)
    {
        goto fail_exit;
    }
```

```
  state = 2;

  strcpy( job_name, param->mJobName );

  EXEC SQL
      INSERT INTO EMP_TEMP
      SELECT *
      FROM   EMP
      WHERE  JOB = :job_name;
  if(sqlca.sqlcode != 0)
  {
      goto fail_exit;
  }

  EXEC SQL COMMIT WORK;
  if(sqlca.sqlcode != 0)
  {
      goto fail_exit;
  }
  state = 1;
  if(Disconnect(my_context) != SUCCESS)
  {
      goto fail_exit;
  }
  state = 0;
  EXEC SQL CONTEXT FREE :my_context;
  pthread_exit(0);
  return NULL;

fail_exit:

  PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
  switch(state)
  {
      case 2:
          (void)Disconnect(my_context);
      case 1:
          EXEC SQL CONTEXT FREE :my_context;
          break;
      default:
          break;
```

```
    }

    pthread_exit(0);
    return NULL;
}
```

Multi-process Application

A multi-process application is an application program which creates and executes multiple processes by u sing a single process.

GOLDILOCKS embedded SQL does not support a multi-process application. Use a fork function to write a multi-process application. If the application does not execute a new program but continues to executing the existing program after calling the fork function, then be aware of the followings.

- It is difficult to predict which error would occur because the context connected from the parent proc ess is copied to the child process after forking.
- Each OS has a different semaphore management policy. Some OS does not allow both the parent pr ocess and child process clear the semaphore created from the parent process.

If writing a multi-process application, then execute forking without any connection. Also, it is required to check the semaphore management policy of the corresponding OS.

C++ Application

Extension of Output Filename

Precompiler (gpec) of GOLDILOCKS generates C code by precompiling the embedded SQL source code. T he extension of the output file is .c by default. The general C++ source code has various file extensions de pending on the compiler types.

-o option which is the output file specification option of gpec is used to specify the file extension. For exa mple, testfile.gc is converted to testfile.cpp as follows.

```
gpec $(GPEC_OPT) testfile.gc -o testfile.cpp
```

SQLCA_STORAGE_CLASS

When the same symbol is declared as a global variable, the conflict on the symbol is occurred in C++ appl ication. The embedded SQL of GOLDILOCKS has the sqlca variable by default, and the conflict of these va

riables causes a problem when linking multiple C++ files.

SQLCA_STORAGE_CLASS macro should be defined for all other files except for one file to avoid this probl em. For example, if a single application is generated by linking 3 files, SQLCA_STORAGE_CLASS macro is defined for two of three files as follows.

```
#define SQLCA_STORAGE_CLASS extern
EXEC SQL INCLUDE SQLCA;
```

SQLCA_STORAGE_CLASS macro definition should be positioned ahead of the following statement.

```
EXEC SQL INCLUDE SQLCA;
```

XA

Definition of xa_open string

xa_open string includes information for connecting to Resource Manager (RM). For more information, re fer to **SQLDriverConnect attributes**.

The following is an example of xa_open string.

```
DSN=GOLDILOCKS;UID=test;PWD=test;CONN_NAME=XA_CONN
```

Using XA in Precompiler

A user can choose one of the followings when using XA in the precompiler.

- Using the default connection
- Using the named connection

Using Default Connection

It uses the connection connected only with the access information in xa_open string.
The following is an example of xa_open string for the default connection.

```
DSN=GOLDILOCKS;UID=test;PWD=test
```

It is described as follows when using the embedded SQL with the default connection.

```
EXEC SQL
        UPDATE Deposit
        Set InterestRates = :value :value_ind
        WHERE AccountNumber = :account_number;
```

Any connection should not exist in the default context before executing xa open to use XA with the default context.
Do not create multiple XA connections with xa_open string which does not have CONN_NAME.
When creating multiple connections without CONN_NAME, then the first created connection is matched with the default context.

Using Named Connection

It uses the connection name together with the connection information in xa_open string.
The following is an example of xa_open string for the named connection.

```
DSN=GOLDILOCKS;UID=test;PWD=test;CONN_NAME=XA_CONN
```

The connection name should be specified as follows in an embedded SQL which uses the named connect ion.

```
EXEC SQL AT XA_CONN
        UPDATE Deposit
        Set InterestRates = :value :value_ind
        WHERE AccountNumber = :account_number;
```

Example Program

- GOLDILOCKS Sample - XA

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

- Include GOLDILOCKS ODBC header

```
#include <goldilocks.h>
EXEC SQL INCLUDE SQLCA;
#define  SUCCESS  0
#define  FAILURE  -1
```

```
#define  PRINT_SQL_ERROR(aMsg)                                          \
    {                                                                   \
        printf("\n");                                                   \
        printf(aMsg);                                                   \
        printf("\nSQLCODE : %d\nSQLSTATE : %s\nERROR MSG : %s\n",   \
               sqlca.sqlcode,                                           \
               SQLSTATE,                                                \
               sqlca.sqlerrm.sqlerrmc );                                \
    }
```

- User-specific definitions

```
#define  BUF_LEN 101
#define GOLDILOCKS_SQL_THROW( aLabel )              \
    goto aLabel;
#define GOLDILOCKS_SQL_TRY( aExpression )           \
    do                                          \
    {                                           \
        if( !(SQL_SUCCEEDED( aExpression ) ) )  \
        {                                       \
            goto GOLDILOCKS_FINISH_LABEL;           \
        }                                       \
    } while( 0 )
#define GOLDILOCKS_FINISH                           \
    goto GOLDILOCKS_FINISH_LABEL;                   \
    GOLDILOCKS_FINISH_LABEL:
```

- Print diagnostic record to console

```
void PrintDiagnosticRecord( SQLSMALLINT aHandleType, SQLHANDLE aHandle )
{
    SQLCHAR       sSQLState[6];
    SQLINTEGER    sNaiveError;
    SQLSMALLINT   sTextLength;
    SQLCHAR       sMessageText[SQL_MAX_MESSAGE_LENGTH];
    SQLSMALLINT   sRecNumber = 1;
    SQLRETURN     sReturn;
```

- SQLGetDiagRec returns the current values which includes an error, warning.

```
while( 1 )
    {
        sReturn = SQLGetDiagRec( aHandleType,
                                 aHandle,
                                 sRecNumber,
                                 sSQLState,
                                 &sNaiveError,
                                 sMessageText,
                                 100,
                                 &sTextLength );
        if( sReturn == SQL_NO_DATA )
        {
            break;
        }
        GOLDILOCKS_SQL_TRY( sReturn );
        printf("\n=============================================\n" );
        printf("SQL_DIAG_SQLSTATE     : %s\n", sSQLState );
        printf("SQL_DIAG_NATIVE       : %d\n", sNaiveError );
        printf("SQL_DIAG_MESSAGE_TEXT : %s\n", sMessageText );
        printf("=============================================\n" );
        sRecNumber++;
    }
    return;
    GOLDILOCKS_FINISH;
    printf("SQLGetDiagRec failure.\n" );
    return;
}
```

- Create table

```
int testCreateTable()
{
    EXEC SQL AT XA_CONN
        DROP TABLE IF EXISTS DEPOSIT;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    EXEC SQL AT XA_CONN
        CREATE TABLE DEPOSIT (
            NAME          VARCHAR(30),
```

```
            BALANCE       INTEGER,
            ACCOUNTNUMBER VARCHAR(100),
            ACCOUNTDAY    DATE,
            INTERESTRATES NUMBER(10, 5),
            PHONENUMBER   VARCHAR(30) );
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    EXEC SQL AT XA_CONN COMMIT;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    return SUCCESS;
  fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL AT XA_CONN ROLLBACK;
    return FAILURE;
}
```

- Drop table

```
int testDropTable()
{
    EXEC SQL AT XA_CONN
        DROP TABLE DEPOSIT;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    EXEC SQL AT XA_CONN COMMIT;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
    return SUCCESS;
  fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    EXEC SQL AT XA_CONN ROLLBACK;
    return FAILURE;
```

```
}
```

- Insert function

```
int testInsert( )
{
    EXEC SQL BEGIN DECLARE SECTION;
    char            sName[BUF_LEN];
    int             sNameInd                    = 0;
    int             sBalance                    = 0;
    int             sBalanceInd                 = 0;
    char            sAccountNumber[BUF_LEN];
    int             sAccountNumberInd           = 0;
    DATE            sAccountDay;
    int             sAccountDayInd              = 0;
    double          sInterestRates              = 0;
    int             sInterestRatesInd           = 0;
    char            sPhoneNumber[BUF_LEN];
    int             sPhoneNumberInd             = 0;
    EXEC SQL END DECLARE SECTION;
    sNameInd              = snprintf( (char*)sName, BUF_LEN, "sunje" );
    sBalance              = 30000000;
    sAccountNumberInd     = snprintf( (char*)sAccountNumber, BUF_LEN, "9999-99-9999" );
    sAccountDay.year      = 2009;
    sAccountDay.month     = 1;
    sAccountDay.day       = 1;
    sAccountDay.hour      = 0;
    sAccountDay.minute    = 0;
    sAccountDay.second    = 0;
    sAccountDay.fraction  = 0;
    sInterestRates        = (double)5.0;
    sPhoneNumberInd       = snprintf( (char*)sPhoneNumber, BUF_LEN, "010-9999-9999" );
    EXEC SQL AT XA_CONN
        INSERT INTO DEPOSIT
        VALUES ( :sName :sNameInd,
                 :sBalance :sBalanceInd,
                 :sAccountNumber :sAccountNumberInd,
                 :sAccountDay :sAccountDayInd,
                 :sInterestRates :sInterestRatesInd,
                 :sPhoneNumber :sPhoneNumberInd );
    if(sqlca.sqlcode != 0)
```

```
    {
        goto fail_exit;
    }
```

- The number of rows affected by INSERT statement

```
printf("\n%d row created.\n\n", sqlca.sqlerrd[2] );
    return SUCCESS;
  fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    return FAILURE;
}
```

- Update function

```
int testUpdate( )
{
    EXEC SQL BEGIN DECLARE SECTION;
    char     sCondition[BUF_LEN];
    int      sConditionInd           = 0;
    double   sValue                  = 0;
    int      sValueInd               = 0;
    EXEC SQL END DECLARE SECTION;
    sValue        = (SQLREAL)6.0;
    sConditionInd = snprintf( (char*)sCondition,
                              BUF_LEN,
                              "9999-99-9999" );
    EXEC SQL AT XA_CONN
        UPDATE Deposit
        Set InterestRates = :sValue :sValueInd
        WHERE AccountNumber = :sCondition :sConditionInd;
    if(sqlca.sqlcode != 0)
    {
        goto fail_exit;
    }
```

- The number of rows affected by UPDATE statement

```
printf("\n%d row updated.\n\n", sqlca.sqlerrd[2] );
    return SUCCESS;
  fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
```

```
    return FAILURE;
}
```

- Select function

```
int testSelect( )
{
    EXEC SQL BEGIN DECLARE SECTION;
    char            sName[BUF_LEN];
    int             sNameInd                    = 0;
    int             sBalance                    = 0;
    int             sBalanceInd                 = 0;
    char            sAccountNumber[BUF_LEN];
    int             sAccountNumberInd           = 0;
    DATE            sAccountDay;
    int             sAccountDayInd              = 0;
    double          sInterestRates              = 0;
    int             sInterestRatesInd           = 0;
    char            sPhoneNumber[BUF_LEN];
    int             sPhoneNumberInd             = 0;
    EXEC SQL END DECLARE SECTION;
    int             sCount                      = 0;
    int             sIsOpen                     = 0;
    EXEC SQL DECLARE CUR1 CURSOR FOR
        SELECT NAME, BALANCE, ACCOUNTNUMBER, ACCOUNTDAY, INTERESTRATES, PHONENUMBER
        FROM DEPOSIT;
    EXEC SQL AT XA_CONN
        OPEN CUR1;
    if( sqlca.sqlcode != 0 )
    {
        goto fail_exit;
    }
    sIsOpen = 1;
    printf( "=========================================\n" );
    while( 1 )
    {
        EXEC SQL AT XA_CONN
            FETCH CUR1 INTO
            :sName :sNameInd,
            :sBalance :sBalanceInd,
            :sAccountNumber :sAccountNumberInd,
```

```
    :sAccountDay :sAccountDayInd,
    :sInterestRates :sInterestRatesInd,
    :sPhoneNumber :sPhoneNumberInd;
if( sqlca.sqlcode == SQL_NO_DATA )
{
    break;
}
else if( sqlca.sqlcode != 0 )
{
    goto fail_exit;
}
printf( "NAME          : " );
if( sNameInd == -1 )
{
    printf( "(null)" );
}
else
{
    printf( "%s", sName );
}
printf( "\n" );
printf( "BALANCE       : " );
if( sBalanceInd == -1 )
{
    printf( "(null)" );
}
else
{
    printf( "%d", sBalance );
}
printf( "\n" );
printf( "ACCOUNTNUMBER : " );
if( sAccountNumberInd == -1 )
{
    printf( "(null)" );
}
else
{
    printf( "%s", sAccountNumber );
}
printf( "\n" );
```

```
        printf( "ACCOUNTDAY    : " );
        if( sAccountDayInd == -1 )
        {
            printf( "(null)" );
        }
        else
        {
            printf( "%4d-%02d-%02d", sAccountDay.year, sAccountDay.month, sAccountDay.day);
        }
        printf( "\n" );
        printf( "INTERESTRATES : " );
        if( sInterestRatesInd == -1 )
        {
            printf( "(null)" );
        }
        else
        {
            printf( "%lf", sInterestRates );
        }
        printf( "\n" );
        printf( "PHONENUMBER   : " );
        if( sPhoneNumberInd == -1 )
        {
            printf( "(null)" );
        }
        else
        {
            printf( "%s", sPhoneNumber );
        }
        printf( "\n" );
        printf( "----------------------------------------\n" );
        sCount ++;
    }
    printf( "========================================\n" );
    printf( "\n%d rows selected.\n\n", sCount );
    sIsOpen = 0;
    EXEC SQL AT XA_CONN
        CLOSE CUR1;
    if( sqlca.sqlcode != 0 )
    {
        goto fail_exit;
```

```
    }
    return SUCCESS;
  fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    if( sIsOpen == 1 )
    {
        EXEC SQL AT XA_CONN
            CLOSE CUR1;
    }
    return FAILURE;
}
```

- Delete function

```
int testDelete( )
{
    EXEC SQL BEGIN DECLARE SECTION;
    char     sCondition[BUF_LEN];
    int      sConditionInd           = 0;
    EXEC SQL END DECLARE SECTION;
    sConditionInd = snprintf( (char*)sCondition,
                              BUF_LEN,
                              "9999-99-9999" );
    EXEC SQL AT XA_CONN
        DELETE FROM DEPOSIT WHERE AccountNumber = :sCondition :sConditionInd;
    if( sqlca.sqlcode != 0 )
    {
        goto fail_exit;
    }
```

- The number of rows affected by DELETE statement

```
printf("\n%d row deleted.\n\n", sqlca.sqlerrd[2] );
    return SUCCESS;
  fail_exit:
    PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
    return FAILURE;
}
```

- Start function

```
int main( int aArgc, char** aArgv )
{
    SQLHENV       sEnv    = NULL;
    SQLINTEGER    sState  = 0;
    xa_switch_t * sXaSwitch;
    XID           sXid;
    sXaSwitch = SQLGetXaSwitch();
```

- If a user calls SQLAllocEnv() which is included in GOLDILOCKS ODBC

```
GOLDILOCKS_SQL_TRY( SQLAllocHandle( SQL_HANDLE_ENV,
                                    NULL,
                                    &sEnv ) );
    sState = 1;
```

- SQLSetEnvAttr sets attributes which controls aspects of environments.

```
GOLDILOCKS_SQL_TRY( SQLSetEnvAttr( sEnv,
                                   SQL_ATTR_ODBC_VERSION,
                                   (SQLPOINTER)SQL_OV_ODBC3,
                                   0 ) );
    if( (sXaSwitch->xa_open_entry)( "DSN=GOLDILOCKS;UID=test;PWD=test;CONN_NAME=XA_CONN", 0,
TMNOFLAGS ) != XA_OK )
    {
        GOLDILOCKS_SQL_THROW( GOLDILOCKS_FINISH_LABEL );
    }
    sState = 2;
    sXid.formatID = 0;
    sXid.gtrid_length = 2;
    sXid.bqual_length = 1;
    memcpy( sXid.data, "100", sXid.gtrid_length + sXid.bqual_length );
```

- Create table

```
if( testCreateTable() != SUCCESS )
    {
        GOLDILOCKS_SQL_THROW( GOLDILOCKS_FINISH_LABEL );
    }
    sState = 3;
    if( (sXaSwitch->xa_start_entry)( &sXid, 0, TMNOFLAGS ) != XA_OK )
    {
        GOLDILOCKS_SQL_THROW( GOLDILOCKS_FINISH_LABEL );
```

```
    }
    sState = 4;
```

- Insert row

```
if( testInsert() != SUCCESS )
    {
        GOLDILOCKS_SQL_THROW( GOLDILOCKS_FINISH_LABEL );
    }
```

- Update row

```
if( testUpdate() != SUCCESS )
    {
        GOLDILOCKS_SQL_THROW( GOLDILOCKS_FINISH_LABEL );
    }
```

- Select row

```
if( testSelect() != SUCCESS )
    {
        GOLDILOCKS_SQL_THROW( GOLDILOCKS_FINISH_LABEL );
    }
```

- Delete row

```
if( testDelete() != SUCCESS )
    {
        GOLDILOCKS_SQL_THROW( GOLDILOCKS_FINISH_LABEL );
    }
    sState = 3;
    if( (sXaSwitch->xa_end_entry)( &sXid, 0, TMSUCCESS ) != XA_OK )
    {
        GOLDILOCKS_SQL_THROW( GOLDILOCKS_FINISH_LABEL );
    }
    if( (sXaSwitch->xa_prepare_entry)( &sXid, 0, TMNOFLAGS ) != XA_OK )
    {
        GOLDILOCKS_SQL_THROW( GOLDILOCKS_FINISH_LABEL );
    }
    if( (sXaSwitch->xa_commit_entry)( &sXid, 0, TMNOFLAGS ) != XA_OK )
    {
        GOLDILOCKS_SQL_THROW( GOLDILOCKS_FINISH_LABEL );
```

```
    }
    sState = 2;
```

- Drop table

```
if( testDropTable() != SUCCESS )
    {
        GOLDILOCKS_SQL_THROW( GOLDILOCKS_FINISH_LABEL );
    }
    sState = 1;
    if( (sXaSwitch->xa_close_entry)( "", 0, TMNOFLAGS ) != XA_OK )
    {
        GOLDILOCKS_SQL_THROW( GOLDILOCKS_FINISH_LABEL );
    }
```

- SQLFreeHandleEnv releases resources related to the environment.

```
sState = 0;
    GOLDILOCKS_SQL_TRY( SQLFreeHandle( SQL_HANDLE_ENV,
                                   sEnv ) );
    sEnv = NULL;
    return EXIT_SUCCESS;
    GOLDILOCKS_FINISH;
    if( sEnv != NULL)
    {
        PrintDiagnosticRecord( SQL_HANDLE_ENV, sEnv );
    }
    switch( sState )
    {
        case 4:
            (void)(sXaSwitch->xa_end_entry)( &sXid, 0, TMSUCCESS );
            (void)(sXaSwitch->xa_prepare_entry)( &sXid, 0, TMNOFLAGS );
            (void)(sXaSwitch->xa_commit_entry)( &sXid, 0, TMNOFLAGS );
        case 3:
            (void)testDropTable();
        case 2:
            (void)(sXaSwitch->xa_close_entry)( "", 0, TMNOFLAGS );
        case 1:
            (void)SQLFreeHandle( SQL_HANDLE_ENV, sEnv );
            sEnv = NULL;
        default:
            break;
```

```
    }
    return EXIT_FAILURE;
}
```

33.4 Embedded SQL Reference

This chapter describes the SQL statements which are available only in the embedded SQL applications of GOLDILOCKS. The embedded SQL statements on the source code should have the following syntax.

```
<statement> ::= EXEC SQL <exec sql statement>;
<exec sql statement> ::=
      <embedded SQL statement>
    ¦ <embedded get group_id statement>
    ¦ <embedded specific statement>
    ;

<embedded SQL statement> ::=
    [ AT <db_name> ] [ ATOMIC ] [ FOR <iteration_count> ] <sql statement>
    ;
<embedded get group_id statement> ::=
    [ AT <db_name> ]  <get group_id statement> <sql statement>
<embedded specific statement> ::=
      <autocommit statement>
    ¦ <declare section statement>
    ¦ <include statement>
    ¦ <exception statement>
    ¦ <context statement>
    ¦ <option statement>
    ¦ <allocate statement>
    ¦ <free statement>
    ;

<autocommit statement> ::= [ AT <db_name> ] AUTOCOMMIT { ON ¦ OFF };
<declare section statement> ::= { BEGIN ¦ END } DECLARE SECTION;
<include statement> ::= INCLUDE { SQLCA ¦ <identifier> };
<exception statement> ::= WHENEVER <exception_condition> <exception_action>;
<context statement> ::= CONTEXT <context action>;
<context action> ::=
      ALLOCATE :context_name
    ¦ FREE :context_name
    ¦ USE :context_name
    ¦ USE DEFAULT
    ;
<option statement> ::= OPTION ( <option> );
```

```
<get group_id statement> :: GET GROUPID INTO :group_id;
<allocate statement> ::= [AT <db_name>] ALLOCATE :cursor_variable;
<free statement> ::= [AT <db_name>] FREE :cursor_variable;
```

EXEC SQL ALLOCATE

Feature

It allocates the cursor variable in the embedded SQL.

Syntax

```
EXEC SQL [ AT <db_name> ] ALLOCATE :hostvar
<db_name> ::=
       <identifier>
    | :hostvar
    ;
```

Description

It allocates the memory of the run-time cursor variable. This variable should be allocated after declaring SQL_CURSOR type variable in DECLARE SECTION to allocate the run-time cursor variable. This statement not only allocates the memory but also allocates the client's statement, so it is related to the context. In other words, it can be used only in the context where the cursor variable is allocated.

Example

```
{
    EXEC SQL BEGIN DECLARE SECTION;
    SQL_CURSOR empCursor;
    EXEC SQL END DECLARE SECTION;
    ...
    EXEC SQL AT CONN1 ALLOCATE :empCursor;
    ...
}
```

EXEC SQL AT

Feature

It specifies the connection name applied to the embedded SQL statement.

Syntax

```
EXEC SQL [ AT <db_name> ] ...
<db_name> ::=
      <identifier>
    ¦ :hostvar
    ;
```

Description

The connection name can be specified when connected in the embedded SQL application. The name is used when the embedded SQL statement is performed by using the certain connection.

Example

```
{
    ...
    EXEC SQL AT :conn_name CONNECT :uid IDENTIFIED BY :pwd USING :conn_str;

    EXEC SQL AT :conn_name
        UPDATE EMP
        SET    sal = sal * 1.1
        WHERE  JOB = 'SALES';


    ...
}
```

For More Information

Refer to **Connection**.

EXEC SQL ATOMIC INSERT

Feature

It performs the atomic array insert.

Syntax

```
EXEC SQL ATOMIC <insert_statement>;
```

Description

Atomic array insert is performed in an embedded SQL application. Atomic array insert is for inserting mult iple rows at once. It succeeds only when the entire row is inserted. If any row fails to be inserted, the inse rtion of the entire row is failed.
It has a better performance than the individual insertion because the insertion is performed with a single command.

Example

```
{
    EXEC SQL BEGIN DECLARE SECTION;
    int    emp_number[20];
    char   emp_name[20][10];
    int    dept_number[20];
    EXEC SQL END DECLARE SECTION;
```

- Setting the values of emp_number, emp_name, dept_number

```
...
```

- Inserting the values of emp_number, emp_name, dept_number

```
EXEC SQL ATOMIC INSERT INTO emp (empno, ename, deptno)
                            VALUES (:emp_number, :emp_name, :dept_number);
}
```

For More Information

Refer to **Atomic Insert**.

EXEC SQL AUTOCOMMIT

Feature

It changes the autocommit setting.

Syntax

```
EXEC SQL AUTOCOMMIT { ON | OFF };
```

Description

It sets the autocommit as follows.

Table 33-24 Setting the autocommit

Flag	Description
ON	Statement is executed and then autocommit is performed.
OFF	Commit is not performed until the explicit commit statement comes.

Example

```
{
    ...
    EXEC SQL AT :conn_name CONNECT :uid IDENTIFIED BY :pwd USING :conn_str;

    EXEC SQL AUTOCOMMIT ON;
    EXEC SQL AT :conn_name
        UPDATE EMP
        SET    sal = sal * 1.1
        WHERE  JOB = 'SALES';

    ...
}
```

For More Information

Refer to **Auto Commit**.

EXEC SQL BEGIN DECLARE SECTION

Feature

It is a precompiler indicator, and it specifies the host variable declaration area.

Syntax

```
EXEC SQL BEGIN DECLARE SECTION;
```

Description

It is a precompiler indicator which specifies the host variable declaration section, and it is always used together with EXEC SQL END DECLARE SECTION. When precompiler encounters this statement, it is regarded as the start of declare section, and the variable thereafter is processed as the host variable.

Example

```
{
    EXEC SQL BEGIN DECLARE SECTION;
    int    emp_number[20];
    char   emp_name[20][10];
    int    dept_number[20];
    EXEC SQL END DECLARE SECTION;

    ...
}
```

For More Information

Refer to the followings.

- **Declaring Host Variable**
- **EXEC SQL END DECLARE SECTION**

EXEC SQL COMMIT RELEASE

Feature

It commits the transaction and terminates the current connection.

Syntax

```
EXEC SQL [ AT <db_name> ] COMMIT [ WORK ] RELEASE;
```

Description

It commits the transaction and terminates the current connection.

```
EXEC SQL AT :conn_name COMMIT RELEASE;
```

The statement above is as same as the following.

```
EXEC SQL AT :conn_name COMMIT;
EXEC SQL AT :conn_name DISCONNECT;
```

Example

```
{
    ...
    EXEC SQL AT :conn_name CONNECT :uid IDENTIFIED BY :pwd USING :conn_str;

    EXEC SQL AT :conn_name
        UPDATE EMP
        SET    sal = sal * 1.1
        WHERE  JOB = 'SALES';
    EXEC SQL AT :conn_name COMMIT RELEASE;
    ...
}
```

For More Information

Refer to **RELEASE Option**.

EXEC SQL CONNECT

Feature

It connects to GOLDILOCKS.

Syntax

```
EXEC SQL [ AT <db_name> ] CONNECT <user_name> IDENTIFIED BY <password> [ AT <db_name> ] [
USING <conn_string> ]
<db_name> ::=
      <identifier>
    ¦ :hostvar
    ;
<user_name> ::=
      <identifier>
    ¦ :hostvar
    ;
<password> ::=
      <identifier>
    ¦ :hostvar
    ;
<conn_string> ::= :hostvar;
```

Description

It sets the connection to GOLDILOCKS.

Example

```
{
    ...
    EXEC SQL CONNECT :uid IDENTIFIED BY :pwd USING :conn_str;


    ...
}
```

For More Information

Refer to **Connecting to Database**.

EXEC SQL CONTEXT ALLOCATE

Feature

It allocates the run-time context memory.

Syntax

```
EXEC SQL CONTEXT ALLOCATE :context;
```

Description

It allocates run-time context memory. It should be allocated for variable of SQL_CONTEXT type after the v ariable of SQL_CONTEXT type is declared in declare section to allocate the run-time context. This stateme nt allocates the memory only, so to use it, USE should be specified and perform the connect.

Example

```
{
    ...
    EXEC SQL BEGIN DECLARE SECTION;
    SQL_CONTEXT ctxt;
    EXEC SQL END DECLARE SECTION;
    EXEC SQL CONTEXT ALLOCATE :ctxt;

    EXEC SQL CONTEXT USE :ctxt;
    EXEC SQL CONNECT :uid IDENTIFIED BY :pwd USING :conn_str;

    EXEC SQL
        UPDATE EMP
        SET    sal = sal * 1.1
        WHERE  JOB = 'SALES';
    EXEC SQL DISCONNECT;
    EXEC SQL CONTEXT FREE :ctxt;

    ...
}
```

For More Information

Refer to **SQL_CONTEXT**.

EXEC SQL CONTEXT FREE

Feature

It frees the run-time context memory.

Syntax

```
EXEC SQL CONTEXT FREE :context;
```

Description

It frees run-time context memory. Disconnect should be performed before freeing the run-time context s o that the connection will not be used any more. Otherwise, an unexpected error may occur.

Example

```
{
    ...
    EXEC SQL BEGIN DECLARE SECTION;
    SQL_CONTEXT ctxt;
    EXEC SQL END DECLARE SECTION;
    EXEC SQL CONTEXT ALLOCATE :ctxt;

    EXEC SQL CONTEXT USE :ctxt;
    EXEC SQL CONNECT :uid IDENTIFIED BY :pwd USING :conn_str;

    EXEC SQL
        UPDATE EMP
        SET    sal = sal * 1.1
        WHERE  JOB = 'SALES';
    EXEC SQL DISCONNECT;
    EXEC SQL CONTEXT FREE :ctxt;

    ...
}
```

For More Information

Refer to **SQL_CONTEXT**.

EXEC SQL CONTEXT USE

Feature

It informs the use of run-time context.

Syntax

```
EXEC SQL CONTEXT USE { :context ¦ DEFAULT };
```

Description

It is an indicator which informs the precompiler of the use of run-time context, and it specifies the run-time context to be used. The run-time context which is declared and allocated by a user can be used by using SQL_CONTEXT variable in USE statement, and the default context in the application is used when performing the USE DEFAULT statement.

Example

```
{
    ...
    EXEC SQL BEGIN DECLARE SECTION;
    SQL_CONTEXT ctxt;
    double      max_sal;
    EXEC SQL END DECLARE SECTION;
    EXEC SQL CONTEXT ALLOCATE :ctxt;

    EXEC SQL CONTEXT USE :ctxt;
    EXEC SQL CONNECT :uid IDENTIFIED BY :pwd USING :conn_str;

    EXEC SQL
        UPDATE EMP
        SET    sal = sal * 1.1
        WHERE  JOB = 'SALES';
    EXEC SQL COMMIT RELEASE;
    EXEC SQL CONTEXT FREE :ctxt;

    EXEC SQL CONTEXT USE DEFAULT;
    EXEC SQL CONNECT :uid IDENTIFIED BY :pwd USING :conn_str;
```

```
    EXEC SQL
        SELECT MAX(sal)
        INTO   max_sal
        FROM   EMP
        WHERE  JOB = 'SALES';
    EXEC SQL DISCONNECT;
    ...
}
```

For More Information

Refer to **SQL_CONTEXT**.

EXEC SQL DISCONNECT

Feature

It disconnects from GOLDILOCKS.

Syntax

```
EXEC SQL [ AT <db_name> ] DISCONNECT [ ALL ]
```

Description

It disconnects from GOLDILOCKS. A specific connection can be freed with AT clause, and the current connection is freed if AT clause is not used. All connections which are used in the current application are freed by using DISCONNECT ALL.

Example

```
{
    ...
    EXEC SQL AT :conn_name DISCONNECT;

    ...
}
```

For More Information

Refer to **Disconnecting Database**.

EXEC SQL END DECLARE SECTION

Feature

It is a precompiler indicator, and it specifies the the host variable declaration section.

Syntax

```
EXEC SQL END DECLARE SECTION;
```

Description

It is a precompiler indicator which specifies the host variable declaration section, and it is always used tog ether with EXEC SQL BEGIN DECLARE SECTION. When precompiler encounters this statement during ana lyzing the declare section, it is regarded as the end of the declare section.

Example

```
{
    EXEC SQL BEGIN DECLARE SECTION;
    int    emp_number[20];
    char   emp_name[20][10];
    int    dept_number[20];
    EXEC SQL END DECLARE SECTION;

    ...
}
```

For More Information

Refer to the followings.

- **Declaring Host Variable**
- **EXEC SQL BEGIN DECLARE SECTION**

EXEC SQL FOR

Feature

It specifies the number of arrays in array operation.

Syntax

```
EXEC SQL FOR { :array_count ¦ integer_constant } <sql statement>;
```

Description

If the host variable of the SQL statement is an array, it is a precompiler indicator which specifies the array count. When FOR clause is given, the array count of the host array is ignored and the arrays as many as th e number specified in FOR clause are performed.

The constant or variable which refers to array count should be an integer.

Example

```
EXEC SQL BEGIN DECLARE SECTION;
int    emp_number[20];
char   emp_name[20][10];
int    dept_number[20];
int    record_cnt;
EXEC SQL END DECLARE SECTION;
```

- Setting the values of emp_number, emp_name, dept_number

```
...
```

- Inserting emp_number, emp_name, dept_number

```
record_cnt = 10;
EXEC SQL FOR :record_cnt INSERT INTO emp (empno, ename, deptno)
    VALUES (:emp_number, :emp_name, :dept_number);
```

For More Information

Refer to **Using FOR Clause**.

EXEC SQL FREE

Feature

It frees the cursor variable in the embedded SQL.

Syntax

```
EXEC SQL [ AT <db_name> ] FREE :hostvar
<db_name> ::=
       <identifier>
    ¦ :hostvar
    ;
```

Description

It frees the memory of the run-time cursor variable. If the cursor variable is not explicitly freed, then it is fr eed when the context is disconnected or the program is terminated. The cursor variable is related to the a llocated context, so FREE command is also related to it.

Example

```
{
    EXEC SQL BEGIN DECLARE SECTION;
    SQL_CURSOR empCursor;
    EXEC SQL END DECLARE SECTION;
    ...
    EXEC SQL AT CONN1 FREE :empCursor;
    ...
}
```

EXEC SQL GET GROUPID INTO

Feature

It obtains the group ID of the SQL statement.

Syntax

```
EXEC SQL [ AT <db_name> ] GET GROUPID INTO :group_id { delete_stmt ¦ insert_stmt ¦ select_stmt
¦ update_stmt };
```

Description

It obtains the the group ID of the SQL statement in the cluster environment which uses the global connection. Only the signed numeric type is allowed for the host variable :group_id. The group ID can be obtained only for the delete, insert, select, update SQL statement, and the shard key should be set in the table in advance.

The SQL statement which obtained the group ID is internally cached in SQLPrepare status without executing SQLExecute.

-1 which is an invalid group ID value can be returned.

Example

```
{
EXEC SQL BEGIN DECLARE SECTION;
int  group_id[10];
int  emp_no[10];
char emp_name[10][20];
int  dept_no[10];
EXEC SQL END DECLARE SECTION;
```

- Set emp_no, emp_name, dept_no values

```
...
```

- Obtain the group id.

```
EXEC SQL GET GROUPID INTO :group_id
         INSERT INTO emp (empno, ename, deptno) VALUES (:emp_no, :emp_name, :dept_no);
```

- INSERT emp_no, emp_name, dept_no.

```
EXEC SQL INSERT INTO emp (empno, ename, deptno) VALUES (:emp_no, :emp_name, :dept_no);
}
```

EXEC SQL INCLUDE

Feature

It includes the embedded SQL header file.

Syntax

```
EXEC SQL INCLUDE <Header file name>;
```

Description

It includes the embedded SQL header file. If the header file is included with #include of C language, the p recompiler does not interpret it, so the precompiler can not recognize it even when the information whic h the precompiler should recognize such as declare section in the header file is included in it. Therefore, t he EXEC SQL INCLUDE statement should be used when the embedded SQL statement which the precom piler should recognize is used.

Example

```
EXEC SQL INCLUDE decl.h;
```

For More Information

Refer to **Precompiled Header File**.

EXEC SQL INCLUDE SQLCA

Feature

It includes sqlca.h header file.

Syntax

```
EXEC SQL INCLUDE SQLCA;
```

Description

It is a special type of EXEC SQL INCLUDE statement, and it includes sqlca.h header file supported by GOL DILOCKS. This header file is required for run-time exception handling in the embedded SQL application.

Example

```
EXEC SQL INCLUDE SQLCA;
```

For More Information

Refer to **Detecting Run-time Error**.

EXEC SQL OPTION

Feature

It applies the option to the process of precompiling the embedded SQL source code.

Syntax

```
EXEC SQL OPTION ( <option_desc> );
<option_desc> ::=
      INCLUDE = <directory path>
    ;
```

Description

It describes the option which is applied to the process of precompiling the embedded SQL source code. It supports only specifying INCLUDE path in the current version, and the option specifies the directory of the header file to be precompiled in EXEC SQL INCLUDE.

Example

```
EXEC SQL OPTION ( INCLUDE = include );
```

For More Information

Refer to **Specifying Header File Path**.

EXEC SQL ROLLBACK RELEASE

Feature

It rolls back the transaction, then releases the connection.

Syntax

```
EXEC SQL [ AT <db_name> ] ROLLBACK [ WORK ] RELEASE;
```

Description

It rolls back the transaction, then releases the current connection.

```
EXEC SQL AT :conn_name ROLLBACK RELEASE;
```

The statement above is as same as the following.

```
EXEC SQL AT :conn_name ROLLBACK;
EXEC SQL AT :conn_name DISCONNECT;
```

Example

```
{
    ...
    EXEC SQL AT :conn_name CONNECT :uid IDENTIFIED BY :pwd USING :conn_str;

    EXEC SQL AT :conn_name
        UPDATE EMP
        SET    sal = sal * 1.1
        WHERE  JOB = 'SALES';
    EXEC SQL AT :conn_name ROLLBACK RELEASE;
    ...
}
```

For More Information

Refer to **RELEASE Option**.

EXEC SQL WHENEVER

Feature

It executes run-time exception handling in the embedded SQL application.

Syntax

```
EXEC SQL WHENEVER <conditions> <actions>;
<conditions> ::=
      SQLERROR
    ¦ SQLWARNING
    ¦ NOT FOUND
    ¦ SQLSTATE <sqlstate class value>[<sqlstate subclass value>]
    ;
<sqlstate_char> ::= [0-9A-Z];
<sqlstate class value> ::= <sqlstate_char> <sqlstate_char>;
<sqlstate subclass value> ::= <sqlstate_char> <sqlstate_char> <sqlstate_char>;
<actions> ::=
      CONTINUE
    ¦ GOTO <label>
    ¦ STOP
    ¦ DO <c statements>
    ;
```

Description

It automates and processes the run-time exception handling in an embedded SQL application. There are f our conditions, a single action can be specified per each condition. The action can be reassigned as neede d. For more information, refer to **Handling Implicit Error**.

Example

```
EXEC SQL WHENEVER SQLERROR STOP;
EXEC SQL WHENEVER SQLERROR CONTINUE;
EXEC SQL WHENEVER SQLERROR GOTO exit_label;
EXEC SQL WHENEVER NOT FOUND DO break;
EXEC SQL WHENEVER SQLERROR GOTO close_label;
EXEC SQL WHENEVER SQLERROR DO sql_error();
```

```
EXEC SQL WHENEVER SQLWARNING CONTINUE;
EXEC SQL WHENEVER SQLSTATE HY000 DO sql_error();
```

For More Information

Refer to **Handling Implicit Error**.

34.

PDO

34.1 Overview of PDO

GOLDILOCKS PDO (PDO_GOLDILOCKS) driver provides PDO interface to access GOLDILOCKS database in PDO, and it is provides in a package.

PDO_GOLDILOCKS is built based on GOLDILOCKS CLI driver.

34.2 Installation /Configuration

Requirement

The client package of GOLDILOCKS should be installed in the system which is as same as that of PHP, and environment variables of $GOLDILOCKS_HOME should be set.

Installation

Installation by Using PECL

Create the package file if PDO_GOLDILOCKS package file does not exist.

```
% pecl pickle
Attempting to process the second package file
Package PDO_GOLDILOCKS-3.2.0.tgz done
```

Install the package by using PECL if PDO_GOLDILOCKS package file exists.

```
% sudo -E pecl install PDO_GOLDILOCKS-3.2.0.tgz
```

Installation by Building the Source

1. Decompress PDO_GOLDILOCKS.

```
% tar xvzf PDO_GOLDILOCKS-3.2.0.tgz
% cd PDO_GOLDILOCKS-3.2.0
```

2. Create the environment for the build.

Create the build environment for PDO_GOLDILOCKS expanded module by using phpize command.
If phpize command can not be found, PHP devel package in an appropriate version should be installed.

```
% whereis phpize
phpize: /usr/bin/phpize /usr/share/man/man1/phpize.1.gz
```

```
% /usr/bin/phpize
Configuring for:
PHP Api Version:         20121113
Zend Module Api No:      20121212
Zend Extension Api No:   220121212
```

3. Build it.

Execute *configure* by adding the path of *$GOLDILOCKS_HOME* to *--with-pdo-goldilocks* option and addi ng the path of *php-config* to *--with-php-config* option, then build it.

```
% whereis php-config
php-config: /usr/bin/php-config /usr/share/man/man1/php-config.1.gz
```

```
% ./configure --with-php-config=/usr/bin/php-config
--with-pdo-goldilocks=/home/goldilocks/goldilocks_home/
```

```
% make
```

4. Install it.

```
% sudo -E make install
```

Altering PHP Configuration File

When using CentOS 6.0 or higher or Fedora 15 or higher, the pdo_goldilocks.ini file should be created in /etc/php.d directory and extension property should be added as follows.

```
extension=pdo_goldilocks.so
```

When using other OS extension_dir should be appropriately set in php.ini file and add extension property should be added as follows.

```
extension=pdo_goldilocks.so
```

Restarting Web Server

Restart the web server.

Checking Installation of PDO_GOLDILOCKS

Check whether PDO_GOLDILOCKS is activated by using phpinfo().

```
% php -i | grep PDO
PDO
PDO support => enabled
PDO drivers => goldilocks, sqlite
PDO Driver for GOLDILOCKS => enabled
PDO Driver for SQLite 3.x => enabled
```

34.3 Usage

Data Source Name (DSN)

Data Source Name (DSN) of PDO_GOLDILOCKS is configured as follows.

Property	Description
DSN prefix	goldilocks
DSN	ODBC Data Source Name (DSN)
HOST	IP address of a server
PORT	PORT number of server
UID	User ID
PWD	User password

34.4 Examples

- Connecting to GOLDILOCKS

```
print "\n[Connecting to GOLDILOCKS]\n";
$db = new PDO('goldilocks:HOST=192.168.0.16;PORT=22581', 'test', 'test');
//$db = new PDO('goldilocks:DSN=GOLDILOCKS;UID=test;PWD=test');
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

- Set test data

```
$result = $db->exec('DROP TABLE IF EXISTS t1');
$result = $db->exec('CREATE TABLE t1 ( id INTEGER GENERATED BY DEFAULT AS IDENTITY, name
VARCHAR(32) )');
$result = $db->exec("INSERT INTO t1(name) VALUES ('Carol')");
$result = $db->exec("INSERT INTO t1(name) VALUES ('Ted')");
$result = $db->exec("INSERT INTO t1(name) VALUES (null)");
$result = $db->exec("INSERT INTO t1(name) VALUES ('William')");
$result = $db->exec("INSERT INTO t1(name) VALUES ('Chelsea')");
$result = $db->exec("INSERT INTO t1(name) VALUES ('Colin')");
$result = $db->exec('DROP TABLE IF EXISTS t2');
$result = $db->exec('CREATE TABLE t2 ( id INTEGER, name VARCHAR(32) )');
$result = $db->exec("INSERT INTO t2 VALUES (1, 'John')");
$result = $db->exec("INSERT INTO t2 VALUES (1, 'John')");
$result = $db->exec("INSERT INTO t2 VALUES (1, 'John')");
$result = $db->exec("INSERT INTO t2 VALUES (2, 'Smith')");
$result = $db->exec("INSERT INTO t2 VALUES (2, 'Smith')");
$result = $db->exec('DROP TABLE IF EXISTS images');
$result = $db->exec('CREATE TABLE images ( id INTEGER, contenttype VARCHAR(256), imagedata
LONG VARBINARY )');
$result = $db->exec('DROP PROCEDURE IF EXISTS sp_out_string');
$result = $db->exec("CREATE OR REPLACE PROCEDURE sp_out_string( v1 OUT VARCHAR(32) ) ".
                    "IS ".
                    "BEGIN ".
                    "  v1 := 'WORLD'; ".
                    "END; ");
$result = $db->exec('DROP PROCEDURE IF EXISTS sp_inout_string');
$result = $db->exec("CREATE OR REPLACE PROCEDURE sp_inout_string( v1 INOUT VARCHAR(32) ) ".
                    "IS ".
                    "BEGIN ".
```

```
                    "  v1 := V1 || ' WORLD'; ".
                    "END; ");
```

- Quoting a normal string

```
print "\n[Quoting a normal string]\n";
$string = 'Nice';
print "Unquoted string: $string\n";
print "Quoted string  : " . $db->quote($string) . "\n";
```

- Quoting a dangerous string

```
print "\n[Quoting a dangerous string]\n";
$string = 'Naughty \' string';
print "Unquoted string: $string\n";
print "Quoted string  :" . $db->quote($string) . "\n";
```

- Quoting a complex string

```
print "\n[Quoting a complex string]\n";
$string = "Co'mpl''ex \"st'\"ring";
print "Unquoted string: $string\n";
print "Quoted string  : " . $db->quote($string) . "\n";
```

- Handling an error

```
print "\n[Error Handling]\n";
try
{
    //connect as appropriate as above
    $db->query('invalid query'); //invalid query!
}
catch(PDOException $ex)
{
    print "ERROR:".$ex->getMessage()."\n";
}
```

- Retrieving column metadata

```
print "\n[Retrieving column metadata]\n";
$stmt = $db->query('SELECT * FROM t1');
foreach(range(0, $stmt->columnCount() - 1) as $column_index)
```

```
{
    $meta = $stmt->getColumnMeta($column_index);
    var_dump($meta);
}
```

- Running simple select statements

```
print "\n[Running Simple Select Statements]\n";
print "\n#1\n";
foreach($db->query('SELECT * FROM t1') as $row)
{
    print $row['ID'].' '.$row['NAME']."\n";
}
print "\n#2\n";
$stmt = $db->query('SELECT * FROM t1');

while($row = $stmt->fetch(PDO::FETCH_ASSOC))
{
    print $row['ID'].' '.$row['NAME']."\n";
}
```

- Fetching rows using different fetch styles

```
print "\n[Fetching rows using different fetch styles]\n";
$stmt = $db->prepare("SELECT id, name FROM t1");
$stmt->execute();
print("PDO::FETCH_ASSOC: ");
print("Return next row as an array indexed by column name\n");
$result = $stmt->fetch(PDO::FETCH_ASSOC);
print_r($result);
print("\n");
print("PDO::FETCH_BOTH: ");
print("Return next row as an array indexed by both column name and number\n");
$result = $stmt->fetch(PDO::FETCH_BOTH);
print_r($result);
print("\n");
print("PDO::FETCH_LAZY: ");
print("Return next row as an anonymous object with column names as properties\n");
$result = $stmt->fetch(PDO::FETCH_LAZY);
print_r($result);
print("\n");
print("PDO::FETCH_OBJ: ");
```

```
print("Return next row as an anonymous object with column names as properties\n");
$result = $stmt->fetch(PDO::FETCH_OBJ);
print $result->NAME;
print("\n");
```

- Fetching rows with a scrollable cursor

```
print "\n[Fetching rows using different fetch styles]\n";
print "\n[Fetching rows with a scrollable cursor]\n";
$sql = 'SELECT id, name FROM t1 ORDER BY id';
print "Reading forwards:\n";
try
{
    $stmt = $db->prepare($sql, array(PDO::ATTR_CURSOR => PDO::CURSOR_SCROLL));
    $stmt->execute();
    while( $row = $stmt->fetch(PDO::FETCH_NUM, PDO::FETCH_ORI_NEXT) )
    {
        $data = $row[0] . "\t" . $row[1] . "\n";
        print $data;
    }
    $stmt = null;
}
catch (PDOException $e)
{
    print $e->getMessage();
}
print "Reading backwards:\n";
try
{
    $stmt = $db->prepare($sql, array(PDO::ATTR_CURSOR => PDO::CURSOR_SCROLL));
    $stmt->execute();
    $row = $stmt->fetch(PDO::FETCH_NUM, PDO::FETCH_ORI_LAST);
    do
    {
        $data = $row[0] . "\t" . $row[1] . "\n";
        print $data;
    } while( $row = $stmt->fetch(PDO::FETCH_NUM, PDO::FETCH_ORI_PRIOR) );
    $stmt = null;
}
catch (PDOException $e)
{
```

```
    print $e->getMessage();
}
```

- Constructing an order

```
print "\n[Construction order]\n";
class Person
{
    private $NAME;
    public function __construct()
    {
        $this->tell();
    }
    public function tell()
    {
        if (isset($this->NAME))
        {
            print "I am {$this->NAME}.\n";
        }
        else
        {
            print "I don't have a name yet.\n";
        }
    }
}
$stmt = $db->query("SELECT name FROM t1");
$stmt->setFetchMode(PDO::FETCH_CLASS, 'Person');
$person = $stmt->fetch();
$person->tell();
$stmt->setFetchMode(PDO::FETCH_CLASS¦PDO::FETCH_PROPS_LATE, 'Person');
$person = $stmt->fetch();
$person->tell();
```

- Running simple INSERT, UPDATE, or DELETE statements

```
print "\n[Running Simple INSERT, UPDATE, or DELETE statements]\n";
$affected_rows = $db->exec("INSERT INTO t1(name) VALUES ('John'), ('Marry')");
print $affected_rows." were affected\n";
```

- Getting the last insert id

```
print "\n[Getting the Last Insert Id]\n";
$affected_rows = $db->exec("INSERT INTO t1(name) VALUES ('Smith')");
$insertId = $db->lastInsertId();
print "last insert id : ".$insertId."\n";
```

- Running statements with parameters

```
print "\n[Running Statements With Parameters]\n";
print "\n#1 array\n";
$id = 1;
$name = 'John';
$stmt = $db->prepare("SELECT * FROM t2 WHERE id=? AND name=?");
$stmt->execute(array($id, $name));
$rows = $stmt->fetchAll(PDO::FETCH_ASSOC);
var_dump($rows);
print "\n#2 bindValue\n";
$id = 2;
$name = 'Smith';
$stmt = $db->prepare("SELECT * FROM t2 WHERE id=? AND name=?");
$stmt->bindValue(1, $id, PDO::PARAM_INT);
$stmt->bindValue(2, $name, PDO::PARAM_STR);
$stmt->execute();
$rows = $stmt->fetchAll(PDO::FETCH_ASSOC);
var_dump($rows);
```

- Named placeholders

```
print "\n[Named Placeholders]\n";
print "\n#1 array\n";
$id = 1;
$name = 'John';
$stmt = $db->prepare("SELECT * FROM t2 WHERE id=:id AND name=:name");
$stmt->execute(array(':name' => $name, ':id' => $id));
$rows = $stmt->fetchAll(PDO::FETCH_ASSOC);
var_dump($rows);
print "\n#2 bindValue\n";
$id = 2;
$name = 'Smith';
$stmt = $db->prepare("SELECT * FROM t2 WHERE id=:id AND name=:name");
$stmt->bindValue(':id', $id, PDO::PARAM_INT);
$stmt->bindValue(':name', $name, PDO::PARAM_STR);
```

```
$stmt->execute();
$rows = $stmt->fetchAll(PDO::FETCH_ASSOC);
var_dump($rows);
```

- Executing prepared statements in a loop

```
print "\n[Executing prepared statements in a loop]\n";
$values = array('bob', 'alice', 'lisa', 'john');
$name = '';
$stmt = $db->prepare("INSERT INTO t1(name) VALUES(:name)");
$stmt->bindParam(':name', $name, PDO::PARAM_STR);
foreach($values as $name)
{
   $stmt->execute();
}
foreach($db->query('SELECT * FROM t1') as $row)
{
    print $row[0].' '.$row[1]."\n";
}
```

- Calling a stored procedure with an output parameter

```
print "\n[Calling a stored procedure with an output parameter]\n";
$stmt = $db->prepare("CALL sp_out_string(?)");
$stmt->bindParam(1, $value, PDO::PARAM_STR, 32);
$stmt->execute();
print "procedure returned $value\n";
```

- Calling a stored procedure with an input/output parameter

```
print "\n[Calling a stored procedure with an input/output parameter]\n";
$stmt = $db->prepare("CALL sp_inout_string(?)");
$value = 'HELLO';
$stmt->bindParam(1, $value, PDO::PARAM_STR¦PDO::PARAM_INPUT_OUTPUT, 32);
$stmt->execute();
print "procedure returned $value\n";
```

- Transactions

```
print "\n[Transactions]\n";
print "\n#1 commit\n";
try
```

```
{
    $db->beginTransaction();

    $db->exec("INSERT INTO t1(name) VALUES('kim')");
    $name = 'lee';
    $stmt = $db->prepare("INSERT INTO t1(name) VALUES(?)");
    $stmt->execute(array($name));

    $id = 3;
    $name = 'park';
    $stmt = $db->prepare("INSERT INTO t2 VALUES(?, ?)");
    $stmt->execute(array($id, $name));

    $db->commit();
}
catch(PDOException $ex)
{
    //Something went wrong rollback!
    $db->rollBack();
    print $ex->getMessage()."\n";
}
print "\nT1\n";
foreach($db->query('SELECT * FROM t1') as $row)
{
    print $row[0].' '.$row[1]."\n";
}
print "\nT2\n";
foreach($db->query('SELECT * FROM t2') as $row)
{
    print $row[0].' '.$row[1]."\n";
}
print "\n#2 rollback\n";
try
{
    $db->beginTransaction();

    $db->exec("INSERT INTO t1(name) VALUES('KIM')");
    $name = 'LEE';
    $stmt = $db->prepare("INSERT INTO t1(name) VALUES(?)");
    $stmt->execute(array($name));
```

```
    $id = 3;
    $name = 'PARK';
    $stmt = $db->prepare("INSERT INTO invalid VALUES(?, ?)");
    $stmt->execute(array($id, $name));

    $db->commit();
}
catch(PDOException $ex)
{
    //Something went wrong rollback!
    $db->rollBack();
    print $ex->getMessage()."\n";
}
print "\nT1\n";
foreach($db->query('SELECT * FROM t1') as $row)
{
    print $row[0].' '.$row[1]."\n";
}
print "\nT2\n";
foreach($db->query('SELECT * FROM t2') as $row)
{
    print $row[0].' '.$row[1]."\n";
}
```

- Inserting an image into a database

```
print "\n[Inserting an image into a database]\n";
try
{
    $stmt = $db->prepare("INSERT INTO images (id, contenttype, imagedata) VALUES (?, ?, ?)");
    $id = 1;
    $type = "image/png";
    $fp = fopen("logo.png", "rb");
    $stmt->bindParam(1, $id, PDO::PARAM_INT);
    $stmt->bindParam(2, $type, PDO::PARAM_STR, 256);
    $stmt->bindParam(3, $fp, PDO::PARAM_LOB);
    $db->beginTransaction();
    $stmt->execute();
    $db->commit();
}
catch(PDOException $ex)
```

```
{
    $db->rollBack();
    print $ex->getMessage()."\n";
}
foreach($db->query('SELECT id, contenttype, lengthb(imagedata) FROM images') as $row)
{
    print $row[0].' '.$row[1].' '.$row[2]."\n";
}
```

- Displaying an image from a database

```
print "\n[Displaying an image from a database]\n";
try
{
    $stmt = $db->prepare("SELECT contenttype, imagedata FROM images WHERE id = ?");
    $id = 1;
    $stmt->execute(array($id));
    $stmt->bindColumn(1, $type, PDO::PARAM_STR, 256);
    $stmt->bindColumn(2, $lob, PDO::PARAM_LOB);
    $stmt->fetch(PDO::FETCH_BOUND);
    print 'Content-Type: '.$type."\n";
    print 'size: '.file_put_contents($id.".png", $lob)."\n";
}
catch(PDOException $ex)
{
    print $ex->getMessage()."\n";
}
```

35.

PyDBC

35.1 GOLDILOCKS PyDBC

Overview

PyDBC programs Python accessing GOLDILOCKS database by using API which complies with **Python Database API Specification v2.0(PEP 249)**.

PyDBC requires the Python standard library, and the internal operation which connects to and operate GOLDILOCKS database requires ODBC library because it calls ODBC API. PyDBC uses gdlcs in ODBC library *$GOLDILOCKS_HOME/lib* by default, and the user can modify it by updating *setup.py*.

The internal operation of PyDBC uses ODBC driver, so it is as same as **Overview of ODBC Components**. There are an architecture of which an application links to the driver manager, and an architecture of which an application links to GOLDILOCKS ODBC driver library.

Version

The information of GOLDILOCKS PyDBC version can be viewed by executing *pygoldilocks.so* file as follows.

```
shell>python
>>> import pygoldilocks
>>> print pygoldilocks.version
3.2.0
```

The current version of GOLDILOCKS PyDBC driver is 3.2.0 according to GOLDILOCKS version, and this driver complies with the standard Python database API 2.0. PyDBC driver supports Python 2.7, 3.4, 3.5, 3.6 versions, and PyDBC driver library should be installed according to each Python version.

Installation

The source should be built to install PyDBC. PyDBC links gdlcs library in GOLDILOCKS_HOME/lib and it includes *goldilocks.h* header file in GOLDILOCKS_HOME/include, so the environment variable GOLDILOCKS_HOME should be set in an appropriate position.

When installing PyDBC, the bit of Python should be as same as that of GOLDILOCKS library. Therefore, if GOLDILOCKS is built in 32 bit, then PyDBC should be installed by using Python 32 bit.

Installation on Linux

Linux requires the gcc compiler, and it is built as follows.

```
shell> sudo python setup.py install
```


It does not support HP-UX nor AIX platform.

Installation on Windows

It is built on Windows as follows.

```
shell> python setup.py install
```

An appropriate Microsoft Visual C++ compiler according to python version is required to compile PyDBC For more information, refer to **https://wiki.python.org/moin/WindowsCompilers**.

- Visual Studio 2003.NET compiler is required to build Python version 2.4 or 2.5, and there is not a free ware for this version.
- Visual C++ 2008 compiler is required to build Python version 2.6, 2.7, 3.0, 3.2, and the freeware for t his version is Visual C++ 2008 Express.
- Visual C++ 2010 compiler is required to build Python version 3.3, 3.4, and the freeware for this versi on is Visual C++ 2010 Express.
- Visual C++ 2014 or VC 2017 compiler is rerquired to build Python version 3.5, 3.6.
- Visual C++ 2017 compiler is required to build Python version 3.7.

Examples

Obtaining Connection Class

PyDBC implicitly calls ODBC library. Therefore, **the data source** should be configured to obtain the connec tion.

- Obtain the connection as follows.

```
import pygoldilocks
cnxn = pygoldilocks.connect( 'DSN=GOLDILOCKS;UID=test;PWD=test' )
```

Call connect which is an internal function of pygoldilocks, a module of PyDBC, to obtain the connection.

Data Source should be configured in advance to use DSN.

If CHARSET is not set in DSN, then ODBC library sets it to Console Character Set. PyDBC implicitly perform s the basic encoding for the character set used by ODBC. If the character set used in GOLDILOCKS server i s different from that in ODBC library, then the data conversion occurs and it degrades the performance. F or example, CP949 is used as a basic character set in Windows. If nothing is set, PyDBC implicitly uses CP 949(UHC) when encoding, and ODBC library also uses UHC when processing the character set.

The followings are how to alter the character set of the client.

- Alter CHARSET property in **Data Source Configuration**.
- Add CHARSET property to the connection string.
- Use attrs_before keyword in connect method of pygoldilocks module.

All of the three methods above alter the connection property of ODBC, SQL_ATTR_CHARACTER_SET, an d set the encoding of PyDBC library as well.

Using Cursor and Row Class

A cursor and a row class can be used as follows.

```
cursor = cnxn.cursor()
cursor.execute( "SELECT NAME, ADDRESS FROM EMP" )
rows = cursor.fetchall()
for row in rows:
    print row.A, row.B
cursor.close()
cnxn.close()
```

35.2 API Reference

pygoldilocks Module

pygoldilocks object complies with **Python Database API Specification v2.0**.
For more information, refer to **Python DB API module**.

Properties

- version
  - The version of pygoldilocks module follows that of GOLDILOCKS database. The version is a string in a form of major.minor.patch.

- apilevel
  - It indicates DB API level 2.0, and the value is "2.0" character string constant.

- lowercase
  - It controls whether to convert the column name from the row object of the result value to lowerc ase. The default value is false. It is useful when the case of database column does not conform.

- threadsafety
  - It is constant 1, and it does not share the connection even when threads share the module.

- paramstyle
  - It indicates a parameter and the value is the character string constant "qmark" which means a qu estion mark.

connect

It newly connects to the database.

```
connect( *connectionstring, **kwargs )
```

It inputs the ODBC connect string and keywords. The keywords are as follows.

Keyword	Description	Default value
attrs_before	It sets properties which should be set before the connection. It receives the value in dictionary type.	-
autocommit	It sets whether to auto commit. If it is false, connection.commit should be c alled to reflect it in the database.	False

Keyword	Description	Default value
readonly	If it is true, the connection is set to readonly.	False
timeout	It sets the timeout for the connection. SQL_ATTR_LOGIN_TIMEOUT is set.	-

- attrs_before
  - It sets options which is to be set before the connection. These options are set by using **SQLSetConnectAttr**. It receives the properties and values in dictionary type. For more information about configuration, refer to **ODBC attributes**.

```
cnxn =  pygoldilocks.connect( "DSN=GOLDILOCKS", attr_before={ pygoldilocks.SQL_ATTR_MAX_ROWS : 1000 })
```

Date

```
>>> print pygoldilocks.Date(1984,11,23),  type(pygoldilocks.Date(1984,11,23))
1984-11-23 <type 'datetime.date'>
```

It creates a date object corresponding to the given value.

Time

```
>>> print pygoldilocks.Time(11,23,23), type(pygoldilocks.Time(11,23,23))
11:23:23 <type 'datetime.time'>
```

It creates a time object corresponding to the given value.

Timestamp

```
>>> print pygoldilocks.Timestamp(1984,11,23,11,23,23),
type(pygoldilocks.Timestamp(1984,11,23,11,23,23))
1984-11-23 11:23:23 <type 'datetime.datetime'>
```

It creates a datetime object corresponding to the given value.

DATETIME

```
>>> print pygoldilocks.DATETIME(1984,11,23,11,23,23),
type(pygoldilocks.DATETIME(1984,11,23,11,23,23))
1984-11-23 11:23:23 <type 'datetime.datetime'>
```

It creates a datetime object corresponding to the given value. It is as same as **Timestamp**.

Binary

```
>>> print pygoldilocks.Binary('binary'), type(pygoldilocks.Binary('binary'))
binary <type 'bytearray'>
```

It creates a bytearray object corresponding to the given value. It is as same as **BINARY**.

BINARY

```
>>> print pygoldilocks.BINARY('binary'), type(pygoldilocks.BINARY('binary'))
binary <type 'bytearray'>
```

It creates a bytearray object corresponding to the given value.

STRING

```
>>> print pygoldilocks.STRING('str'), type(pygoldilocks.STRING('str'))
str <type 'str'>
```

It creates an str object corresponding to the given value.

NUMBER

```
>>> print pygoldilocks.NUMBER(100.001), type(pygoldilocks.NUMBER(100.001))
100.001 <type 'float'>
```

It creates a float object corresponding to the given value.

ROWID

```
>>> print pygoldilocks.ROWID('AA'), type(pygoldilocks.ROWID('AA'))
AA <type 'str'>
```

It is used to describe the row ID column of the database, and returns an str object.

TimeFromTicks

```
>>> print pygoldilocks.TimeFromTicks( 10 )
09:00:10
```

It returns a datetime.time object which is set as an argument value.

DateFromTicks

```
>>> print pygoldilocks.DateFromTicks( 360000 )
1970-01-05
```

It returns a datetime.date object which is set as an argument value.

TimestampFromTicks

```
>>> print pygoldilocks.DateFromTicks( 360000 )
1970-01-05
```

It returns a datetime.timestamp object which is set as an argument value.

setDecimalSeparator

It sets the decimal point delimiter in NUMERIC type obtained from the database. The default value uses a period (.).

getDecimalSeparator

It obtains the set decimal point delimiter in NUMERIC type.

Connection

It is an object managing the connection with the database, and it is created with connect() function of py goldilocks module.

Properties

- autocommit
  - It can set the autocommit mode of the connection.
- searchescape
  - It obtains an escape character of ODBC. pygoldilocks uses '/'.
- timeout
  - It sets SQL_ATTR_QUERY_TIMEOUT by using SQLSetConnectAttr function.

함수

- cursor()
  - It returns a new cursor object.
- commit()
  - It commits the executed SQL statement.
- rollback()
  - It rolls back the executed SQL statement.
- close()
  - It closes the connection. If the autocommit is false, then the SQL statement which was not comm itted is rolled back.
- getinfo(info)
  - It can obtain the connection properties by using SQLGetInfo function of ODBC. For more informa tion, refer to **SQLGetInfo**.

```
dns_name = cnxn.getinfo( pygoldilocks.SQL_DATA_SOURCE_NAME )
```

- execute(sql, [*params])
  - It creates a new cursor object, and executes execute function of this object, then returns a cursor object.

```
cursor = cnxn.execute( "SELECT COUNT(*) FROM EMP" )
```

For more information, refer to Cursor.execute() function. This function does not exist in Python API, but i t is provided for the convenience. Whenever this function is called, a cursor object is allocated, so it is not recommended to use it when it is required to execute one or more SQL statements.

- set_attr(attr_id, value)

- The connection properties can be set by executing SQLSetConnectAttr function.
- The following is an example of controlling the transaction isolation level of the database by using set_attr function.

```
connection.set_attr( pygoldilocks.SQL_ATTR_TXN_ISOLATION, pygoldilocks.SQL_TXN_SERIALIZABLE )
```

Cursor

Generally, a cursor object means the database cursor which is used to manage the fetch operation. The database cursor is mapped to ODBC statement handle (HSTMT). The cursor objects which is created by the same connection are not separated. In other words, all updates executed by a cursor to the database are also applied to other cursors.

Cursor does not manage the database transaction, but the connection commits or rolls back the transaction.

Properties

Description

It is the read-only property, and it includes the contents for each column which was returned by SELECT statement executed last with tuple type. Each tuple includes the followings.

1. Column name (or alias)
2. Type code
3. Display size
4. Internal size
5. Precision
6. Scale
7. Nullable

When SELECT statement is not called, then the description is none.

rowcount

It is the number of rows which were updated by SQL statement which was executed last.

arraysize

It is the number of rows which can be fetched per one time by using **fetchmany([size = cursor.arraysize])** function. The default value 1.

connection

It is the read-only property, and it indicates the connection object which created the corresponding curso r object.

fast_executemany

If it is set to true, then makes the parameters in array and executes them at once when executing **execute many(sql, [*params])** function. If it is set to false, it separately executes each parameter.

Function

execute(sql, [*params])

It executes SQL statement through SQLPrepare and SQLExecute functions, then returns a cursor which ca lled this function.
The parameter option can be used as follows.

```
cursor.execute( "SELECT A FROM TEST WHERE B=? AND C=?", x, y )
```

```
cursor.execute( "SELECT A FROM TEST WHERE B=? AND C=?", (x, y) )
```

executemany(sql, [*params])

It executes SQL statement for each parameter and returns *none*. Parameter *params* should be a sequence type of a sequence or a sequence generator.

```
params = [ ( 1, 'A' ), ( 2, 'B' ) ]
cursor.executemany("INSERT INTO TEST( C1, C2 ) VALUES ( ?, ? )", params)
```

SQL statement is executed twice in the example above. In other words, it is separately executed for (1, 'A') and (2, 'B') each. The operation of executemany depends on whether fast_executemany of a cursor object is set to true or false.

The example above is as same as follows.

```
params = [ ( 1, 'A' ), ( 2, 'B' ) ]
for p in params:
    cursor.execute( "INSERT INTO TEST( C1, C2 ) VALUES ( ?, ? )", p )
```

If fast_executemany is set to true, executemany processes the operation with only a single execute. For th at, data in the same index location in items of parameter *params* should be the same data type.

```
params = [ ( 1, 'A' ), ( '2', 'B' ) ]
cursor.executemany("INSERT INTO TEST( C1, C2 ) VALUES ( ?, ? )", params)
```

In the example above, the data type of the first item among two items of parameter params is different. L ikewise, the data type on the same index location between items are different, then executemany does n ot process SQL statement at once, but separately processes it.

If the autocommit of a connection object is true, then SQL statement is processed being splited and each SQL statement is separately committed. If an error occurs while sequentailly processing records, then only some records are committed to the database and the operation is completed leaving some records are no t committed. Therefore, it is recommended to set autocommit to false to check if all records are committ ed to the database when using executemany().

fetchone()

It returns the next row of the query. If the next data does not exist, it is *none*.

fetchall()

It returns all rows left in the query. Be cautious when using it because it reads all rows to the memory.

fetchmany([size = cursor.arraysize])

It returns rows which were left as many as size or cursor.arraysize. The next data returns an empty sequen ce data. The default value of cursor.arraysize is 1.

commit()

It commits SQL statement. It is a function executed by a connection object which created a cursor object, and it is applied to all cursor which were created in the same connection object. It is as same as commit o f the connection object.

rollback()

It rolls back SQL statement. It is a function executed by a connection object which created a cursor object, and it is applied to all cursor which were created in the same connection object. It is as same as rollback o f the connection object.

skip(count)

It passes through the record through SQLFetchScroll and SQL_FETCH_NEXT as many times as it is set in c ount.

nextset()

It returns false because GOLDILOCKS ODBC does not support SQLMoreResults.

close()

It closes a cursor object.

setinputsizes(size_list)

It is an optional function, and receives sequence type as a parameter. It sets INPUT parameter size of SQL BindParameter.

setoutputsize(size)

It is an optional function, and is used for a purpose which is different from that of DB API, and it allocates the buffer size of OUTPUT parameter.

callproc(procname [, params])

It calls the storage procedure corresponding to procname. The parameter should be a sequence type, and it includes the output parameter. However, data located in the output parameter when inputting is mean ingless. callproc function updates data corresponding to INOUT, OUT of the input parameter data, and re turns it in sequence type.

```
create_proc = """CREATE OR REPLACE PROCEDURE PROC1( A1 INTEGER, A2 OUT CHAR(10) )
IS
  V1 CHAR(10);
BEGIN
  SELECT T1.I1
    INTO V1
    FROM T1
    WHERE T1.I1 >= A1 AND T1.I1 <= A1;
  A2 := V1;
END;\
"""
cursor.execute( create_proc )
result = cursor.callproc( 'PROC1', ( 1, 0 ) )
```

callfunc(funcname [, params])

It calls a function corresponding to funcname. callfunc() returns the function data.

```
create_func = """
CREATE OR REPLACE FUNCTION FUNC1( A1 INTEGER, A2 INTEGER )
  RETURN INTEGER
  IS
    V1 INTEGER;
  BEGIN
    SELECT COUNT(*)
      INTO V1
      FROM T1
      WHERE T1.I1 >= A1 AND T1.I1 <= A2;
    RETURN V1;
  END;\
"""
cursor.execute( create_func )
cursor.commit()
result = cussr.callfunc( 'FUNC1', ( 1,  4) )
```

tables(table=None, catalog=None, schema=None, tableType=None)

It returns the table information of the database which satisfies the given condition. The character '_' and '%' are translated as a wild card. Each row has the following column information. For more information, refer to **SQLTables**.

1. table_cat: It is the name of the catalog.
2. table_schem: It is the name of the schema.
3. table_name: It is the name of the table.
4. table_type: 'TABLE', 'VIEW', 'SYSTEM TABLE', 'GLOBAL TEMPORARY', 'LOCAL TEMPORARY', 'IMMUTABLE TABLE', 'ALIAS', 'SYNONYM' or a specific type name can be a table type.
5. remarks: It is a description of a table.

```
print cursor.tables( table= 'TEST' ).fetchone()
#print table name
for row in cursor.tables():
 print row.table_name
```

If a parameter is empty, information of all table of which a user has a privilege is returned.

columns(table=None, catalog=None, schema=None, tableType=None)

It obtains the column information of the specified table through **SQLColumns** function. Each row includes the following column information.

1. table_cat
2. table_schem
3. table_name
4. column_name
5. data_type
6. type_name
7. column_size
8. buffer_length
9. decimal_digits
10. num_prec_radix
11. nullable
12. remarks
13. column_def
14. sql_data_type
15. sql_datetime_sub
16. char_octet_length
17. ordinal_position
18. is_nullable: SQL_NULLABLE, SQL_NO_NULLS or SQL_NULLS_UNKNOWN.

```
#print column name of table TEST
for r in cursor.columns( table = 'TEST' ):
    print r.column_name
```

statistics(table, catalog=None, schema=None, unique=False, quick=True)

It obtains the information about the specified table through **SQLStatistics** function.
If unique is true, it returns an unique index, and if it is false, it returns all indexes.
If quick is true, CARDINALYTIY and PAGES are returned only when it is instantly available, otherwise, NUL L is returned to the corresponding column.

1. table_cat
2. table_schem
3. table_name
4. non_unique
5. index_qualifier
6. index_name
7. type
8. ordinal_position
9. column_name
10. asc_or_desc
11. cardinality
12. pages

13. filter_condition


A wildcard character is not allowed.

rowIdColumns(table, catalog=None, schema=None, nullable=True)

It returns the result set of columns which uniquely identifies a row by executing **SQLSpecialColumns** with SQL_BEST_ROWID. Each row includes the following column information.

1. scope: SQL_SCOPE_CURROW, SQL_SCOPE_TRANSACTION, or SQL_SCOPE_SESSION
2. column_name
3. data_type: SQL type constant of ODBC
4. type_name
5. column_size
6. buffer_length
7. decimal_digits
8. pseudo_column: SQL_PC_UNKNOWN, SQL_PC_NOT_PSEUDO or SQL_PC_PSEUDO

rowVerColumns(table, catalog=None, schema=None, nullable=True)

It returns the result set of columns which are automatically updated when a row is updated by executing **SQLSpecialColumns** with SQL_ROWVER. Each row includes the following column information.

1. scope: SQL_SCOPE_CURROW, SQL_SCOPE_TRANSACTION, or SQL_SCOPE_SESSION
2. column_name
3. data_type: SQL type constant of ODBC
4. type_name
5. column_size
6. buffer_length
7. decimal_digits
8. pseudo_column: SQL_PC_UNKNOWN, SQL_PC_NOT_PSEUDO, or SQL_PC_PSEUDO

primaryKeys(table, catalog=None, schema=None)

It returns the result set of columns which configures major keys of a table by executing **SQLPrimaryKeys** f unction. Each row includes the following column information.

1. table_cat
2. table_schem
3. table_name
4. column_name
5. key_seq
6. pk_name

foreignKeys(table=None, catalog=None, schema=None, foreignTable=None, foreignCatalog=None, foreignSchema=None)

It creates the result set of column names of a specified table, or the result set of column names which are foreign keys of another table referring to the basic key of the specified table, by executing **SQLForeignKeys** function. Each row includes the following column information.

1. pktable_cat
2. pktable_schem
3. pktable_name
4. pkcolumn_name
5. fktable_cat
6. fktable_schem
7. fktable_name
8. fkcolumn_name
9. key_seq
10. update_rule
11. delete_rule
12. fk_name
13. pk_name
14. deferrability

procedures(procedure=None, catalog=None, schema=None)

It creates the result set of the information about the procedure by executing **SQLProcedures**. Each row includes the following column information.

1. procedure_cat
2. procedure_schem
3. procedure_name
4. num_input_params
5. num_output_params
6. num_result_sets
7. remarks
8. procedure_type

getTypeInfo(sqlType=None)

It creates the result set of the information about the specified data type or about all data types which are supported by GOLDILOCKS ODBC, by executing **SQLGetTypeInfo** function. Each row includes the following column information.

1. type_name

2. data_type
3. column_size
4. literal_prefix
5. literal_suffix
6. create_params
7. nullable
8. case_sensitive
9. searchable
10. unsigned_attribute
11. fixed_prec_scale
12. auto_unique_value
13. local_type_name
14. minimum_scale
15. maximum_scale
16. sql_data_type
17. sql_datetime_sub
18. num_prec_radix
19. interval_precision

Row

A row object is returned with fetch function of a cursor object. It is processed as a tuple type as described in DB API.

```
row = cursor.fetchone()
for column in row:
    print column
```

The following features are added to pygoldilocks.

- It can access the data by using a column name.
- It can access the value of cursor.description through a row even after a cursor object is closed.
- It can update the value of a row.

Accessing to a row by using a column name is not only convenient but it also improves the readability. Ho wever, if a column name includes Python reserved name or a whitespace, then it be accessed only throug h row.__getattribute__().

```
cursor.execute( "select c1 from test")
print cursor.description
```

```
row = cursor.fetchone()
print row.C1
```

```
(('C1', <type 'str'>, 10, 10, 10, 0, True),)
test
```

Basically, the identifier of GOLDILOCKS database is uppercase. However, sometimes it is required to be specified in lowercase, so be cautious of using uppercase or lowercase when accessing to a row by using a column name.

Properties

- cursor_description

It is the copy of property description of a cursor object which created the corresponding row. For more information, refer to **Cursor.description**.

35.3 Exception

Python exceptions occur by pygoldilocks when GOLDILOCKS ODBC detects an error. The exception classes are as follows, which are as same as **Python DB API**.

- Error
  - DatabaseError
    - DataError
    - OperationalError
    - IntegrityError
    - InternalError
    - ProgrammingError
    - NotSupportedError

If an error occurs, generally, the exception is processed based on SQLSTATE value provided by the database.

SQLSTATE	Exception
0A000	NotSupportedError
01002	OperationalError
08001	OperationalError
08003	OperationalError
08004	OperationalError
08007	OperationalError
08S01	OperationalError
28000	InterfaceError
40002	IntegrityError
22***	DataError
23***	IntegrityError
24***	ProgrammingError
25***	ProgrammingError
42***	ProgrammingError

35.4 Data Type

Transferring Python Parameter to GOLDILOCKS

The data is converted as follows when transferring Python parameter to GOLDILOCKS ODBC.

Table 35-1 Python 3

Python datatype	Description	ODBC datatype
None	-	SQL_VARCHAR
str	UTF-8	SQL_VARCHAR or SQL_LONGVARCHAR
bytes, bytearray	binary	SQL_VARBINARY or SQL_LONGVARBINARY
bool	bit	SQL_BIT
datetime.date	date	SQL_TYPE_DATE
datetime.time	time	SQL_TYPE_TIME
datetime.datetime	timestamp	SQL_TYPE_TIMESTAMP
int	integer	SQL_BIGINT
float	floating point	SQL_DOUBLE
decimal	numeric	SQL_NUMERIC

Table 35-2 Python 2

Python datatype	Description	ODBC datatype
None		SQL_VARCHAR
str	UTF-8	SQL_VARCHAR or SQL_LONGVARCHAR
bytearray	binary	SQL_VARBINARY or SQL_LONGVARBINARY
buffer	binary	SQL_VARBINARY or SQL_LONGVARBINARY
bool	bit	SQL_BIT
datetime.date	date	SQL_TYPE_DATE
datetime.time	time	SQL_TYPE_TIME
datetime.datetime	timestamp	SQL_TYPE_TIMESTAMP
int	integer	32 bit: SQL_INTEGER, 64 bit: SQL_BIGINT
long	bigint	SQL_BIGINT
float	floating point	SQL_DOUBLE
decimal	numeric	SQL_NUMERIC

SQL Value Received from GOLDILOCKS

The data is converted as follows when transferring the data of GOLDILOCKS database to Python.

Table 35-3 Python 3

ODBC datatype	Description	Python datatype
any	NULL	None
SQL_CHAR, SQL_VARCHAR, SQL_LONGVARCHAR	text	text
SQL_BINARY_SQL_VARBINARY, SQL_LONGVARBINARY	binary	bytes
SQL_NUMERIC	decimal, numeric	decimal.Decimal
SQL_BOOLEAN	bit, bool	bool
SQL_SMALLINT, SQL_INTEGER	integers	int
SQL_BIGINT	long	long
SQL_REAL, SQL_FLOAT, SQL_DOUBLE	floating point	float
SQL_TYPE_TIME	time	datetime.time
SQL_TYPE_DATE	date	datetime.date
SQL_TYPE_TIMESTAMP	timestamp	datetime.timestamp
SQL_TYPE_TIME_WITH_TIMEZONE	time with timezone	text
SQL_TYPE_TIMESTAMP_WITH_TIMEZONE	timestamp with timezone	text
SQL_C_INTERVAL_***	interval	text

Table 35-4 Python 2

ODBC datatype	Description	Python datatype
any	NULL	None
SQL_CHAR, SQL_VARCHAR, SQL_LONGVARCHAR	text	text
SQL_BINARY_SQL_VARBINARY, SQL_LONGVARBINARY	binary	bytes
SQL_NUMERIC	decimal, numeric	decimal.Decimal
SQL_BOOLEAN	bit, bool	bool
SQL_SMALLINT, SQL_INTEGER	integers	int
SQL_BIGINT	long	long
SQL_REAL, SQL_FLOAT, SQL_DOUBLE	floating point	float
SQL_TYPE_TIME	time	datetime.time
SQL_TYPE_DATE	date	datetime.date
SQL_TYPE_TIMESTAMP	timestamp	datetime.timestamp
SQL_TYPE_TIME_WITH_TIMEZONE	time with timezone	text
SQL_TYPE_TIMESTAMP_WITH_TIMEZONE	timestamp with timezone	text
SQL_C_INTERVAL_***	interval	text

The text of Python data type is converted to unicode in Python 3. It is converted to the unicode or a string according to the character set of the database in Python 2.

Table 35-5 Python 2 text

DB character set	Python type
UTF-8	str
SQL_ASCII	str
UHC	unicode
GB18030	unicode

36.

Ruby

36.1 Overview

GOLDILOCKS ruby is a driver which enables to use GOLDILOCKS database in an application created with r uby, and it is provided as a RubyGem package.
GOLDILOCKS ruby driver is created based on GOLDILOCKS CLI driver.

36.2 Installation

Requirement

The client package of GOLDILOCKS should be installed in the same system as that of Ruby, and $GOLDIL OCKS_HOME environment variable should be set.

Installing/ Removing

GOLDILOCKS ruby driver can be installed/ removed by using gem.

Removing

Remove the installed GOLDILOCKS ruby driver.

```
% sudo -E gem uninstall ruby-goldilocks
```

Installating

Create the package file when GOLDILOCKS ruby driver package file does not exist.

```
% gem build goldilocks.gemspec
  Successfully built RubyGem
  Name: ruby-goldilocks
  Version: 3.2.0
  File: ruby-goldilocks-3.2.0.gem
```

Install the package file by using gem when GOLDILOCKS ruby driver package file exists.

```
% sudo -E gem install ruby-goldilocks-3.2.0.gem
Building native extensions.  This could take a while...
Successfully installed ruby-goldilocks-3.2.0
Parsing documentation for ruby-goldilocks-3.2.0
Installing ri documentation for ruby-goldilocks-3.2.0
Done installing documentation for ruby-goldilocks after 0 seconds
1 gem installed
```

36.3 Examples

Connecting

Connecting by Using DSN

```
require 'goldilocks'
$c = GOLDILOCKS.connect("GOLDILOCKS")
```

Connecting by Using DSN, UID, PWD

```
require 'goldilocks'
$c = GOLDILOCKS.connect("GOLDILOCKS", "test", "test")
```

Connecting by Using Connecting String

```
$c = GOLDILOCKS::Database.new.drvconnect("DSN=GOLDILOCKS")
```

```
$c = GOLDILOCKS::Database.new.drvconnect("HOST=192.168.0.1;PORT=22581;UID=test;PWD=test")
```

Creating Table

```
$c.run("create table test (id int not null, str varchar(32) not null)")
```

Inserting Data

```
$q = $c.run("insert into test (id, str) values (1, 'foo')")
$q.run("insert into test (id, str) values (2, 'bar')")
$p = $c.proc("insert into test (id, str) values (?, ?)") {}
$p.call(3, "FOO")
$p[4, "BAR"]
```

Retrieving Data

```
$q = $c.prepare("select id,str from test order by id")
if $q.column(0).name.upcase != "ID" then raise "fetch failed" end
if $q.column(1).name.upcase != "STR" then raise "fetch failed" end
$q.execute
if $q.fetch != [1, "foo"] then raise "fetch: failed" end
if $q.fetch != [2, "bar"] then raise "fetch: failed" end
if $q.fetch != [3, "FOO"] then raise "fetch: failed" end
if $q.fetch != [4, "BAR"] then raise "fetch: failed" end
if $q.fetch != nil then raise "fetch: failed" end
$q.close
if $q.execute.entries != [[1, "foo"], [2, "bar"], [3, "FOO"], [4, "BAR"]] then
  raise "fetch: failed"
end
$q.close
if $q.execute.fetch_all != [[1, "foo"], [2, "bar"], [3, "FOO"], [4, "BAR"]] then
  raise "fetch: failed"
end
$q.close
$q.execute
if $q.fetch_many(2) != [[1, "foo"], [2, "bar"]] then raise "fetch: failed" end
if $q.fetch_many(3) != [[3, "FOO"], [4, "BAR"]] then raise "fetch: failed" end
if $q.fetch_many(99) != nil then raise "fetch: failed" end
```

```
$q.close
a = []
$q.execute {|r| a=r.entries}
if a.size != 4 then raise "fetch: failed" end
$q.close
a = []
$q.execute.each {|r| a.push(r)}
if a.size != 4 then raise "fetch: failed" end
$q.close
a = []
$q.execute.each_hash {|r| a.push(r)}
if a.size != 4 then raise "fetch: failed" end
$q.close
a = []
$q.execute.each_hash(true) {|r| a.push(r)}
if a.size != 4 then raise "fetch: failed" end
$q.close
a = []
$q.execute.each_hash(:key=>:Symbol) {|r| a.push(r)}
if a.size != 4 then raise "fetch: failed" end
$q.close
a = []
$q.execute.each_hash(:key=>:Symbol,:table_names=>true) {|r| a.push(r)}
if a.size != 4 then raise "fetch: failed" end
$q.close
a = []
$q.execute.each_hash(:key=>:String,:table_names=>false) {|r| a.push(r)}
if a.size != 4 then raise "fetch: failed" end
$q.close
a = []
$q.execute.each_hash(:key=>:Fixnum,:table_names=>false) {|r| a.push(r)}
if a.size != 4 then raise "fetch: failed" end
$q.close
```

Updating Data

```
$q = $c.run("update test set id=0, str='hoge'")
if $q.nrows != 4 then
  $stderr.print "update row count: expected 4, got ", $q.nrows, "\n"
```

```
end
```

Deleting Data

```
$count = $c.do("delete from test where 1 = 1")
if $count != 4
  $stderr.print "delete row count: expected 4, got ", $count, "\n"
end
$count = $c.do("delete from test where 1 = 1")
if $count != 0
  $stderr.print "delete row count: expected 0, got ", $count, "\n"
end
```

Arranging All Created Statement

```
$c.drop_all
```

Dropping Table

```
$c.do("drop table test")
```

Disconnecting

```
$c.disconnect
```

36.4 Examples of ActiveRecord

The following table is created in advance as an example.

```
CREATE TABLE members
(
  id         INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  name       VARCHAR(255),
  email      VARCHAR(255),
  created_at TIMESTAMP,
  updated_at TIMESTAMP
);
```

Connecting

```
require 'rubygems'
require 'active_record'
require 'goldilocks'
ActiveRecord::Base.establish_connection(
    :adapter => "goldilocks",
    :host => "192.168.0.1",
    :port => "22581",
    :uid => "TEST",
    :pwd => "test",
    :option => "" )
```

The property values for the connection are as follows.

Property	Description
:adapter	goldilocks
:dsn	ODBC data source name (DSN)
:host	IP address of a server
:port	PORT number of a server
:uid	User ID
:pwd	User password
:opt	Connection string

Inserting Data

```
class Members < ActiveRecord::Base
end
```

```
Members.create(:name => "David",  :email => "David@mail.com")
Members.create(:name => "Olivia", :email => "Olivia@mail.com")
Members.create(:name => "Amelia", :email => "Amelia@mail.com")
Members.create(:name => "Hazel",  :email => "Hazel@mail.com")
Members.create(:name => "Lily",   :email => "Lily@mail.com")
```

Retrieving Data

```
require 'pp'
puts "\nfind(1)"
puts '-'*80
David = Members.find(1)
pp(David)
puts '-'*80
puts "\nwhere(:name => \"Amelia\")"
puts '-'*80
Amelia = Members.where(:name => "Amelia")
pp(Amelia)
puts '-'*80
puts "\nfind_each"
puts '-'*80
Members.find_each do |member|
  pp(member)
end
puts '-'*80
```

Updating Data

```
puts "\nupdate Lily -> Harry"
puts '-'*80
Members.transaction do
 Lily = Members.where(:name => 'Lily')[0]
 Lily.name = 'Harry'
 Lily.save
end
puts "\nfind_each"
Members.find_each do |member|
```

```
 pp(member)
end
puts '-'*80
```

Deleting Data

```
puts "\ndelete Harry"
puts '-'*80
Members.transaction do
 Harry = Members.where(:name => 'Harry')[0]
 Harry.destroy
end
puts "\nfind_each"
Members.find_each do |member|
 pp(member)
end
puts '-'*80
```

37.

Hibernate

37.1 Overview

Hibernate is a Object-Replation Mapping (ORM) framework of which Java Persistent API (JPA) model is a pplied to the database. It maps the relationship between DB table and Java object so that it helps the per sistent logic processing. In other words, when using hibernate, it is easy to access and control DB without using SQL.

37.2 Interworking with Hibernate

Downloading Hibernate

Hibernate can be downloaded at **https://sourceforge.net/projects/hibernate/**. When the downloaded file is decompressed, many jar files are found in lib directory. These files are used to interwork with hibernate.

Hibernate of GOLDILOCKS is created based on Hibernate version 5.2.

Interworking with GoldilocksDialect Class

GOLDILOCKS provides GoldilocksDialect.java file which is appropriate to Hibernate ORM version. When u sing Goldilocks.java 5 version file in Hibernate ORM 4 version, then a compile error occurs. Therefore, use GoldilocksDialect.java file which is appropriate for Hibernate version.

Especially, Hibernate ORM 4 version provides each different api according to the patch version, so GOLDI LOCKS provides GoldilocksDialect.java according to it. For example, Goldilocks.java 4.0.0 version is used i n Hibernate ORM 4.1.4 version, and GoldilocksDialect.java 4.1.5 version is used in Hibernate ORM 4.1.9 version.

GOLDILOCKS provides GoldilocksDialect.java in the directory under $GOLDILOCKS_HOME/app_dev/Hiber nate/.

Porting to Hibernate jar File

It should be ported to the downloaded Hibernate jar file to interwork with Hibernate. GoldilocksDialect.class file and GoldilocksDialect$1.class file can be included in Hibernate jar file as follows.

1. Create a class file by compiling GoldilocksDialect.java file using javac. (The class file is differently created per each version.)
2. Decompress Hibernate jar file.
3. Move GoldilocksDialect.class file and GoldilocksDialect$1.class file to the decomprressed org/hibernate/dialect directory.
4. Compress it again with Hibernate jar file.
5. Add the newly created jar file to classpath.

```
$ javac -cp hibernate-cor-5.2.11.Final.jar GoldilocksDialect.java
$ jar -xvf hibernate-core-5.2.11.Final.jar
$ pwd
hibernate-release-5.2.11.Final/lib/required/
$ mv GoldilocksDialect*.class org/hibernate/dialect/
$ jar -cvf hibernate.5.2.11.jar  META-INF/MANIFEST.MF .
```

Porting to Program Source

Interworking is available by directly using the source file without porting the class file to jar file. Create org.hibernate.dialect package in the user project, then insert GoldilocksDialect.java file into this package.

37.3 Examples

Configuration

hibernate-configuration XML file and hibernate-mapping XML file are required to interqork Hibernate an d DB.

Hibernate Configuration File

Configuration XML file sets the connect information to access the dialect class and DB, and sets Hibernat e-mapping files.
For more information, refer to **Configuration**.

The following is an example of hibernate.conf.xml file.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <property
name="hibernate.connection.driver_class">sunje.goldilocks.jdbc.GoldilocksDriver</property>
        <property
name="hibernate.connection.url">jdbc:goldilocks://192.168.0.21:22581/test</property>
        <property name="hibernate.connection.username">test</property>
        <property name="hibernate.connection.password">test</property>
        <property name="hibernate.dialect">org.hibernate.dialect.GoldilocksDialect</property>
        <property name="show_sql">true</property>
        <property name="format_sql">true</property>
        <property name="hbm2ddl.auto"> create </property>
        <mapping resource="SampleTable.mapping.xml" />
    </session-factory>
</hibernate-configuration>
```

Set the DB connecting information on <session-factory> tag, and specifies the dialect class name of DB to connect to hibernate.dialect property. Also, specify hibernate-mapping files by using <mapping-resource> tag.

Hibernate Mapping File

Mapping XML is a configuration file including mapping information about DB table and Java object.
For more information, refer to **Mapping**.

The following is an example of SampleTable.mapping.xml file which was used in the example above.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
    <class name="SampleTable" table="SAMPLETABLE">
        <id column="ID" name="id" type="long">
            <generator class="increment" />
        </id>
        <property column="STUDENT_NAME" name="name" type="string" />
        <property column="BIGINT_VALUE" name="bigintValue" type="long" />
        <property column="INT_VALUE" name="intValue" type="int" />
        <property column="SMALL_VALUE" name="smallValue" type="short" />
        <property column="FLOAT_VALUE" name="floatValue" type="float" />
        <property column="DOUBLE_VALUE" name="doubleValue" type="double" />
        <property column="NUMERIC_VALUE" name="numericValue" type="big_integer" precision="10"
scale="5" />
        <property column="DATE_VALUE" name="dateValue" type="date" />
        <property column="TIME_VALUE" name="timeValue" type="time" />
        <property column="TIMESTAMP_VALUE" name="timestampValue" type="timestamp" />
        <property column="VARBINARY_VALUE" name="binaryValue" type="byte[]" />
    </class>
</hibernate-mapping>
```

The mapping information about Java class and DB table are described on <class> tag. Data type used in a table column and Java are specified on <property> tag.

Examples of Application

Java class to be mapped to DB table should be described. The following is an example of SampleTable.jav
a which is described above.

```
import java.math.BigInteger;
import java.sql.Date;
import java.sql.Time;
import java.sql.Timestamp;
public class SampleTable {
    private long id;
    private String     name;
    private long       bigintValue;
    private int        intValue;
    private short      smallValue;
    private float      floatValue;
    private double     doubleValue;
    private BigInteger numericValue;
    private Date       dateValue;
    private Time       timeValue;
    private Timestamp  timestampValue;
    private byte[]     binaryValue;
    private String     longvarcharValue;
    private byte[]     longvarbinaryValue;

    public SampleTable() {
        this.name = null;
    }

    public SampleTable( String name ) {
        this.name = name;
    }
    public long getId() {
        return id;
    }

    public void setId( long id ) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName( String name ) {
        this.name = name;
```

```
    }
....
```

get/ set function of a variable corresponding to a column of SampleTable table should be described.

The data is processed in SamleTable table as follows.

1. It obtains org.hibernate.SessionFactory object through a configuraion file.
2. It obtains org.hibernate.Session object through SessionFactory object.
3. It processes the desired operation by calling a method of a session object.

The following is an example of inserting, updating, retrieving, deleting data in SampleTable table of DB. It obtains SessionFactory object through org.hibernate.Configuration class.

```
public class HibernateSample
{
    private static SessionFactory mFactory;
    public static void main( String[] args ) {
        try {
            mFactory = new Configuration().configure( "hibernate.conf.xml"
).buildSessionFactory();
        } catch( Throwable ex ) {
            System.err.println( "Failed to create sessionFactory object." + ex);
            throw new ExceptionInInitializerError(ex);
        }
        HibernateSample sHibernateSample = new HibernateSample();
```

- Insert 3 records

```
Long sRecord1 = sHibernateSample.addSample( "RECORD_1" );
Long sRecord2 = sHibernateSample.addSample( "RECORD_2" );
Long sRecord3 = sHibernateSample.addSample( "RECORD_3" );
```

- Update record 1

```
sHibernateSample.updateSample( sRecord1, ....
```

- Fetch table

```
sHibernateSample.fetchSample( 0, 3 );
```

- Delete record 2

```
sHibernateSample.deleteSample( sRecord2 );
sHibernateSample.closeFactory();
    }
public void closeFactory() {
        mFactory.close();
    }
....
```

The following is an example of a method inserting a record to HibernateSample class. First, it obtains a se ssion object through SessionFactory class, then obtains org.hibernate.Transaction object. It can be commi tted or rolled back after starting a transaction by using a transaction class and inserting a record.

```
public Long addSample( String name ) {
    Session session = mFactory.openSession();
    Transaction tx = null;
    Long id = null;

    try{
```

- Begin transaction

```
tx = (Transaction) session.beginTransaction();
SampleTable sample = new SampleTable( name );
```

- Insert

```
id = (Long) session.save(sample);
```

- Commit transaction

```
tx.commit();

        System.out.println("add success");
    } catch( HibernateException e) {
        if( tx != null ) {
            tx.rollback();
        }
        System.out.println("add fail");
        e.printStackTrace();
    } finally {
        session.close();
    }
```

```
    return id;
}
```

The following is an example of a method retrieving SampleTable table. It obtains org.hibernate.Query obj ect through a session class. fetchSample method retrieves SampleTable from offset to count.

```
public void fetchSample( int offset, int count ) {
    Session session = mFactory.openSession();
    Query query = null;

    try{
        query = session.createQuery( "FROM SampleTable" );
        query.setFirstResult( offset );
        query.setMaxResults( count );
        query.setFetchSize( 20 );

        @SuppressWarnings( "unchecked" )
        List<SampleTable> samples = (List<SampleTable>) query.list();
        for( Iterator<SampleTable> iter = samples.iterator(); iter.hasNext(); ) {
            SampleTable sample = (SampleTable) iter.next();
            System.out.print( "id: " + sample.getId() );
           ... Ellipsis ...
        }
    } catch ( HibernateException e ) {
        e.printStackTrace();
    } finally {
        session.close();
    }
}
```

The following is an exampel of a method updating SampleTable table. After altering to a set method des cribed in SampleTable class, then it is updated by calling an update method of a session class.

```
public void updateSample( long id, ... ) {
    Session session = mFactory.openSession();
    Transaction tx = null;
    try {
        tx = session.beginTransaction();
        SampleTable sample = (SampleTable)session.get(SampleTable.class, id);
        sample.setName( name );
        ...
        session.update( sample );
```

```
        tx.commit();
        System.out.println( "update success" );
        } catch ( HibernateException e ) {
            if( tx != null ){
                tx.rollback();
            }
            System.out.println("update fail");
            e.printStackTrace();
        } finally {
    session.close();
    }
}
```

The following is an example of deleting a record. It can be deleted by using a delete method of a session class.

```
public void deleteSample( long id ) {
    Session session = mFactory.openSession();
    Transaction tx = null;

    try {
        tx = session.beginTransaction();
        SampleTable sample = (SampleTable) session.get( SampleTable.class, id );
        session.delete( sample );
        tx.commit();

        System.out.println( "delete success" );
    } catch ( HibernateException e ) {
        if( tx != null ) {
            tx.rollback();
        }
        System.out.println("delete fail");
        e.printStackTrace();
    } finally {
        session.close();
    }
}
```

Part VI.

Utility Manual

38.

gcreatedb

38.1 Overview of gcreatedb

Definition

gcreatedb is a database creation utility provided by GOLDILOCKS.

Feature

A new database is created by using DB_Name, comment, time zone, character set specified by a user.

Usage

```
gcreatedb [options]
```

Option

```
--cluster             cluster system (if not specified, stand-alone system)
--db_name             database name
--db_comment          database comment
--timezone            timezone ( {+/-}{TZH:TZM} )
--character_set       character set
                           SQL_ASCII
                           UTF8
                           UHC
                           GB18030
--char_length_units   char length units
                           OCTETS
                           CHARACTERS
--home                home directory
--member              local member name
--host                host address
--port                host port
--silent              suppersses the display of the result message
```

```
--help                print help message
```

Example

```
$ gcreatedb --db_name="goldilocks" --db-comment="goldilocks database" --timezone="+09:00"
--character_set="UTF8" --silent
```

38.2 Command Option

--cluster

It creates the database of a cluster environment.
If this option is not used, then it creates the database of a stand alone environment.

--db_name

It specifies the database name which a user wants to create.
The default value is "goldilocks".
The maximum length of database name is 128.

--db_comment

It specifies the database comment which a user wants to create.
The default value is "goldilocks database".
The maximum length of database comment is 1024.

--timezone

It is the time zone value of database.
The default value is the **TIMEZONE** value of the property.
The property is applied when database is created and it has a value in range of '-14:00' ~ '+14:00'.

--character_set

It is the database character set.
The default value is the **CHARACTER_SET** value of the property.

The property is applied when database is created, and its value is set to one of the followings.

- GB18030

- SQL_ASCII
- UHC
- UTF8

--char_length_units

It is the char length unit value which is used when defining the string column type such as CHAR, VARCHAR and omitting its char length unit as follows.

```
CREATE TABLE t1
(
    id   CHAR( 10 OCTETS ),          ❶ It refers to 10 bytes.
    name VARCHAR( 128 CHARACTERS ),  ❷ It refers to 128 characters.
    addr VARCHAR( 1024 )             ❸ char length unit is omitted.
```

The default value is the **CHAR_LENGTH_UNITS** value of the property.

The property is applied when database is created, and its value is set to one of the followings.
- OCTETS: It is the number of bytes.
- CHARACTERS: It is the number of characters.

The default value of char length unit is defined as CHARACTERS in the SQL standard and it is defined as follows in other DBMS.

- OCTETS is used in Oracle, DB2.
- CHARACTERS is used in MS-SQL, MySQL, PostgreSQL.

--home

It sets the home directory of the database.
The default value uses $GOLDILOCKS_HOME environment variable.

--member

It sets the name of a local database in a cluster environment.

--host

It sets the host address of a local database in a cluster environment.

--port

It sets the port number of a local database in a cluster environment.

--silent

Result messages are not displayed.

--help

Help messages are displayed.

```
$ gcreatedb --help
Usage
    gcreatedb [options]
Options:
    --db_name                database name
    --db_comment             database comment
    --timezone               timezone ( {+/-}{TZH:TZM} )
    --character_set          character set
                               SQL_ASCII
                               UTF8
                               UHC
                               GB18030
    --char_length_units      char length units
                               OCTETS
                               CHARACTERS
    --silent                 suppresses the display of the result message
    --help                   print help message
examples:
    gcreatedb --db_name="goldilocks" --db_comment="goldilocks database" --timezone="+09:00"
```

```
--character_set="UTF8" --char_length_units="OCTETS" --silent
```

39.

glsnr

39.1 Overview of glsnr

glsnr is a listener which GOLDILOCKS enables remote access through the network in the client/ server env ironment. glsnr should be run on the server for the network access to GOLDILOCKS.

glsnr is used as follows.

```
$ glsnr [options]
```

39.2 Command Options

The followings are cell prompt options for using glsnr.

--silent

Description

It does not display the message for execution.

Example

```
$ glsnr --start --silent
```

--start

Description

It starts running glsnr. If glsnr is already running, an error occurs.

Example

```
$ glsnr --start
Listener is started successfully.
```

--stop

Description

It stops the currently running glsnr.

Example

```
$ glsnr --stop
Listener is stopped.
```

--status

Description

It displays the message about glsnr status.

Example

```
$ glsnr --status
Listener is not running.
$ glsnr --start
Listener is started successfully.
$ glsnr --status
Listener process ID : 27880
Listener configuration file : /home/goldilocks/goldilocks_home/conf/goldilocks.listener.conf
Unix Domain Path : /tmp/unix-glsnr.22581
TCP Listen Host : 0.0.0.0, Port : 22581
default C/S mode : Dedicated
Connection Timeout(second) : 100
Listener is running.
```

--home

Description

It sets db home.

Example

```
$ glsnr --start --home Gliese/home/g1n1_home
Listener is started successfully.
$ glsnr --status
```

```
Listener process ID : 20777
Listener configuration file :
/home/goldilocks/Gliese/home/g1n1_home/conf/goldilocks.listener.conf
Unix Domain Path : /tmp/unix-glsnr.22581
TCP Listen Host : 0.0.0.0, Port : 22581
default C/S mode : Dedicated
Connection Timeout(second) : 100
Listener is running.
```

--help

Description

It displays the help message.

Example

```
$ glsnr --help

Usage:
  glsnr [options]

Options:

  --silent       don't print message
  --start        start listener
  --stop         stop listener
  --status       show listener status
  --help         show listner help messages
```

39.3 Listener Configuration

Configuration File and Environment Variables

glsnr uses the configuration file or environment variables for setting the configuration.

The environment variable is specified by using the name which 'GOLDILOCKS_' is added to the property name of the configuration as a prefix. For example, setting the LISTEN_PORT in configuration file is as same as specifying $GOLDILOCKS_LISTEN_PORT.

The contents of configuration file precedes the environment variable settings. (In other words, environment variables are applied only if the configuration file is not set.)

The environment file of glsnr is $GOLDILOCKS_DATA/conf/goldilocks.listener.conf. If the corresponding file is changed or the environment variables are set and glsnr is started to run, then the glsnr driving environment is modified.

If a user want to modify and apply the glsnr environment during running glsnr, a user should stop glsnr and change the content of configuration file or set the environment variable and then restart glsnr.

If a user stops the glsnr, a problem does not occur for the client which is already connected, but the problem occurs when being connected from a new client.

LISTEN_PORT

It is the port on which the glsnr waits for the connection.

Item	Description
Name	LISTEN_PORT
Description	It is the port on which the glsnr waits for a connection.
Data type	INT
Default value/ range	22581 / 1024 ~ 49151

Description

The client who wants a TCP connection should try to connect to the specified port.
The port is available from 1024 to 49151.

TCP_HOST

It is an IP address of NIC of which glsnr waits for the connection.

Item	Description
Name	TCP_HOST
Description	It is the IP address which the glsnr binds.
Data type	ip address
Default value	0.0.0.0

Description

The client who wants a TCP connection should access the IP address specified above. The IP address is us ed in IPv4 or IPv6 format.

BACKLOG

It is the number of clients which the glsnr can handle when multiple clients simultaneously access.

Item	Description
Name	BACKLOG
Description	It is the number of client of which glsnr waits for the connection.
Data type	INT
Default value/ range	1024 / 1 ~ 32768

Description

This setting value does not guarantee the number of concurrent connector of the client.

DEFAULT_CS_MODE

It sets the access mode when the access mode is not designated as dedicated or shared on the client.

Item	Description
Name	DEFAULT_CS_MODE
Description	It sets the default access mode.

Item	Description
Data type	String (dedicated | shared)
Default value	dedicated

Description

- The client/ server mode connected via glsnr supports two modes, which are dedicated and shared.
- Generally, the access mode is set as dedicated or shared on the client (It is .odbcini for ODBC), then it accesses. However, if it is not set on the client, the access mode is determined by setting DEFAULT_CS_MODE.

TCP_VALIDNODE_CHECKING

It sets whether to check the validity of the client attempting to access.

Item	Description
Name	TCP_VALIDNODE_CHECKING
Description	It sets whether to check the client validity.
Data type	String (NO | INVITED | EXCLUDED)
Default value	NO

Description

- If the value is set to *NO*, the client validity is not checked.
- If the value is set to *INVITED* and the file set in TCP_INVITED_FILE exists, then only the client having the IP address of the file set in TCP_INVITED_FILE is set to be valid.
- If the value is set to *EXCLUDED* and the file set in TCP_EXCLUDED_FILE exists, then only the client excluding the client who has IP address of the file set in TCP_EXCLUDED_FILE is set to be valid.

TCP_INVITED_FILE

It is used only when the value of TCP VALIDNODE_CHECKING is *INVITED*.

Item	Description
Name	TCP_INVITED_FILE
Description	It is the file with the valid user (IP address) list.
Data type	String

Item	Description
Default value	'goldilocks.invited.conf'

Description

If the set file exists, only the user (IP address) within the file is allowed to access.

TCP_EXCLUDED_FILE

It is used only when the value of TCP VALIDNODE_CHECKING is *EXCLUDED*.

Item	Description
Name	TCP_EXCLUDED_FILE
Description	It is the file with the invalid user (IP address) list.
Data type	String
Default value	'goldilocks.excluded.conf'

Description

If the set file exists, anyone except for the user (IP address) within the file is allowed to access.

TIMEOUT

It is the glsnr timeout value and its unit is second.

Item	Description
Name	TIMEOUT
Description	glsnr timeout
Data type	INT
Default value/ range	100 / (0 ~ 2147483647)

Description

If the response is too slow or there is not a response from the client when glsnr communicates with the cl ient, the connection is released by the specified timeout.

LISTENER_LOG_DIR

It sets the directory which stores the log to be display from the glsnr.

Item	Description
Name	LISTENER_LOG_DIR
Description	It sets the directory which stores the log of the glsnr.
Data type	String
Default value	'<GOLDILOCKS_DATA>/trc'

Description

<GOLDILOCKS_DATA> of the setting value is replaced with the value of the environment variable $GOLDILOCKS_DATA.

UDS_DIR

It sets the directory in which the Unix Domain Socket file used in glsnr is stored.

Item	Description
Name	UDS_DIR
Description	It sets the directory in which the Unix Domain Socket file used in glsnr is stored.
Data type	String
Default value/ range	'/tmp' / Maximum 60 byte

Description

The maximum length of the directory should be set within 60 bytes. The absolute path of the Unix Domain Socket file (directory name + file name) depends on OS, but usually it is around 100 bytes.

40.

gsql/gsqlnet (Interactive SQL Tool)

40.1 Overview of gsql

The followings are execution files.

Table 40-1 Execution files

Execution file name	Description
gsql	It is used in Direct Attach (D/A) environment.
gsqlnet	It is used in Client/ Server (C/S) environment.


All commands supported by gsql are equally supported by gsqlnet.

Definition

gsql is an interactive utility for processing SQL statements provided by GOLDILOCKS.

Using gsql, a user can query the results of the SELECT statement as well as execute SQL statements without any application.

gsql program executes the SQL statement, and queries a brief information of the objects such as tables and indexes through the gsql specific commands beginning with \, and it provides various features such as startup or shutdown of the server, controlling the output, and querying the execution plan of SQL statement.

Examples

This chapter describes the method of creating/dropping the table, manipulating data, performing a query by using gsql.

gsql is operated in an interactive mode as follows. When connecting normally, it waits for entering the SQL statement or gsql command with *gSQL>* prompt.

```
% gsql test test
Connected to GOLDILOCKS Database.
gSQL>
```

Create a table by executing **CREATE TABLE** statement and commit the transaction as follows.

```
gSQL> CREATE TABLE t1 ( id INTEGER, name VARCHAR(128) );
Table created.
gSQL> COMMIT;
Commit complete.
```

Insert the data into the created table by executing **INSERT INTO** statement, then commit the transaction a s follows. The second INSERT statement is an example of inserting two rows.

```
gSQL> INSERT INTO t1 VALUES ( 1, 'leekmo' );
1 row created.
```

- Two rows are inserted.

```
gSQL> INSERT INTO t1 VALUES ( 2, 'mkkim' ), ( 3, 'xcom73' );
2 rows created.
gSQL> COMMIT;
Commit complete.
```

The column addr is added to the table and data is updated by executing **UPDATE** statement as follows.

```
gSQL> ALTER TABLE t1 ADD COLUMN ( addr VARCHAR(1024) );
Table altered.
gSQL> UPDATE t1 SET addr = 'Seoul, Korea' WHERE id = 1;
1 row updated.
gSQL> UPDATE t1 SET addr = 'Inchon, Korea' WHERE id = 3;
1 row updated.
gSQL> COMMIT;
Commit complete.
```

The inserted data is queried by using **SELECT** query as follows. The query result consists of the header rep resenting column names and the information of each row in a single line. NULL is expressed in lowercase null.

```
gSQL> SELECT * FROM t1 ORDER BY 1;
ID NAME   ADDR
-- ------ -------------
 1 leekmo Seoul, Korea
 2 mkkim  null
 3 xcom73 Inchon, Korea
3 rows selected.
```

A host variable is declared, and the value is put in the host variable then a query using the host variable is

performed as follows.

```
gSQL> \var v_id INTEGER
gSQL> \exec :v_id := 1
gSQL> SELECT * FROM t1 WHERE id = :v_id;
ID NAME   ADDR
-- ------ ------------
 1 leekmo Seoul, Korea
1 row selected.
```

The SQL statement is prepared by using **\prepare sql** and the SELECT statement above is repeatedly perf ormed by using **\exec**, changing the value of host variable. ODBC and JDBC simulate prepare/ execute op eration in gsql interactive mode as follows.

```
gSQL> \var v_id INTEGER
gSQL> \prepare sql SELECT * FROM t1 WHERE id = :v_id;
SQL prepared.
gSQL> \exec :v_id := 2
gSQL> \exec
ID NAME  ADDR
-- ----- ----
 2 mkkim null
1 row selected.
gSQL> \exec :v_id := 3
gSQL> \exec
ID NAME   ADDR
-- ------ -------------
 3 xcom73 Inchon, Korea
1 row selected.
```

A scrollable cursor is declared in an embedded SQL, ODBC, JDBC, and the data retrieval using the cursor i s simulated via gsql as follows. The cursor cur1 declared by using **DECLARE cursor_name** is the scrollable KEYSET cursor and various fetch orientations may be used by using **FETCH cursor_name**.

```
gSQL> \var v_id INTEGER
gSQL> \var v_name VARCHAR(128)
gSQL> DECLARE cur1 KEYSET CURSOR FOR SELECT id, name FROM t1 ORDER BY id;
Cursor declared.
gSQL> OPEN cur1;
Cursor is open.
gSQL> FETCH cur1 INTO :v_id, :v_name;
V_ID V_NAME
```

```
---- ------
   1 leekmo
1 row fetched.
gSQL> FETCH cur1 INTO :v_id, :v_name;
V_ID V_NAME
---- ------
   2 mkkim
1 row fetched.
gSQL> FETCH cur1 INTO :v_id, :v_name;
V_ID V_NAME
---- ------
   3 xcom73
1 row fetched.
gSQL> FETCH cur1 INTO :v_id, :v_name;
no rows fetched.
gSQL> FETCH ABSOLUTE 2 cur1 INTO :v_id, :v_name;
V_ID V_NAME
---- ------
   2 mkkim
1 row fetched.
gSQL> FETCH PRIOR cur1 INTO :v_id, :v_name;
V_ID V_NAME
---- ------
   1 leekmo
1 row fetched.
gSQL> CLOSE cur1;
Cursor closed.
```

The created table t1 is dropped and the gsql interactive mode is terminated as follows.

```
gSQL> DROP TABLE t1;
Table dropped.
gSQL> COMMIT;
Commit complete.
gSQL> \q
%
```

40.2 Executing gsql

Information of gsql

Refer to the following options.

Table 40-2 Options

Option	Description
--help	Option list
--version	gsql version information

A user can view the gsql command options which are executable at the shell prompt via **--help** option as follows.

```
% gsql --help
Usage
    gsql [user_name [password]] [options]
Arguments:
    user_name       user name
    password        password
Options:
    --version                    print version information and exit
    --import       FILE          import sql FILE
... Ellipsis ...
%
```

gsql program has a version separate from GOLDILOCKS server version, and the version is viewed by using **--version** option as follows. It is recommended to use the gsql program in the same version as the GOLDIL OCKS server.

```
% gsql --version
%
```

The version of GOLDILOCKS server can be view via **VERSION** function as follows.

```
gSQL> SELECT version() FROM dual;
VERSION()
-----------------------------------
1 row selected.
```

Server Connection

Refer to the following options.

Table 40-3 Options

Option	Description
Username Password	It is the user and password of connection.
--as {SYSDBA|ADMIN}	It specifies the connection role.
--conn-string	It specifies the connection string.
--dsn	It specifies Data Source Name (DSN).

User Connection

Using gsql program, it can be connected to GOLDILOCKS server in various ways.

The simplest way to connect is using a username and password as follows. The following is an example t hat the test user connects to it by using the test password. When successfully connected, it operates in an interactive mode and the *gSQL>* prompt waits for the user command.

```
% gsql test test
Connected to GOLDILOCKS Database.
gSQL>
```

The following is an example that an error occurs when using an invalid username or password. When the login is failed, an error message is displayed and then the gsql program is terminated.

```
% gsql test invalid_password
ERR-28000(16004): invalid username/password; logon denied
%
```

The following is an example that an error occurs when the GOLDILOCKS server does not run. The gsql pr ogram is terminated with the error message as follows.

```
% gsql test test
ERR-HY000(11031): Unable to attach the shared memory segment
%
```

In addition, it can be connected via the following options. For more details, refer to the respective link.

- **--conn-string**
- **--dsn**

SYSDBA Connection

The **--as {SYSDBA|ADMIN}** option is used to drive the server or to connect to SYSDBA role which has full access privilege on the database.

The following is an example of connecting with --as sysdba in the state of when the GOLDILOCKS server is not driven. *gSQL>* prompt is waiting in the state which is ready to drive GOLDILOCKS with the message *Connected to an idle instance*.

```
% gsql sys gliese --as sysdba
Connected to an idle instance.
gSQL>
```

The following is an example of connecting with --as sysdba when the GOLDILOCKS server is already driven.

```
% gsql sys gliese --as sysdba
Connected to GOLDILOCKS Database.
gSQL>
```

For more information, refer to **Startup and Shutdown Server**.

Quit gsql

gsql is quit by using `\quit` or `\q` command as follows.
For more information, refer to **`\quit`**.

```
gSQL> \quit
%
```

Controlling Interactive Mode

Refer to the following options.

Table 40-4 Options

Option	Description
--prompt	It specifies the prompt.
--no-prompt	It removes the prompt.
--import	It executes the SQL file with input.
--silent	When executing the SQL file, the command and the result are not displayed.

Option	Description
--enable-color	The query result is displayed by grouping per row.

Controlling Prompt

When starting an interactive mode, the prompt may be changed by using --prompt option as follows. Th e following is an example that the prompt is changed to *GOLDILOCKS>* prompt, not to *gSQL>* prompt.

```
% gsql test test --prompt GOLDILOCKS
Connected to GOLDILOCKS Database.
GOLDILOCKS>
```

For more information about controlling the prompt, refer to the following options.

- --prompt
- --no-prompt

Executing from File

When processing batch operation by using gsql program, the SQL statement included in the file may be e xecuted by using --import option. The following is an example of executing PerformanceViewSchema.sql file in the $GOLDILOCKS_HOME/admin directory. The example uses --silent option not to output the exec ution results.

```
% cd $GOLDILOCKS_HOME/admin
% gsql --as SYSDBA --import 'PerformanceViewSchema.sql' --silent
%
```

For more information about executing gsql from the file, refer to the following options.

- --import
- --silent
- --enable-color

Storing gsql configuration

gsql Configuration File (gsql.ini)

The gsql configuration file (.gsql.ini) consists of options which is automatically applied when gsql starts u p, and it should be in the directory $HOME.

A user can set multiple dsn in the configuration file, and can specify different options for each dsn. If a us er do not enter dsn when gsql starts up, the options in GOLDILOCKS are applied.

Example

```
[GOLDILOCKS]
AUTOCOMMIT = OFF
AUTOTRACE = OFF
LINESIZE = 80
PAGESIZE = 20
VERTICAL = OFF
TIME = OFF
TIMING = OFF
ERROR = ON
COLSIZE = 8192
NUMSIZE = 20
DDLSIZE = 10000
HISTORY = 128
```

Table 40-5 Options

Option	Minimum value	Maximum value	Default value
AUTOCOMMIT	OFF	ON	OFF
AUTOTRACE	OFF	ON/TRACEONLY	OFF
LINESIZE	1	10000	80
PAGESIZE	1	10000	20
VERTICAL	OFF	ON	OFF
TIME	OFF	ON	OFF
TIMING	OFF	ON	OFF
ERROR	ON	OFF	ON
COLSIZE	1	10485760	8192
NUMSIZE	1	50	20
DDLSIZE	1	10485760	10000
HISTORY	0	100000	128

If .gsql.ini does not exist or the option value is invalid, it is ignored.

glogin.sql

glogin.sql is a global configuration file and it is positioned in $GOLDILOCKS_DATA/conf/glogin.sql. If glogin.sql file exists when driving gsql, then it reads the file and executed the statement included in the file. In this way, all gsql sessions are set (e.g.line size).

login.sql

login.sql is a user configuration file. If login.sql file exists in the current directory when driving gsql, then it reads the file and executed the statement included in the file.
The login.sql configuration takes precedence over the glogin.sql configuration.

40.3 Using Interactive Command

gsql Interactive Mode Commnad

Refer to the following commands.

Table 40-6 Commands

gsql command	Description
\help	gsql command list

gsql program operates in an interactive mode through **Server Connection** as follows.

```
% gsql test test
Connected to GOLDILOCKS Database.
gSQL>
```

A user may enter the SQL statement on the prompt line then execute it or enter the unique command wh ich is executable in an gsql interactive mode then execute it. The gsql command begins with back-slash (\) to distinguish it from the SQL statement. The gsql command which begins with back-slash(\) can no t be used in the application, while SQL statement can be directly used in the application.


The gsql interactive mode command begins with \.

The following is an example of executing SQL statement in an interactive mode.

```
gSQL> SELECT * FROM dual;
DUMMY
-----
X
1 row selected.
```

The following is an example of executing the same SELECT statement by using **\exec sql**, gsql command. \exec sql command is a unique gsql command although the syntax of \ exec sql is as same as the synta x of EXEC SQL referring to the beginning of SQL statement in an **Embedded SQL**.

```
gSQL> \exec sql SELECT * FROM dual;
DUMMY
-----
```

```
X
1 row selected.
```

The current time can be output on the prompt by setting via **\set time**.

```
gSQL> \set time on
12:45:34 gSQL> SELECT * FROM dual;
DUMMY
-----
X
1 row selected.
12:45:37 gSQL>
```

The unique commands of gsql interactive mode can be queried with the command **\help** as follows.

```
gSQL> \help
\help
\q[uit]
\i[mport] {'FILE'}                Import SQL
\ed[it] [{'FILE'|[HISTORY] num}] Edit SQL statement
\\                                Executes the most recent history entry
\{n}                              Executes n'th history entry
\hi[story]                        Show history entries
\desc     {[schema.]table_name}   Show table description
\idesc    {[schema.]index_name}   Show index description
\spo[ol]  ['filename' | OFF]      Stores query results in a file
\ho[st]   [command]               Executes an operating system command
\set vertical     {ON|OFF}
\set time         {ON|OFF}
\set timing       {ON|OFF}
\set color        {ON|OFF}
\set error        {ON|OFF}
\set autocommit   {ON|OFF}
\set autotrace    {ON|TRACEONLY|OFF}
\set serveroutput {ON|OFF}
\set heading      {ON|OFF}
\set linesize     {n}      0 < n <= 100000
\set pagesize     {n}      0 < n <= 100000
\set colsize      {n}      0 < n <= 104857600
\set numsize      {n}      0 < n <= 50
\set ddlsize      {n}      0 < n <= 100000
\set history      {n}      n <= 100000 ( if n < 0, clear history buffer )
```

```
\set sqlprompt    "prompt_sql"
\var              {host_var_name} {INTEGER|BIGINT|VARCHAR(n)}
\exec             [{:host_var_name} := {constant}]
\exec sql         {sql string}
\prepare sql      {sql string}
\dynamic sql      {host_var_name}
\explain plan     [{ON|ONLY}] {sql string}
\print            [{host_var_name}]
\ddl_cluster
\ddl_db
\ddl_tablespace    {name}
\ddl_profile       {name}
\ddl_auth          {name}
\ddl_schema        {name}
\ddl_table         {[schema.]name}
\ddl_constraint    {[schema.]name}
\ddl_index         {[schema.]name}
\ddl_view          {[schema.]name}
\ddl_sequence      {[schema.]name}
\ddl_synonym       {[schema.]name}
\ddl_public_synonym {name}
\ddl_procedure     {[schema.]name}
\ddl_package       {[schema.]name}
\startup          {[nomount|mount|open]}
\shutdown         {[abort|immediate|transactional|normal]}
\cstartup         {[nomount|mount|open]}
\cshutdown        {[abort|immediate|transactional|normal]}
\connect          userid password [as {sysdba|admin}]
```

Startup and Shutdown Server

Refer to the following commands.

Table 40-7 Commands

gsql Commad	Description
\startup	It starts up the server.
\shutdown	It shuts down the server.
\cstartup	It starts up the server for the cluster environment.
\cshutdown	It shuts down the server for the cluster environment.

Standalone

It should be connected by using SYSDBA role or ADMIN role as follows to startup or shutdown the server.

- SYSDBA role
  - It has all privileges related to DBA such as the server startup or shutdown.
- ADMIN role
  - It has the same privileges as SYSDBA but it is allowed to connect only one session.
  - It is the role for when a valid session does not exist or for emergency connection under abnormal circumstances.

The following is an example of connecting by using SYSDBA role when the GOLDILOCKS server is not driven.

```
% gsql sys gliese --as sysdba
Connected to an idle instance.
gSQL>
```

gsql is connected to an idle instance in the state of when GOLDILOCKS server does not exist. gsql drives the GOLDILOCKS server on the specific step through **\startup** and connects to the server. The following is an example of driving the GOLDILOCKS server on NOMOUNT phase.

```
gSQL> \startup NOMOUNT
Startup success
gSQL>
```

\startup command is used only when initially driving GOLDILOCKS. Each step of the GOLDILOCKS server is switched by using **ALTER SYSTEM {MOUNT | OPEN} DATABASE** after driving the GOLDILOCKS server and connecting to the server.

```
gSQL> ALTER SYSTEM MOUNT DATABASE;
System altered.
gSQL> ALTER SYSTEM OPEN DATABASE;
System altered.
gSQL>
```

\startup command is executed without any extra option as follows when GOLDILOCKS server is driven on OPEN phase whose service is available without switching the step.

```
% gsql sys gliese --as sysdba
Connected to an idle instance.
gSQL> \startup
Startup success
```

```
gSQL>
```

If the server is already driven on a certain step, \startup command outputs an error message as follows. Unless the server is driven up to OPEN phase, it may be driven on OPEN phase by using ALTER SYSTEM as follows.

```
% gsql sys gliese--as sysdba
Connected to GOLDILOCKS Database.
gSQL> \startup
ERR-HY000(11029): shared memory segment exists
gSQL> ALTER SYSTEM MOUNT DATABASE;
System altered.
gSQL> ALTER SYSTEM OPEN DATABASE;
System altered.
gSQL>
```

For more information about the server startup, refer to the followings.

- Step of the server startup: **Multi-level Startup**
- Command for the initial server startup: **\startup**
- Altering the server step: **ALTER SYSTEM {MOUNT | OPEN} DATABASE**

\shutdown command is used as follows to shutdown the server. For more information about the server shutdown, refer to **\shutdown**.

```
gSQL> \shutdown
Shutdown success
gSQL>
```

Cluster

LOCATOR_DSN property should be defined in odbc.ini file to startup or shutdown the server by using \cstartup command and \cshutdown command, and it should be connected with SYSDBA role or ADMIN role by using gsqlnet.

The following is an example of connecting with SYSDBA role by using gsqlnet when GOLDILOCKS server is not driven.

```
% gsqlnet sys gliese --as sysdba
Connected to an idle instance.
gSQL>
```

The following is an example of driving GOLDILOCKS server on LOCAL OPEN phase by using \cstartup co

mmand. Then it switches the server to OPEN phase.

```
gSQL> \cstartup LOCAL OPEN
Startup success
gSQL> ALTER SYSTEM OPEN GLOBAL DATABASE;
System altered.
gSQL>
```

\cstartup command and \cshutdown command are used in C/S environment, so glsnr should be on servic e on all servers to startup or shutdown the server.

The following is an example of trying to drive GOLDILOCKS server when glsnr is not on service.

```
gSQL> \cstartup
ERR-HY000(58000): MEMBER(G1N1): the sender failed to connect to the member(1)
ERR-HY000(11067): MEMBER(G1N1): fail to connect to an host with a socket : connect() :
stnConnect() returned errno(111)
gSQL>
```

The following is an example of defining LOCATOR_DSN property and LOCATOR DSN property in **odbc.ini file**.

```
% cat .odbc.ini
# Edit the SYSTEM or USER DSN ini file (/etc/odbc.ini or ~/.odbc.ini) and add a data source
using the syntax:
[GOLDILOCKS]
HOST=127.0.0.1
PORT=20101
UID=test
PWD=test
LOCATOR_DSN=LOCATOR
[LOCATOR]
FILE=/home/test/.locator.ini
```

The following is an example of trying to drive the server when LOCATOR_DSN property is not defined in odbc.ini file.

```
gSQL> \cstartup
ERR-HY000(40057): not specified valid location information
gSQL>
```

Commands excluding \CSTARTUP {LOCAL | GLOBAL} OPEN are applied only to the corresponding GOLDI LOCKS server. Therefore, other servers should be started up to LOCAL OPEN phase for multi-level startup

by using **ALTER SYSTEM {MOUNT | OPEN} DATABASE** after `\CSTARTUP NOMOUNT, MOUN` commands.

The following is an example of trying multi-level startup after `\CSTARTUP MOUNT`. The corresponding server i s started on MOUNT phase, but the other server is not started up to LOCAL OPEN phase, so an error occu rs.

```
gSQL> \CSTARTUP MOUNT
Startup success
gSQL> ALTER SYSTEM OPEN LOCAL DATABASE;
System altered.
gSQL> ALTER SYSTEM OPEN GLOBAL DATABASE;
ERR-HY000(58000): MEMBER(G1N1): the sender failed to connect to the member(1)
ERR-HY000(11067): MEMBER(G1N1): fail to connect to an host with a socket : connect() :
stnConnect() returned errno(111)
gSQL>
```

For more information about the server startup, refer to the followings.

- Step of the server startup: **Multi-level Startup**
- Command for the initial server startup: **\cstartup**
- Altering the server step: **ALTER SYSTEM {MOUNT | OPEN} DATABASE**

\cshutdown command is used as follows to shutdown the entire server. For more information about the e ntire server shutdown, refer to **\cshutdown**.

```
gSQL> \cshutdown
Shutdown success
gSQL>
```

Executing SQL Statements

Refer to the following commands.

Table 40-8 Commands

gsql commad	Description
\import	It executes SQL file.
\set autocommit	It automatically COMMITs whenever executing the SQL statement.
SQL References	It is a type of SQL statement.

Entering SQL Statement

In an interactive mode, gsql outputs the interactive mode prompt gSQL> and waits for a user input. After entering the SQL statement, gsql performs the SQL statement by transferring it to the server. In an interactive mode, the SQL statement is completed by entering semi-colon (;), then entering `enter`. After executing the SQL statement, gsql outputs the result and the prompt gSQL>, and then waits for user input.

The following is an example of entering the SQL statement on a single line and executing it.

```
gSQL> SELECT * FROM dual;
DUMMY
-----
X
1 row selected.
gSQL>
```

The following is an example of entering SQL statement across multiple lines and executing it. If `enter` is used in the state of which the SQL statement is not completed, gsql outputs the line number, moves to the next line and waits for user input.

```
gSQL> SELECT id, name
2
```

When a user continuously completes entering all SQL statements, then enters semicolon (;) and `enter` as follows, the SQL statement is executed and *gSQL>* prompt waits for the next command.

```
gSQL> SELECT id, name
2 FROM t1
3 WHERE id > 0;
ID NAME
-- ------
 1 leekmo
 2 mkkim
 3 xcom73
3 rows selected.
gSQL>
```

The SQL statement is not recognized to be completed of input because the semi-colon (;) within the single quote (') is recognized as a string.

```
gSQL> SELECT id, ';'
2 FROM t1
3 WHERE id = 1;
```

```
ID ';'
-- ---
 1 ;
1 row selected.
gSQL>
```

Entering SQL Statement from File

In an interactive mode, `\import` is used to execute the SQL statement from the file, not to execute it by dir ectly entering the SQL statement. The following is an example of executing multiple SQL statements by re ading the file sample.sql. For more information, refer to **`\import`**.

```
gSQL> \import 'sample.sql'
```

- Drop table

```
DROP TABLE IF EXISTS t1;
Table dropped.
```

- Create table

```
CREATE TABLE t1
(
    id   INTEGER PRIMARY KEY,
    name VARCHAR(128),
    addr VARCHAR(128)
);
Table created.
```

- Create index

```
CREATE INDEX t1_idx_name ON t1(name);
Index created.
```

- Insert three rows

```
INSERT INTO t1
       VALUES ( 1, 'leekmo', 'Seoul, Korea'  ),
              ( 2, 'mkkim' , 'Seoul, Korea'  ),
              ( 3, 'xcom73', 'Inchon, Korea' );
3 rows created.
```

- Commit transaction

```
COMMIT;
Commit complete.
gSQL>
```

gsql Comment

The comments in an interactive mode are as same as those in the SQL statement. The comments input fr om the file via the **--import** option is also as same as those in the SQL statement. For more information ab out comments on SQL statement, refer to **Comments**.

The following is an example of using the line comment in an interactive mode.

```
gSQL> -- outside line-comment
gSQL> SELECT id, name
2 FROM t1 -- inside line-comment
3 WHERE id = 1;
ID NAME
-- ------
 1 leekmo
1 row selected.
gSQL>
```

The following is an example of executing the file content with the `\import` command. It is treated in the s ame way as the comment in an interactive mode. And *gSQL>* prompt waits for the next SQL statement af ter all contents of the file are executed.

```
gSQL> \import 'sample.sql'
```

- Drop table

```
DROP TABLE IF EXISTS t1;
Table dropped.
```

- Create table

```
CREATE TABLE t1
(
    id   INTEGER PRIMARY KEY,
    name VARCHAR(128),
    addr VARCHAR(128)
```

```
);
Table created.
```

- Create index

```
CREATE INDEX t1_idx_name ON t1(name);
Index created.
```

- Insert three rows

```
INSERT INTO t1
       VALUES ( 1, 'leekmo', 'Seoul, Korea'  ),
              ( 2, 'mkkim' , 'Seoul, Korea'  ),
              ( 3, 'xcom73', 'Inchon, Korea' );
3 rows created.
```

- Commit transaction

```
COMMIT;
Commit complete.
gSQL>
```

The following is an example of using the multi-line comment in an interactive mode.

```
gSQL> SELECT id, name
2     /*
3     multi-line
4     comment
5     */
6     FROM t1
7     WHERE id = 1;
ID NAME
-- ------
 1 leekmo
1 row selected.
gSQL>
```

The comments which is available in gsql interactive mode are as same as those in the SQL statement. For more information about the comments, refer to **Comments** of SQL statement.

SQL Execution Result

When a user performs a query such as SELECT statement in an interactive mode, gsql outputs the query results. The query result consists of the header with the column names, the data of the query result, and summary the query results at the end.

The following is an example of executing the SELECT statement.

```
gSQL> SELECT * FROM t1;
ID NAME   ADDR
-- ------ -------------
 1 leekmo Seoul, Korea
 2 mkkim  Seoul, Korea
 3 xcom73 Inchon, Korea
3 rows selected.
gSQL>
```

In the example above, the header of output results consists of the column name and the hyphen (-) which distinguishes it from data. The query result is displayed in the middle of the output result, each row of query result is displayed on a single line, and each column value is separated with the space. The summary of the query result is displayed at the end. In this case, it refers that three rows are retrieved.

Data Definition Language, **Data Manipulation Language**, **Control Language** statements (except for SQL query) displays the summarized information of the execution result corresponding to the SQL statements characteristics.

The followings are the examples of respectively executing the DDL, DML, Control Language, displaying the results.

- DDL statement

```
gSQL> CREATE TABLE t1 ( id INTEGER, name VARCHAR(128) );
Table created.
```

- DML statement

```
gSQL> INSERT INTO t1 VALUES ( 1, 'leekmo' );
1 row created.
```

- Transaction control statement

```
gSQL> COMMIT;
Commit complete.
gSQL>
```

SQL Execution Error

The error which occurs during executing the SQL statement consists of the following information when it.

- SQLSTATE information: SQLSTATE value of SQL standard
- Error code: Unique error code value of GOLDILOCKS
- Error message: An error message describing the cause of error

The following is an example of an error when executing the SELECT statement. The error message consist s of that value of ERR-42000 corresponds to SQLSTATE and the unique error code of GOLDILOCKS is the value of (16040) and a table or view corresponding to invalid_table does not exist at line 2.

```
gSQL> SELECT id, name
2       FROM invalid_table
3      WHERE id > 0;
ERR-42000(16040): table or view does not exist :
      FROM invalid_table
           *
ERROR at line 2:
gSQL>
```

Interactive Command Execution Result

The interactive commands used together with SQL statements displays the same results and errors as tho se executing SQL statements.

The following is an example of preparing the SQL statement by using **`\prepare sql`** and executing it by u sing **`\exec`**.

```
gSQL> \prepare sql SELECT * FROM t1;
SQL prepared.
gSQL> \exec
ID NAME   ADDR
-- ------ -------------
 1 leekmo Seoul, Korea
 2 mkkim  Seoul, Korea
 3 xcom73 Inchon, Korea
3 rows selected.
gSQL>
```

The following is an example of an error occurred in the SQL statement when using **`\prepare sql`**. It displa ys an error which is as same as that occurs when executing the SQL statement.

- An error of \prepare sql command

```
gSQL> \prepare sql SELECT * FROM invalid_table;
ERR-42000(16040): table or view does not exist :
SELECT * FROM invalid_table
              *
ERROR at line 1:
```

- An error of the same SQL statement

```
gSQL> SELECT * FROM invalid_table;
ERR-42000(16040): table or view does not exist :
SELECT * FROM invalid_table
              *
ERROR at line 1:
gSQL>
```

If the gsql commands in an interactive mode do not include the SQL statement or when the command do es not have any result to display, only *gSQL>* prompt is displayed without any extra message when the co mmand is successfully performed.

The following is an example of successfully performing the interactive command without a message. The error message is displayed when an error occurs.

- When the interactive gsql command is normally performed

```
gSQL> \var v1 INTEGER
gSQL> \exec :v1 := 1
```

- When the interactive gsql command causes an error

```
gSQL> \var v2 INVALID TYPE
ERR-42000(40000): syntax error
\var v2 INVALID TYPE
........^     ^
Error at line 1
gSQL>
```

For more information about the performance result of the interactive command, refer to the description and example of the interactive command in **Interactive Command References**.

Autocommitting SQL statement

The SQL statements in the gsql interactive mode may commit or roll back the transaction by using the **COMMIT** or **ROLLBACK** statement.

The following is an example of performing COMMIT or ROLLBACK after performing multiple INSERT statements. Whereas the first INSERT statement is successfully committed by COMMIT, the second and third INSERT statements are rolled back by ROLLBACK in the following example.

```
gSQL> INSERT INTO t1 VALUES ( 1, 'leekmo' );
1 row created.
gSQL> COMMIT;
Commit complete.
gSQL> INSERT INTO t1 VALUES ( 2, 'mkkim' );
1 row created.
gSQL> INSERT INTO t1 VALUES ( 3, 'xcom73' );
1 row created.
gSQL> ROLLBACK;
Rollback complete.
gSQL> SELECT * FROM t1;
ID NAME
-- ------
 1 leekmo
1 row selected.
gSQL>
```

If a user wants to automatically COMMIT every transaction whenever performing the SQL statements, it can be controlled by using the **`\set autocommit`** as follows. When performing the same SQL statement as the example above, each SQL statement is automatically committed, so the transaction of the second and the third INSERT statements are completed regardless of ROLLBACK.

```
gSQL> \set autocommit on
gSQL> INSERT INTO t1 VALUES ( 1, 'leekmo' );
1 row created.
gSQL> COMMIT;
Commit complete.
gSQL> INSERT INTO t1 VALUES ( 2, 'mkkim' );
1 row created.
gSQL> INSERT INTO t1 VALUES ( 3, 'xcom73' );
1 row created.
gSQL> ROLLBACK;
Rollback complete.
```

```
gSQL> SELECT * FROM t1;
ID NAME
-- ------
 1 leekmo
 2 mkkim
 3 xcom73
3 rows selected.
gSQL>
```

For more information about controlling automatic COMMIT, refer to **\set autocommit**.

Forced Termination of SQL Statement Being Executed

SQL statements being executed in an interactive mode is forcibly terminated by entering Ctrl+C.

The following is an example of which the SELECT statement being executed for a long time is forcibly ter minated. If Ctrl+C is entered by using a keyboard during performing the SQL statement, the error messag e, operation canceled, is displayed and the SQL statement is forcibly terminated.

```
gSQL> SELECT COUNT(*)
  FROM
       t1 AS v01,
       t1 AS v02,
       t1 AS v03,
       t1 AS v04,
       t1 AS v05,
       t1 AS v06,
       t1 AS v07,
       t1 AS v08,
       t1 AS v09,
       t1 AS v10,
       t1 AS v11,
       t1 AS v12,
       t1 AS v13,
       t1 AS v14,
       t1 AS v15,
       t1 AS v16,
       t1 AS v17,
       t1 AS v18,
       t1 AS v19,
       t1 AS v20;
```

```
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
^C
ERR-HY008(13043): operation canceled
gSQL>
```

Controlling Output Result

Refer to the following commands.

Table 40-9 Commands

gsql command	Description
\set color	It identifies each row with different color.
\set colsize	It controls the maximum size of the column results.
\set error	It controls whether an error message is displayed.
\set linesize	It controls the maximum output length of the row.
\set numsize	It controls the maximum number of digit of the numeric value.
\set pagesize	It controls the number of row to be included in a page.
\set timing	It outputs the execution time of the SQL statement.
\set vertical	It displays each column in a single line.

Controlling Page Configuration

As described in **SQL Execution Result**, the query results display the column names in the header, displays t he data, then the summary information at the end.

If there are many query results, they are divided into the unit of pagesize (default value is 20), and each p age outputs the column names and data set. When the output of all pages is completed, the summary in formation is displayed at the end.

The following is an example of the output of query results by changing the pagesize to 5. For more infor mation about controlling the pagesize, refer to **\set pagesize**.

```
gSQL> \set pagesize 5
gSQL> SELECT table_schema, table_name FROM dictionary WHERE table_name like 'ALL_%' FETCH 20;
TABLE_SCHEMA      TABLE_NAME
----------------- ------------------
DICTIONARY_SCHEMA ALL_ALL_TABLES
DICTIONARY_SCHEMA ALL_COL_COMMENTS
DICTIONARY_SCHEMA ALL_COL_PRIVS
DICTIONARY_SCHEMA ALL_COL_PRIVS_MADE
```

```
DICTIONARY_SCHEMA ALL_COL_PRIVS_RECD
TABLE_SCHEMA      TABLE_NAME
----------------- -----------------
DICTIONARY_SCHEMA ALL_CONSTRAINTS
DICTIONARY_SCHEMA ALL_CONS_COLUMNS
DICTIONARY_SCHEMA ALL_DB_PRIVS
DICTIONARY_SCHEMA ALL_DB_PRIVS_MADE
DICTIONARY_SCHEMA ALL_DB_PRIVS_RECD
TABLE_SCHEMA      TABLE_NAME
----------------- -----------------
DICTIONARY_SCHEMA ALL_INDEXES
DICTIONARY_SCHEMA ALL_IND_COLUMNS
DICTIONARY_SCHEMA ALL_SCHEMAS
DICTIONARY_SCHEMA ALL_SCHEMA_PATH
DICTIONARY_SCHEMA ALL_SCHEMA_PRIVS
TABLE_SCHEMA      TABLE_NAME
----------------- ---------------------
DICTIONARY_SCHEMA ALL_SCHEMA_PRIVS_MADE
DICTIONARY_SCHEMA ALL_SCHEMA_PRIVS_RECD
DICTIONARY_SCHEMA ALL_SEQUENCES
DICTIONARY_SCHEMA ALL_SEQ_PRIVS
DICTIONARY_SCHEMA ALL_SEQ_PRIVS_MADE
20 rows selected.
gSQL>
```

Each row of the query result is output in linesize unit (default value is 80). If the row length of the result i s bigger than linesize, it is output on the next line.

The following is an example of the query result before and after the linesize is adjusted to 200. For more i nformation about controlling linesize, refer to **`\set linesize`**.

```
gSQL> SELECT * FROM dict_columns WHERE table_name = 'USER_TABLES' FETCH 3;
TABLE_SCHEMA      TABLE_NAME  COLUMN_NAME
----------------- ----------- ---------------
COMMENTS
----------------------------------------
DICTIONARY_SCHEMA USER_TABLES TABLE_SCHEMA
Schema of the table
DICTIONARY_SCHEMA USER_TABLES TABLE_NAME
Name of the table
DICTIONARY_SCHEMA USER_TABLES TABLESPACE_NAME
Name of the tablespace containing the table
```

```
3 rows selected.
```

- Set the linesize to 200.

```
gSQL> \set linesize 200
gSQL> SELECT * FROM dict_columns WHERE table_name = 'USER_TABLES' FETCH 3;
TABLE_SCHEMA      TABLE_NAME  COLUMN_NAME     COMMENTS
----------------- ----------- --------------- ----------------------------------------
DICTIONARY_SCHEMA USER_TABLES TABLE_SCHEMA    Schema of the table
DICTIONARY_SCHEMA USER_TABLES TABLE_NAME      Name of the table
DICTIONARY_SCHEMA USER_TABLES TABLESPACE_NAME Name of the tablespace containing the table
3 rows selected.
gSQL>
```

To increase the readability of each row on the terminal by differentiating rows, each row can be output b y varying the color through **\set color** as described below. For more information, refer to **\set color**.

```
gSQL> \set color on
gSQL> SELECT table_schema, table_name FROM dictionary WHERE table_name like 'ALL_%' FETCH 10;
TABLE_SCHEMA      TABLE_NAME
----------------- ------------------
DICTIONARY_SCHEMA ALL_ALL_TABLES
DICTIONARY_SCHEMA ALL_COL_COMMENTS
DICTIONARY_SCHEMA ALL_COL_PRIVS
DICTIONARY_SCHEMA ALL_COL_PRIVS_MADE
DICTIONARY_SCHEMA ALL_COL_PRIVS_RECD
DICTIONARY_SCHEMA ALL_CONSTRAINTS
DICTIONARY_SCHEMA ALL_CONS_COLUMNS
DICTIONARY_SCHEMA ALL_DB_PRIVS
DICTIONARY_SCHEMA ALL_DB_PRIVS_MADE
DICTIONARY_SCHEMA ALL_DB_PRIVS_RECD
10 rows selected.
gSQL>
```

To output the query result rows in the column unit on a single line, not in the line unit, `\set vertical on` i s used. The following is an example of displaying the query results in a column unit. For more information, refer to **\set vertical**.

```
gSQL> SELECT * FROM v$system_stat FETCH 5;
STAT_NAME             STAT_VALUE COMMENTS
--------------------- ---------- ------------------------------------------------------------
SYSTEM_SAR                     3 system available resource( 0:none, 1:session 2:database )
```

```
MAX_ENVIRONMENT_COUNT        128 maximum environment count
FREE_ENVIRONMENT_ID            3 available environment identifier
MAX_SESSION_COUNT            128 maximum session count
FREE_SESSION_ID                4 available session identifier
5 rows selected.
gSQL> \set vertical on
gSQL> SELECT * FROM v$system_stat FETCH 5;
               STAT_NAME # SYSTEM_SAR
              STAT_VALUE # 3
                COMMENTS # system available resource( 0:none, 1:session 2:database )
               STAT_NAME # MAX_ENVIRONMENT_COUNT
              STAT_VALUE # 128
                COMMENTS # maximum environment count
               STAT_NAME # FREE_ENVIRONMENT_ID
              STAT_VALUE # 3
                COMMENTS # available environment identifier
               STAT_NAME # MAX_SESSION_COUNT
              STAT_VALUE # 128
                COMMENTS # maximum session count
               STAT_NAME # FREE_SESSION_ID
              STAT_VALUE # 4
                COMMENTS # available session identifier
5 rows selected.
gSQL>
```

Controlling Data Output

gsql converts all data included in the query result to a string, then outputs it.

For a numeric data type, numsize (default value is 20) is output in the digit unit, and a numeric value bigg er than numsize is output in an exponent form. The following is an example of controlling the output by using the numsize. For more information, refer to **`\set numsize`**.

```
gSQL> SELECT num, ( num * num ) AS result FROM t1;
          NUM               RESULT
------------- --------------------
1234567890123 1.52415787532276E+24
1 row selected.
gSQL> \set numsize 50
gSQL> SELECT num, ( num * num ) AS result FROM t1;
          NUM                    RESULT
------------- -------------------------
```

```
1234567890123 152415787532275580095129
1 row selected.
gSQL>
```

The output of the numeric value may be controlled in various ways by using the **TO_CHAR(number)** fun ction as the following example.

```
gSQL> SELECT num, TO_CHAR( num * num, '$999,999,999,999,999,999,999,999,999' ) AS result FROM
t1;
          NUM RESULT
------------- -----------------------------------
1234567890123    $1,524,157,875,322,755,800,955,129
1 row selected.
gSQL>
```

For the date/ time data types such as DATE/ TIME/ TIMESTAMP, gsql determines the output type by using the property information as follows.

- **NLS_DATE_FORMAT**
- **NLS_TIMESTAMP_FORMAT**
- **NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT**
- **NLS_TIME_FORMAT**
- **NLS_TIME_WITH_TIME_ZONE_FORMAT**

The property is used when the format information is not entered in the functions such as TO_CHAR(), TO _DATE(). When connected for the first time, gsql acquires the property information and uses it to output the date/ time data value, whereas the property above is used only when the application using ODBC or J DBC uses the functions such as TO_CHAR(), TO_DATE() in SQL statement. When the property is changed to the following SQL statement in an interactive mode, gsql uses the updated property information.

- **ALTER SESSION SET property_name**
- **ALTER SYSTEM SET property_name**

The following is an example of controlling the output for the DATE column.

```
gSQL> SELECT enter_date FROM t1;
ENTER_DATE
----------
2014-08-26
1 row selected.
gSQL> ALTER SESSION SET NLS_DATE_FORMAT = 'DD-MON-YYYY';
Session altered.
gSQL> SELECT enter_date FROM t1;
```

```
ENTER_DATE
-----------
26-AUG-2014
1 row selected.
gSQL>
```

The output of the date/ time value can be controlled by using **TO_CHAR(datetime)** function without alte ring property as follows.

```
gSQL> SELECT TO_CHAR( enter_date, 'DD-MON-YYYY' ) as result FROM t1;
RESULT
----------------
26-AUG-2016
1 row selected.
gSQL>
```

A binary string such as BINARY or VARBINARY is output as hex values as follows.

```
gSQL> select * from t1;
BINARY_VALUE
--------------------
A010002F370000000000
1 row selected.
gSQL>
```

The data of the LONG VARCHAR, LONG VARBINARY type may include a very long string, so it outputs th e data as much as colsize (default value is 8192). The following is an example of outputting the part of th e TEXT column data of LONG VARCHAR data type by reducing the colsize. The whole data may be outpu t by increasing the colsize. For more information, refer to **`\set colsize`**.

```
gSQL> \set colsize 100
gSQL> SELECT view_name, text FROM all_views WHERE view_name = 'ALL_ALL_TABLES';
VIEW_NAME      TEXT
-------------- ----------------------------------------------------
ALL_ALL_TABLES SELECT
                     auth.AUTHORIZATION_NAME               ❶ OWNER
                   , sch.SCHEMA_NAME
1 row selected.
gSQL>
```

Execution Time of SQL Statement

The execution time of the SQL statement is output by using `\set timing on` as follows. For more information, refer to **`\set timing`**.

```
gSQL> \set timing on
gSQL> SELECT * FROM dual;
DUMMY
-----
X
1 row selected.
Elapsed time: 0.08500 ms
```

Logging Output Result

All results performed in gsql are simultaneously logged to the file when they are output on the terminal.

Table 40-10 Commands

gsql command	Description
`\spool`	It logs the output result to a file.

For more information about the features related to logging output results, refer to the commands of **`\spool`**.

The output result is logged as the following example.

- Start to log the execution result on result.txt

```
gSQL> \SPOOL 'result.txt'
```

- Check the spool status

```
gSQL> \SPOOL

currently spooling to result.txt
gSQL> SELECT * FROM T1 WHERE C1 < 10;
```

- Stop the spool feature

```
gSQL> \SPOOL OFF
```

Querying SQL Object Information

The information about SQL objects may be queried with the views of **DICTIONARY_SCHEMA**, **INFORMATION_SCHEMA**.

Table 40-11 Commands

gsql command	Description
\desc	It queries the table information.
\idesc	It queries the index information.

For example, there are the table and index created as follows.

```
CREATE TABLE t1
(
    id   INTEGER PRIMARY KEY,
    name VARCHAR(128),
    addr VARCHAR(128)
);
CREATE INDEX t1_idx_name ON t1(name);
```

A user may query the information related to the table by performing a series of SQL statements as follows.

```
gSQL> SELECT table_schema, table_name FROM user_tables WHERE table_name = 'T1';
TABLE_SCHEMA TABLE_NAME
------------ ----------
PUBLIC       T1
1 row selected.
gSQL> SELECT column_name, data_type, nullable FROM user_tab_columns WHERE table_schema =
'PUBLIC' AND table_name = 'T1';
COLUMN_NAME DATA_TYPE         NULLABLE
----------- ----------------- --------
ID          NUMBER            N
NAME        CHARACTER VARYING Y
ADDR        CHARACTER VARYING Y
3 rows selected.
gSQL> SELECT index_name FROM user_indexes WHERE table_schema = 'PUBLIC' AND table_name = 'T1';
INDEX_NAME
--------------------
T1_PRIMARY_KEY_INDEX
T1_IDX_NAME
```

```
2 rows selected.
gSQL>
```

gsql may easily query the information related to the table by using \desc as follows. For more information about the output, refer to **\desc**.

```
gSQL> \desc t1
COLUMN_NAME TYPE                   IS_NULLABLE
----------- ---------------------- -----------
ID          NUMBER(10,0)           FALSE
NAME        VARCHAR(128)           TRUE
ADDR        VARCHAR(128)           TRUE
INDEX_NAME           TABLESPACE_NAME INDEX_TYPE IS_UNIQUE COLUMNS
-------------------- --------------- ---------- --------- -------
T1_PRIMARY_KEY_INDEX MEM_TEMP_TBS    BTREE      TRUE      ID
T1_IDX_NAME          MEM_TEMP_TBS    BTREE      FALSE     NAME
CONSTRAINT_NAME CONSTRAINT_TYPE ASSOCIATED_INDEX     COLUMNS
--------------- --------------- -------------------- -------
T1_PRIMARY_KEY  PRIMARY KEY     T1_PRIMARY_KEY_INDEX ID
gSQL>
```

\idesc describes the information related to the index as the following example. For more information, refer to **\idesc**.

```
gSQL> \idesc t1_idx_name
COLUMN_NAME ORDINAL_POSITION IS_ASCENDING_ORDER IS_NULLS_FIRST
----------- ---------------- ------------------ --------------
NAME                       1 TRUE               FALSE
gSQL>
```

Output DDL Statement of SQL Object

The following gsql commands output the DDL statement corresponding to the current state of the SQL object. As well as the CREATE statement which creates the SQL object, DDL statement of the related object may be output through the various options.

The objects stored in the recycle bin are not output.

Table 40-12 Commands

gsql command	Description
\ddl_cluster	It outputs the cluster-related DDL.
\ddl_db	It outputs the database-related DDL.
\ddl_tablespace	It outputs the tablespace-related DDL.
\ddl_profile	It outputs the profile-related DDL.
\ddl_audit_policy	It outputs the audit policy-related DDL.
\ddl_auth	It outputs the account-related DDL.
\ddl_schema	It outputs the schema-related DDL.
\ddl_public_synonym	It outputs the public synonym-related DDL.
\ddl_table	It outputs the table-related DDL.
\ddl_constraint	It outputs the constraint-related DDL.
\ddl_index	It outputs the index-related DDL.
\ddl_view	It outputs the view-related DDL.
\ddl_sequence	It outputs the sequence-related DDL.
\ddl_synonym	It outputs the synonym-related DDL.
\ddl_procedure	It outputs the procedure-related DDL.
\ddl_package	It outputs the package-related DDL.
\set ddlsize	It controls the size of the DDL output buffer.

For example, the orders table is created as follows.

```
gSQL>
CREATE TABLE ORDERS
(
    O_ID         INTEGER,
    O_D_ID       INTEGER,
    O_W_ID       INTEGER,
    O_C_ID       INTEGER,
    O_ENTRY_D    TIMESTAMP,
    O_CARRIER_ID INTEGER,
    O_OL_CNT     NUMERIC(8),
    O_ALL_LOCAL  NUMERIC(1),
    PRIMARY KEY(O_W_ID, O_D_ID, O_ID) INDEX ORDERS_PK_IDX
);
Table created.
```

The CREATE TABLE statement which created the orders table in the example above is output by using \ddl_table as follows.

```
gSQL> \ddl_table orders CREATE
SET SESSION AUTHORIZATION "TEST";
CREATE TABLE "PUBLIC"."ORDERS"
    (
        "O_ID" NUMBER( 10, 0 )
      , "O_D_ID" NUMBER( 10, 0 )
      , "O_W_ID" NUMBER( 10, 0 )
      , "O_C_ID" NUMBER( 10, 0 )
      , "O_ENTRY_D" TIMESTAMP( 6 ) WITHOUT TIME ZONE
      , "O_CARRIER_ID" NUMBER( 10, 0 )
      , "O_OL_CNT" NUMBER( 8, 0 )
      , "O_ALL_LOCAL" NUMBER( 1, 0 )
    )
    PCTFREE  10
    PCTUSED  60
    INITRANS 4
    MAXTRANS 8
    STORAGE
    (
        INITIAL 524288
        NEXT    262144
        MINSIZE 524288
        MAXSIZE 35184372088832
    )
    TABLESPACE "MEM_DATA_TBS"
;
```

From the example above, SET AUTHORIZATION statement refers to the owner who performed CREATE TABLE statement. The CREATE TABLE statement is output together with the information that the user does not input such as the schema name to which the table belongs, the tablespace name in which the table is stored and the physical information of the table.

DDL corresponding to the constraints generated in the table is output by using the CONSTRAINT option of \ddl_table as follows.

```
gSQL> \ddl_table orders CONSTRAINT
SET SESSION AUTHORIZATION "TEST";
ALTER TABLE "PUBLIC"."ORDERS"
    ADD CONSTRAINT "PUBLIC"."ORDERS_PRIMARY_KEY"
    PRIMARY KEY
    (
        "O_W_ID" ASC NULLS LAST
```

```
      , "O_D_ID" ASC NULLS LAST
      , "O_ID" ASC NULLS LAST
    )
    INDEX "ORDERS_PK_IDX"
    PCTFREE  10
    INITRANS 4
    MAXTRANS 8
    STORAGE
    (
        INITIAL 524288
        NEXT    262144
        MINSIZE 524288
        MAXSIZE 35184372088832
    )
    TABLESPACE "MEM_TEMP_TBS"
    NOT DEFERRABLE
    INITIALLY IMMEDIATE
;
```

For more information, refer to the gsql command corresponding to the SQL object.

> ⚠ If a user uses the command such as `\ddl_table` and simultaneously performs DDL on the relevant SQL object, it may output different results, so DDL should not be simultaneously performed.

Controlling History

gsql controls the history of executed SQL statements in an interactive mode.

Table 40-13 Commands

gsql command	Description
****	It executes the previous SQL.
\{n}	It executes the SQL statement corresponding to the history number.
\history	It queries the SQL execution history.
\set history	It controls the number of history buffer.

The following is an example of executing SQL statement which has already been performed may be performed again.

- The most recent successfully executed SQL statement is executed again.

```
gSQL> \\
DUMMY
-----
X
1 row selected.
gSQL> \history
ID SQL
-- ---------------------------------------------
 1 DROP TABLE IF EXISTS t1
 2 CREATE TABLE t1
   (
       id   INTEGER PRIMARY KEY,
       name VARCHAR(128),
       addr VARCHAR(128)
   )
 3 CREATE INDEX t1_idx_name ON t1(name)
 4 INSERT INTO t1
          VALUES ( 1, 'leekmo', 'Seoul, Korea'  ),
                 ( 2, 'mkkim' , 'Seoul, Korea'  ),
                 ( 3, 'xcom73', 'Inchon, Korea' )
 5 COMMIT
 6 select * from t1
```

- The first SQL in the history is executed.

```
gSQL> \1
Table dropped.
```

For more information about the history related features of gsql, refer to **Commands** .

Editing SQL

It edits the SQL statement by using the text editor. The text editor to be used may be specified in the environment variable EDITOR. If the environment variable EDITOR does not exist, vi is used by default.

Table 40-14 Command

gsql command	Description
\edit	It edits the SQL statement by using the text editor.

When using the editing features, gsql runs the editor and hands over control to it. The user can freely edit

the SQL statements with the editor. When the editor ends, the control is handed over to gsql, and its con tent will be added as the last history.

The SQL statement edited by the editor may be executed through the last history execution command \\.

> ⚠ Only a single SQL statement can be edited with the SQL editor. An error occurs when executing multiple SQL statements.

The followings may be edited by using \edit.

- The most recently performed SQL statement
- The SQL statement stored in a file
- The SQL statement stored in the gsql history

For more information about the edit-related features in gsql, refer to **\edit**.

The SQL statement may be edited as the following example.

- The most recently performed SQL statement is edited.

```
gSQL> SELECT * FROM T1;
C1
--
 1
11
2 rows selected.
gSQL> \EDIT
SELECT * FROM T1 WHERE C1 < 10;
```

- The edited SQL statement is executed.

```
gSQL> \\
C1
--
 1
1 row selected.
```

Controlling Connection

Refer to the following commands.

Table 40-15 Commands

gsql command	Description
\connect	It connects with a new user.
\quit	It quits the connection.

It is connected with a new user by using \connect in an interactive mode as follows. The existing session is terminated and a new session is created by using \connect. For more information, refer to **\connect**.

```
gSQL> \connect test test
gSQL>
```

The user may be changed by using the **SET SESSION AUTHORIZATION user_identifier** statement as follows. \connect commits all transactions being executed and creates a new session whereas the SET SESSION AUTHORIZATION statement changes the user while keeping intact the existing sessions.

```
gSQL> SET SESSION AUTHORIZATION test;
Session set.
gSQL>
```

Interactive mode is terminated by using \quit as follows. For more information, refer to **\quit**.

```
gSQL> \quit
%
```

Using Host Variable

gsql declares a host variable in an interactive mode, and assigns a value to the host variable, then uses the host variable together with the SQL statement.

Table 40-16 Commands

gsql command	Description
\var	It declares the host variable.
\exec :var := value	It assigns the value to the host variable.
\print	It outputs the value of the host variable.
\dynamic sql :var	It executes SQL statements stored in the host variable.

The following is an example of declaring the host variable by using **\var**, and assigning the value to the host variable by using **\exec :var := value**, and querying the value of the host variable by using **\print**. For more information, refer to each command.

```
gSQL> \var v_id INTEGER
gSQL> \exec :v_id := 1
gSQL> \print v_id
V_ID
----
   1
gSQL>
```

The variable declared in an interactive mode may be used as the input parameter in the SQL statement as follows.

```
gSQL> SELECT * FROM t1 WHERE id = :v_id;
ID NAME   ADDR
-- ------ ------------
 1 leekmo Seoul, Korea
1 row selected.
gSQL>
```

The variable declared in an interactive mode may be used as the output parameter in the SQL statement as follows.

```
gSQL> SELECT id INTO :v_id FROM t1 WHERE name = 'mkkim';
V_ID
----
   2
1 row selected.
gSQL>
```

Controlling Method of Treating SQL Statement

The SQL process methods which are frequently used when processing the SQL such as the execution of p repare/ execute statement or the retrieval by using the cursor may be simulated by using the gsql interacti ve mode command before developing the application.

Table 40-17 Commands

gsql command	Description
\exec sql	It directly executes the SQL statement.
\prepare sql	It prepares the SQL statement.
\exec	It executes the prepared SQL statement.
	It executes the SQL statement stored in the host variable.

gsql command	Description
\dynamic sql :var	

The following Java application is a part of code in $GOLDILOCKS_HOME/sample/JDBC/JdbcSample.java.

```
public static void main(String[] args) throws SQLException
    {
        Connection con = createConnectionByDriverManager("TEST", "test");
        Statement stmt = con.createStatement();
        stmt.execute("CREATE TABLE SAMPLE_TABLE ( ID INTEGER, NAME CHAR(20) )");
        PreparedStatement pstmt = con.prepareStatement("INSERT INTO SAMPLE_TABLE VALUES (?,
?)");
        pstmt.setInt(1, 100);
        pstmt.setString(2, "Tom");
        pstmt.executeUpdate();
        pstmt.setInt(1, 200);
        pstmt.setString(2, "Jerry");
        pstmt.executeUpdate();
        ResultSet rs = stmt.executeQuery("SELECT * FROM SAMPLE_TABLE");
        while (rs.next())
        {
            System.out.println("ID = " + rs.getInt(1) + ": " + rs.getString(2));
        }
        rs.close();
        stmt.close();
        pstmt.close();
        con.close();
        Connection con2 = createConnectionByDataSource("TEST", "test");
        Statement stmt2 = con2.createStatement();
        stmt2.execute("DROP TABLE SAMPLE_TABLE");
        stmt2.close();
        con2.close();
    }
```

The implementation above which used the PreparedStatement class of Java code may be simulated by using **\prepare sql** and **\exec** as follows.

```
gSQL> \connect test test
gSQL> \var v_int INTEGER
gSQL> \var v_string VARCHAR(128)
gSQL> CREATE TABLE SAMPLE_TABLE ( ID INTEGER, NAME CHAR(20) );
```

```
Table created.
gSQL> \prepare sql INSERT INTO SAMPLE_TABLE VALUES (:v_int, :v_string);
SQL prepared.
gSQL> \exec :v_int := 100
gSQL> \exec :v_string := 'Tom'
gSQL> \exec
1 row created.
gSQL> \exec :v_int := 200
gSQL> \exec :v_string := 'Jerry'
gSQL> \exec
1 row created.
gSQL> SELECT * FROM SAMPLE_TABLE;
 ID NAME
--- --------------------
100 Tom
200 Jerry
2 rows selected.
gSQL> \connect test test
gSQL> DROP TABLE SAMPLE_TABLE;
Table dropped.
gSQL> \quit
%
```

The following embedded SQL program is a part of code in $GOLDILOCKS_HOME/sample/EmbeddedSQL/sample2.gc.

```
int main(int argc, char **argv)
{
    EXEC SQL BEGIN DECLARE SECTION;
    int          sEmpNo;
    varchar      sEName[20 + 1];
    char         sJob[20];
    long         sSalary;
    EXEC SQL END DECLARE SECTION;
    ... Ellipsis ...
```

- Retrieve employee

```
EXEC SQL
        DECLARE EMP_CUR CURSOR FOR
        SELECT empno, ename, job, sal
        FROM   EMP;
```

```
    EXEC SQL OPEN EMP_CUR;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }
    printf(" EMPNO    ENAME                JOB       SALARY\n");
    printf("====== ==================== ========== ========\n");
    while( 1 )
    {
        EXEC SQL
            FETCH EMP_CUR
            INTO  :sEmpNo, :sEName, :sJob, :sSalary;
        if(sqlca.sqlcode == SQL_NO_DATA)
        {
            break;
        }
        else if(sqlca.sqlcode != 0)
        {
            PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
            goto fail_exit;
        }
        sRecordCount ++;
        printf("%6d %20s %10s %8ld\n",
               sEmpNo, sEName.arr, sJob, sSalary);
    }
    printf("====== ==================== ========== ========\n");
    printf("Record Count = %d\n", sRecordCount);
    printf("====== ==================== ========== ========\n");
    EXEC SQL CLOSE EMP_CUR;
    if(sqlca.sqlcode != 0)
    {
        PRINT_SQL_ERROR("[ERROR] SQL ERROR -");
        goto fail_exit;
    }

    ... Ellipsis ...
}
```

The query processing which uses the cursor in an embedded SQL program above may be simulated in gsq
l interactive mode as follows.

```
gSQL> \var sEmpNo INTEGER
gSQL> \var sEName VARCHAR(20)
gSQL> \var sJob CHAR(20)
gSQL> \var sSalary BIGINT
gSQL> DECLARE EMP_CUR CURSOR FOR
        SELECT empno, ename, job, sal
        FROM   EMP;
Cursor declared.
gSQL> OPEN EMP_CUR;
Cursor is open.
gSQL> FETCH EMP_CUR
            INTO  :sEmpNo, :sEName, :sJob, :sSalary;
SEMPNO SENAME SJOB                SSALARY
------ ------ -------------------- -------
  2854 Park   RND                      800
1 row fetched.
gSQL> FETCH EMP_CUR
            INTO  :sEmpNo, :sEName, :sJob, :sSalary;
SEMPNO SENAME SJOB                SSALARY
------ ------ -------------------- -------
  2098 Kim    SALESMAN                1600
1 row fetched.
gSQL> FETCH EMP_CUR
            INTO  :sEmpNo, :sEName, :sJob, :sSalary;
SEMPNO SENAME SJOB                SSALARY
------ ------ -------------------- -------
  2175 Choi   SALESMAN                1250
1 row fetched.
gSQL> CLOSE EMP_CUR;
Cursor closed.
gSQL>
```

The following statements are limitedly used in an embedded SQL in other DBMS. On the other hand, in GOLDILOCKS, the followings are used not only in an embedded SQL but also can be us ed as an argument of SQL processing functions when developing the application of ODBC or JD BC.

- **SELECT .. INTO**
- **DECLARE cursor_name**
- **OPEN cursor_name**

- FETCH cursor_name
- CLOSE cursor_name

Information of SQL Execution Plan

Refer to the following commands.

Table 40-18 Command

gsql Command	Description
\explain plan	It outputs the execution plan for the SQL statement.
\set autotrace	It sets whether to output the execution plan.

The execution plan of the SQL statement is viewed in an interactive mode by using `\explain plan` or `\set autotrace` as follows. For more information about the command, refer to **\explain plan** or **\set autotrace**, and for more information about the analysis of execution plan, refer to **SQL Execution Plan**.

The following is an example of executing the query No.4 of TPC-H benchmark by using the `\explain plan`.

```
\explain plan
select
    o_orderpriority,
    count(*) as order_count
from
    orders
where
      o_orderdate >= date '1993-07-01'
  and o_orderdate < date '1993-07-01' + interval '3' month
  and exists (
              select
                    *
                from
                    lineitem
               where
                    l_orderkey = o_orderkey
                 and l_commitdate < l_receiptdate
            )
group by
    o_orderpriority
order by
```

```
    o_orderpriority;
O_ORDERPRIORITY ORDER_COUNT
--------------- -----------
1-URGENT              10594
2-HIGH                10476
3-MEDIUM              10410
4-NOT SPECIFIED       10556
5-LOW                 10487
5 rows selected.
>>>  start print plan
< Execution Plan >
================================================================================
¦  IDX  ¦  NODE DESCRIPTION                                        ¦       ROWS ¦
--------------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                       ¦            ¦
¦    1  ¦    SORT INSTANT ACCESS                                 ¦          5 ¦
¦    2  ¦      GROUP HASH INSTANT ACCESS                         ¦          5 ¦
¦    3  ¦        NESTED LOOP JOIN (LEFT SEMI)                    ¦      52523 ¦
¦    4  ¦          TABLE ACCESS ("ORDERS")                       ¦      57218 ¦
¦    5  ¦          INDEX ACCESS ("LINEITEM, LINEITEM_PK_INDEX")  ¦      52523 ¦
================================================================================
     1  -  SORT KEY : "ORDERS.O_ORDERPRIORITY ASC NULLS LAST"
           RECORD COLUMNS : COUNT(*)
           READ COLUMNS : O_ORDERPRIORITY, COUNT(*)
     2  -  AGGREGATIONS : COUNT(*)
           GROUPING COLUMNS : O_ORDERPRIORITY
           RECORD COLUMNS : COUNT(*)
           READ COLUMNS : O_ORDERPRIORITY, COUNT(*)
     3  -  JOINED COLUMNS : ORDERS.O_ORDERPRIORITY
     4  -  READ COLUMNS : O_ORDERKEY, O_ORDERDATE, O_ORDERPRIORITY
             PHYSICAL FILTER : O_ORDERDATE >= CAST( '1993-07-01' AS DATE ) AND O_ORDERDATE < (
CAST( '1993-07-01' AS DATE ) + CAST( '3' AS INTERVAL(MONTH) ) )
     5  -  READ INDEX COLUMNS : L_ORDERKEY
           READ TABLE COLUMNS : L_COMMITDATE, L_RECEIPTDATE
             MIN RANGE : L_ORDERKEY = {O_ORDERKEY}
             MAX RANGE : L_ORDERKEY = {O_ORDERKEY}
             PHYSICAL TABLE FILTER : L_COMMITDATE < L_RECEIPTDATE
<<<  end print plan
```

40.4 Command Option Reference

This chapter describes the options for performing gsql command at the shell prompt.

Username Password

Description

It connects to GOLDILOCKS by using the username and password.

Examples

The following is an example of connecting to GOLDILOCKS with the test user.

```
% gsql test test
Connected to GOLDILOCKS Database.
gSQL>
```

The following is an example of failing to connect to GOLDILOCKS with the invalid username or password.

```
% gsql invalid_user invalid_password
ERR-28000(16004): invalid username/password; logon denied
%
```

--as {SYSDBA|ADMIN}

Description

It connects to GOLDILOCKS with SYSDBA role or ADMIN role.
For more information about the description of the role, refer to **Startup and Shutdown Server**.

```
% gsql sys gliese --as sysdba
```

Example

The following is an example of connecting to GOLDILOCKS with SYSDBA role.

```
% gsql sys gliese --as sysdba
Connected to GOLDILOCKS Database.
gSQL>
```

--conn-string

Description

It connects to GOLDILOCKS by using the connection string.
The details of the connection string to be described should used in a form as same as the input string of t he ODBC function, **SQLDriverConnect**. For more information about the connection string, refer to **SQLDri verConnect** function.

Example

The following is an example of connecting to GOLDILOCKS with --conn-string.

```
% gsql --conn-string 'DSN=GOLDILOCKS;UID=test;PWD=test'
Connected to GOLDILOCKS Database.
gSQL>
```

--dsn

Description

It connects to GOLDILOCKS by using Data Source Name (DSN) defined in the file odbc.ini.
DSN should be defined in the odbc.ini file. For more information about the odbc.ini file, refer to **DSN Con figuration on UNIX**.
If DSN is omitted, the default value is GOLDILOCKS.

Examples

The following is an example of connecting to GOLDILOCKS with DSN.

```
% gsql test test --dsn GOLDILOCKS
Connected to GOLDILOCKS Database.
gSQL>
```

The content of odbc.ini file used in the example above is as follows.

```
[GOLDILOCKS]
DATE_FORMAT = YYYY-MM-DD
TIME_FORMAT = HH24:MI:SS.FF6
```


gsql does not recognize HOST, PORT of odbc.ini file property.

--enable-color

Description

It outputs each row of query results in a different color on the terminal to easily distinguish them.
Each row of the query results is output with a different color in an interactive mode, and it is also applied to the results of the SELECT statement included in the file by using the **--import** option.

Examples

The following is an example of performing gsql by using the --enable-color option, then outputting the query results in an interactive mode.

```
% gsql test test --enable-color
Connected to GOLDILOCKS Database.
gSQL> SELECT id, addr FROM t1;
ID ADDR
-- -------------
 1 Seoul, Korea
 2 Seoul, Korea
 3 Inchon, Korea
3 rows selected.
gSQL>
```

The following is an example of executing a SELECT query included in the file by using the --import option.

```
% gsql test test --import 'sample_select.sql' --enable-color
SELECT id, addr FROM t1;
ID ADDR
-- -------------
 1 Seoul, Korea
 2 Seoul, Korea
 3 Inchon, Korea
3 rows selected.
%
```

--help

Description

It briefly displays the option list of gsql program.

Example

The following is an example of using the --help option.

```
% gsql --help
Usage
    gsql [user_name [password]] [options]
Arguments:
    user_name       user name
    password        password
Options:
    --version                     print version information and exit
    --import       FILE           import sql FILE
    --no-prompt                   suppresses the display of the banner and prompts
    --dsn          DSN            dsn string (default is GOLDILOCKS)
    --conn-string  'CONN-STRING'  connection string
    --prompt       STRING         change prompt string
    --enable-color                enable colored text
    --as           {SYSDBA|ADMIN} privilege
    --silent                      suppresses the display of the result message and echoing of
commands
    --help                        print help message
%
```

--import

Description

It performs the SQL statement which is not an interactive mode in the file in batch.
The file name is enclosed in a single quote ('), and it can use an absolute or relative path.

Examples

The following is an example of executing the SQL statement by using the absolute path of the file.

```
% gsql test test --import '/home/goldilocks/sample.sql'
DROP TABLE IF EXISTS t1;
Table dropped.
CREATE TABLE t1
(
    id   INTEGER PRIMARY KEY,
    name VARCHAR(128),
    addr VARCHAR(128)
);
Table created.
CREATE INDEX t1_idx_name ON t1(name);
Index created.
INSERT INTO t1
       VALUES ( 1, 'leekmo', 'Seoul, Korea'  ),
              ( 2, 'mkkim' , 'Seoul, Korea'  ),
              ( 3, 'xcom73', 'Inchon, Korea' );
3 rows created.
COMMIT;
Commit complete.
%
```

The following is an example of executing the SQL statement by using the relative path of the file.

```
% gsql test test --import 'sample.sql'
DROP TABLE IF EXISTS t1;
Table dropped.
CREATE TABLE t1
(
    id   INTEGER PRIMARY KEY,
```

```
    name VARCHAR(128),
    addr VARCHAR(128)
);
Table created.
CREATE INDEX t1_idx_name ON t1(name);
Index created.
INSERT INTO t1
       VALUES ( 1, 'leekmo', 'Seoul, Korea'  ),
              ( 2, 'mkkim' , 'Seoul, Korea'  ),
              ( 3, 'xcom73', 'Inchon, Korea' );
3 rows created.
COMMIT;
Commit complete.
%
```

--no-prompt

Description

It does not output the prompt when performing an interactive mode.

Example

The following is an example of using --no-prompt.

```
% gsql test test --no-prompt
SELECT * FROM dual;
DUMMY
-----
X
1 row selected.
```

--prompt

Description

It sets the gsql prompt when performing an interactive mode.
The default value is gSQL.
The special characters or spaces are enclosed in double quotes (") as follows.

```
% gsql test test --prompt "GOLDILOCKS Venus.3.2"
```

Example

The following is an example of which the prompt is changed to GOLDILOCKS in an interactive mode by u sing the --prompt option.

```
% gsql test test --prompt GOLDILOCKS
Connected to GOLDILOCKS Database.
GOLDILOCKS> SELECT * FROM dual;
DUMMY
-----
X
1 row selected.
GOLDILOCKS>
```

--silent

Description

It does not output the results of SQL statement.
It is useful when executing large amounts of SQL statements by reading the file with **--import** command.
The --silent option is also applied when operating in an interactive mode.

Example

The following is an example of executing the DictionarySchema.sql file in the $GOLDILOCKS_HOME/adm in directory by using the --silent option.

```
% gsql sys gliese --as sysdba --import 'DictionarySchema.sql' --silent
%
```

--version

Description

It outputs the version information of the gsql program.
It is recommended to use the gsql program with the version as same as the version of GOLDILOCKS. The GOLDILOCKS version can be viewed via the SQL function **VERSION** as follows.

```
gSQL> SELECT version() FROM dual;
VERSION()
-----------------------------------
Release Name.X.X.X revision(XXXXX)
1 row selected.
```

Example

The following is an example of using the --version option.

```
% gsql --version
%
```

40.5 Interactive Command References

All gsql-specific commands start with backslash (\) in an interactive mode.

\\

Syntax

```
\\
```

Description

It executes the most recent successfully executed SQL statement.

Examples

```
gSQL> SELECT * FROM dual;
DUMMY
-----
X
1 row selected.
```

- It executes the most recent successfully executed SQL statement.

```
gSQL> \\
DUMMY
-----
X
1 row selected.
gSQL> SELECT * FROM invalid_table;
ERR-42000(16040): table or view does not exist :
SELECT * FROM invalid_table
              *
ERROR at line 1:
```

- It executes the most recent successfully executed SQL statement rather than executing the failed SQL s tatement.

```
gSQL> \\
DUMMY
-----
X
1 row selected.
```

\connect

Syntax

```
\connect username password [as sysdba]
```

Description

It is newly connected with the entered username and password.
If any uncommitted transaction exists, it performs COMMIT.

Examples

The following is an example of connecting with the test account.

```
gSQL> \connect test test
gSQL>
```

An error occurs if the password is invalid as follows.

```
gSQL> \connect test invalid_password
ERR-28000(16004): invalid username/password; logon denied
gSQL> SELECT * FROM dual;
ERR-08003(40044): connection does not exist
```

The following is an example of connecting with the SYSDBA role.

```
gSQL> \connect sys gliese as sysdba
gSQL>
```

\cstartup

Syntax

```
\cstartup
\cstartup nomount
\cstartup mount
\cstartup open
\cstartup local open
\cstartup global open
```

Description

It starts up GOLDILOCKS server in a cluster environment.

To execute \cstartup command, it should connect with SYSDBA role or ADMIN role. For more informatio n about how to connect with SYSDBA role, refer to **startup and shutdown server**.

- \cstartup nomount
  - It starts up the corresponding server on NOMOUNT phase.
- \cstartup mount
  - It starts up the corresponding server on MOUNT phase.
- \cstartup local open
  - It starts up the corresponding server and other servers on LOCAL OPEN phase.
- \cstartup global open
  - It startsup all servers on GLOBAL OPEN phase.
- \cstartup open
  - It is as same as \cstartup global open.
- \cstartup
  - It is as same as \cstartup global open.

\cstartup local open command and \cstartup global open command startup the corresponding server and other servers to the corresponding phase. Other commands are applied only to the corresponding se rver, so the server should be started up to LOCAL OPEN phase for the multi-level startup using **ALTER SYS TEM {MOUNT | OPEN} DATABASE**.

The server startup phases are classified as NOMOUNT, MOUNT, LOCAL OPEN, GLOBAL OPEN. For more i nformation, refer to **multi-level startup**.

ALTER SYSTEM {MOUNT | OPEN} DATABASE should be performed to move on to the next phase after ex ecuting \cstartup command.

\cstartup using glocator can be viewed via **CSTARTUP and CSHUTDOWN**.

- It is command used only in a C/S environment. It can be used only in gsqlnet.
- Commands other than \cstartup nomount/ mount affects on the corresponding server and o ther servers. Therefore, if LOCATOR_DSN is not specified in **odbc.ini file**, then an error occur s.

Example

The following is an example of starting up GOLDILOCKS.

```
% gsqlnet sys gliese --as sysdba
Connected to an idle instance.
gSQL> \cstartup
Startup success
```

\cshutdown

Syntax

```
\cshutdown
\cshutdown abort
\cshutdown immediate
\cshutdown transactional
\cshutdown normal
```

Description

It shuts down GOLDILOCKS server in a cluster environment.

To execute \cshutdown command, it should connect with SYSDBA role or ADMIN role. For more informati on about how to connect with SYSDBA role, refer to **startup and shutdown server**.

- \cshutdown normal
  - It blocks the connection from the new session and waits for all currently connected sessions to be terminated, then performs the checkpoint and shuts down the server.
- \cshutdown transactional
  - It blocks the start of a new transaction and waits for all currently running transactions to be termi nated, then performs the checkpoint and shuts down the server.

- \cshutdown immediate
  - It blocks the execution of a new unit operation (e.g. FETCH, EXECUTE), waits for all currently running unit operations to be terminated, then rolls back all transactions and performs the checkpoint and shuts down the server.
- \cshutdown abort
  - The server is forcibly shut down immediately regardless of the status of the currently connected session.
- \cshutdown
  - It is as same as \cshutdown normal.

\cshutdown using glocator can be viewed via **CSTARTUP and CSHUTDOWN**.

- It is command used only in a C/S environment. It can be used only in gsqlnet.
- Commands other than \cshutdown affects on the corresponding server and other servers. Therefore, if LOCATOR_DSN is not specified in **odbc.ini file**, then an error occurs.

Example

The following is an example of shutting down GOLDILOCKS.

```
% gsqlnet sys gliese --as sysdba
Connected to GOLDILOCKS Database.
gSQL> \cshutdown
Shutdown success
gSQL>
```

\ddl_cluster

Syntax

```
\ddl_cluster
```

Description

It outputs a cluster DDL corresponding to the current status of a cluster system.
This statement is valid in a cluster system.

Example

The following is an example of executing \ddl_cluster command for a cluster system consisting as 3 x 2.

```
gSQL> \ddl_cluster
SET SESSION AUTHORIZATION "SYS";
CREATE CLUSTER GROUP "G1"
       CLUSTER MEMBER "G1N1" HOST '127.0.0.1' PORT '10110'
;
COMMIT;
SET SESSION AUTHORIZATION "SYS";
ALTER CLUSTER GROUP "G1" ADD
      CLUSTER MEMBER "G1N2" HOST '127.0.0.1' PORT '10120'
;
COMMIT;
SET SESSION AUTHORIZATION "SYS";
CREATE CLUSTER GROUP "G2"
       CLUSTER MEMBER "G2N1" HOST '127.0.0.1' PORT '10210'
;
COMMIT;
SET SESSION AUTHORIZATION "SYS";
ALTER CLUSTER GROUP "G2" ADD
      CLUSTER MEMBER "G2N2" HOST '127.0.0.1' PORT '10220'
;
COMMIT;
SET SESSION AUTHORIZATION "SYS";
CREATE CLUSTER GROUP "G3"
       CLUSTER MEMBER "G3N1" HOST '127.0.0.1' PORT '10310'
;
COMMIT;
SET SESSION AUTHORIZATION "SYS";
ALTER CLUSTER GROUP "G3" ADD
      CLUSTER MEMBER "G3N2" HOST '127.0.0.1' PORT '10320'
;
```

\ddl_db

Syntax

```
\ddl_db
\ddl_db GRANT
\ddl_db COMMENT
```

Description

It outputs the DDL statements corresponding to the current state of the database object.

The objects stored in the recycle bin are not output.

Table 40-19 `\ddl_db` commands

Command	Description
`\ddl_db`	It outputs the DDL statements of all objects created in the database.
`\ddl_db GRANT`	It outputs the GRANT .. ON DATABASE statement corresponding to the privilege information on the database object.
`\ddl_db COMMENT`	It outputs the COMMENT ON DATABASE statement corresponding to the comment information on the database object.

The result of `\ddl_db` command without any option outputs the DDL statements in the following order.

- Database DDL
- Tablespace DDL
- Profile DDL
- Authorization DDL
- Schema DDL
- Authorization schema path DDL
- Database privilege DDL
- Tablespace privilege DDL
- Schema privilege DDL
- Public synonym DDL
- Table DDL
- Table privilege DDL
- Table option DDL
- Constraint DDL
- Index DDL
- View DDL
- View privilege DDL
- Sequence DDL

- Sequence privilege DDL
- Synonym DDL
- Stored procedure/ function DDL
- Package DDL
- Audit policy DDL

However, the information related to the following schemas which include the dictionary information are not output.

- DICTIONARY_SCHEMA
- INFORMATION_SCHEMA
- PERFORMANCE_VIEW_SCHEMA
- DEFINITION_SCHEMA
- FIXED_TABLE_SCHEMA

The DDL statement for the schema above which includes the dictionary information can be output by using **`\ddl_schema`**.

Examples

The following is an example of executing `\ddl_db`.

```
gSQL> \ddl_db
```

- Database DDL

```
SET SESSION AUTHORIZATION "SYS";
COMMENT
    ON DATABASE
    IS 'goldilocks database'
;
```

- Tablespace DDL

```
SET SESSION AUTHORIZATION "SYS";
CREATE MEMORY DATA TABLESPACE "TEST_TBS"
    DATAFILE '/home/GOLDILOCKS/workspace/product/Gliese/home/db/test1.dbf'
        SIZE 10477568 REUSE
    ONLINE
    LOGGING
    EXTSIZE 262144
;
```

```
SET SESSION AUTHORIZATION "SYS";
ALTER TABLESPACE "TEST_TBS"
    ADD DATAFILE '/home/GOLDILOCKS/workspace/product/Gliese/home/db/test2.dbf'
        SIZE 10477568 REUSE
;
SET SESSION AUTHORIZATION "SYS";
ALTER TABLESPACE "TEST_TBS"
    ADD DATAFILE '/home/GOLDILOCKS/workspace/product/Gliese/home/db/test3.dbf'
        SIZE 10477568 REUSE
;
SET SESSION AUTHORIZATION "SYS";
COMMENT
    ON TABLESPACE "TEST_TBS"
    IS 'comment tablespace TPC data'
;
SET SESSION AUTHORIZATION "SYS";
CREATE MEMORY TEMPORARY TABLESPACE "TEMP_TBS"
    MEMORY 'test_mem'
        SIZE 10477568
    EXTSIZE 262144
;
... Ellipsis ...
```

- Sequence privilege DDL

```
SET SESSION AUTHORIZATION "H_USER";
GRANT
    USAGE ON SEQUENCE "H_USER"."H_SEQ"
    TO "PUBLIC"
;
SET SESSION AUTHORIZATION "H_USER";
GRANT
    USAGE ON SEQUENCE "H_USER"."H_SEQ"
    TO "TEST"
    WITH GRANT OPTION
;
SET SESSION AUTHORIZATION "H_USER";
GRANT
    USAGE ON SEQUENCE "H_USER"."H_SEQ"
    TO "C_USER"
;
```

```
SET SESSION AUTHORIZATION "C_USER";
GRANT
    USAGE ON SEQUENCE "C_USER"."C_SEQ"
    TO "TEST"
;
SET SESSION AUTHORIZATION "C_USER";
GRANT
    USAGE ON SEQUENCE "C_USER"."C_SEQ"
    TO "H_USER"
;
```

The following is an example of executing `\ddl_db GRANT`.

```
gSQL> \ddl_db GRANT
SET SESSION AUTHORIZATION "SYS";
GRANT
    ALTER DATABASE ON DATABASE
    TO "TEST"
    WITH GRANT OPTION
;
SET SESSION AUTHORIZATION "SYS";
GRANT
    ALTER SYSTEM ON DATABASE
    TO "TEST"
    WITH GRANT OPTION
;
... Ellipsis ...
```

\ddl_tablespace

Syntax

```
\ddl_tablespace name
\ddl_tablespace name CREATE
\ddl_tablespace name ALTER
\ddl_tablespace name TABLE
\ddl_tablespace name CONSTRAINT
\ddl_tablespace name INDEX
\ddl_tablespace name GRANT
```

```
\ddl_tablespace name COMMENT
```

Description

It outputs the DDL statements for the current state of the tablespace object.


The objects stored in the recycle bin are not output.

Table 40-20 \ddl_tablespace commands

Command	Description
\ddl_tablespace name	It performs all options below.
\ddl_tablespace name CREATE	It outputs the CREATE TABLESPACE statement of the tablespace object.
\ddl_tablespace name ALTER	It outputs the ALTER TABLESPACE statements for the datafile or memory added to the tablespace.
\ddl_tablespace name TABLE	It outputs the CREATE TABLE statements of the tables stored in the tablespace.
\ddl_tablespace name CONSTRAINT	It outputs the ALTER TABLE statements of the constraints stored in the tablespace.
\ddl_tablespace name INDEX	It outputs the CREATE INDEX statements of the indexes stored in the tablespace.
\ddl_tablespace name GRANT	It outputs the GRANT .. ON TABLESPACE statement for the privilege information on the tablespace.
\ddl_tablespace name COMMENT	It outputs the COMMENT ON TABLESPACE statement corresponding o the comment on the tablespace.

Examples

The following is an example of executing \ddl_tablespace CREATE.

```
gSQL> \ddl_tablespace test_tbs CREATE
SET SESSION AUTHORIZATION "SYS";
CREATE MEMORY DATA TABLESPACE "TEST_TBS"
    DATAFILE '/home/GOLDILOCKS/workspace/product/Gliese/home/db/test1.dbf'
        SIZE 10477568 REUSE
    ONLINE
    EXTSIZE 262144
;
```

The following is an example of executing \ddl_tablespace ALTER.

```
gSQL> \ddl_tablespace test_tbs ALTER
SET SESSION AUTHORIZATION "SYS";
```

```
ALTER TABLESPACE "TEST_TBS"
    ADD DATAFILE
        '/home/GOLDILOCKS/workspace/product/Gliese/home/db/test2.dbf'
        SIZE 10477568 REUSE
      ,
        '/home/GOLDILOCKS/workspace/product/Gliese/home/db/test3.dbf'
        SIZE 10477568 REUSE
;
```

\ddl_profile

Syntax

```
\ddl_profile name
\ddl_profile name CREATE
\ddl_profile name COMMENT
```

Description

It outputs the DDL statement for the current state of the profile object.

Table 40-21 `\ddl_profile` commands

Command	Description
`\ddl_profile name`	It performs all options below.
`\ddl_profile name CREATE`	It outputs the CREATE PROFILE statement of the profile object.
`\ddl_profile name COMMENT`	It outputs the COMMENT ON PROFILE statement corresponding to the comment on the profile.

Example

The following is an example of executing `\ddl_profile CREATE`.

```
gSQL> \ddl_profile prof1 CREATE
SET SESSION AUTHORIZATION "SYS";
CREATE PROFILE "PROF1" LIMIT
    FAILED_LOGIN_ATTEMPTS   DEFAULT
    PASSWORD_LOCK_TIME   1/86400
    PASSWORD_LIFE_TIME   UNLIMITED
    PASSWORD_GRACE_TIME   100
```

```
    PASSWORD_REUSE_MAX    DEFAULT
    PASSWORD_REUSE_TIME    DEFAULT
    PASSWORD_VERIFY_FUNCTION    KISA_VERIFY_FUNCTION
;
COMMIT;
```

\ddl_audit_policy

Syntax

```
\ddl_audit_policy name
\ddl_audit_policy name CREATE
\ddl_audit_policy name AUDIT
\ddl_audit_policy name COMMENT
```

Description

It outputs the DDL statement for the current state of the audit policy object.

The objects stored in the recycle bin are not output.

Table 40-22 \ddl_audit_policy commands

Command	Description
\ddl_audit_policy name	It perform all options below.
\ddl_audit_policy name CREATE	It outputs CREATE AUDIT POLICY statement of the audit policy object.
\ddl_audit_policy name AUDIT	It outputs AUDIT POLICY statement of the audit policy object.
\ddl_audit_policy name COMMENT	It outputs the COMMENT ON AUDIT POLICY statement corresponding to the comment on the audit policy.

Examples

The following is an example of executing \ddl_audit_policy CREATE.

```
gSQL> \ddl_audit_policy p1 CREATE
SET SESSION AUTHORIZATION "SYS";
CREATE AUDIT POLICY "P1"
```

```
       ACTIONS SELECT ON "PUBLIC"."T1"
             , INSERT ON "PUBLIC"."T1"
             , UPDATE ON "PUBLIC"."T2"
             , ALL ON "PUBLIC"."SEQ1"
             , SELECT ON "PUBLIC"."SEQ2"
             , EXECUTE ON "PUBLIC"."FUNC2"
;
COMMIT;
```

\ddl_auth

Syntax

```
\ddl_auth name
\ddl_auth name CREATE
\ddl_auth name SCHEMA PATH
\ddl_auth name SCHEMA
\ddl_auth name TABLE
\ddl_auth name CONSTRAINT
\ddl_auth name INDEX
\ddl_auth name VIEW
\ddl_auth name SEQUENCE
\ddl_auth name SYNONYM
\ddl_auth name PROCEDURE
\ddl_auth name PACKAGE
\ddl_auth name COMMENT
```

Description

It outputs the DDL statements for the current state of the account object.

The objects stored in the recycle bin are not output.

Table 40-23 \ddl_auth commands

Command	Description
\ddl_auth name	It performs all options below.
\ddl_auth name CREATE	It outputs the CREATE USER statement of the user object.
\ddl_auth name SCHEMA	It outputs the CREATE SCHEMA statement of schemas owned by a user.

Command	Description
\ddl_auth name SCHEMA PATH	It outputs the ALTER USER statement corresponding to the schema path of the account.
\ddl_auth name TABLE	It outputs the CREATE TABLE statements of tables owned by a user.
\ddl_auth name CONSTRAINT	It outputs the ALTER TABLE statements of constraints owned by a user.
\ddl_auth name INDEX	It outputs the CREATE INDEX statements of indexes owned by a user.
\ddl_auth name VIEW	It outputs the CREATE VIEW statements of views owned by a user.
\ddl_auth name SEQUENCE	It outputs the CREATE SEQUENCE statements of sequences owned by a user.
\ddl_auth name SYNONYM	It outputs the CREATE SYNONYM statements of sequences owned by a user.
\ddl_auth name PROCEDURE	It outputs the CREATE PROCEDURE/ FUNCTION statements of sequences owned by a user.
\ddl_auth name PACKAGE	It outputs the CREATE PACKAGE/PACKAGE BODY statements of packages owned by a user.
\ddl_auth name COMMENT	It outputs the COMMENT ON AUTHORIZATION statement corresponding to comments on the account.

Examples

The following is an example of executing \ddl_auth CREATE.

```
gSQL> \ddl_auth h_user CREATE
SET SESSION AUTHORIZATION "SYS";
CREATE USER "H_USER"
    IDENTIFIED BY H_USER
    DEFAULT TABLESPACE "TEST_TBS"
    TEMPORARY TABLESPACE "TEMP_TBS"
    WITHOUT SCHEMA
;
```

The following is an example of executing \ddl_auth SCHEMA.

```
gSQL> \ddl_auth h_user SCHEMA
SET SESSION AUTHORIZATION "SYS";
CREATE SCHEMA "H_USER"
    AUTHORIZATION "H_USER"
;
```

The following is an example of executing \ddl_auth TABLE.

```
gSQL> \ddl_auth h_user TABLE
SET SESSION AUTHORIZATION "H_USER";
```

```
CREATE TABLE "H_USER"."REGION"
    (
        "R_REGIONKEY" NUMBER( 10, 0 )
      , "R_NAME" CHARACTER( 25 OCTETS )
      , "R_COMMENT" CHARACTER VARYING( 152 OCTETS )
    )
    PCTFREE  10
    PCTUSED  60
    INITRANS 4
    MAXTRANS 8
    STORAGE
    (
        INITIAL 524288
        NEXT    262144
        MINSIZE 524288
        MAXSIZE 35184372088832
    )
    TABLESPACE "TEST_TBS"
;
SET SESSION AUTHORIZATION "H_USER";
CREATE TABLE "H_USER"."NATION"
    (
        "N_NATIONKEY" NUMBER( 10, 0 )
      , "N_NAME" CHARACTER( 25 OCTETS )
      , "N_REGIONKEY" NUMBER( 10, 0 )
      , "N_COMMENT" CHARACTER VARYING( 152 OCTETS )
    )
    PCTFREE  10
    PCTUSED  60
    INITRANS 4
    MAXTRANS 8
    STORAGE
    (
        INITIAL 524288
        NEXT    262144
        MINSIZE 524288
        MAXSIZE 35184372088832
    )
    TABLESPACE "TEST_TBS"
;
```

\ddl_schema

Syntax

```
\ddl_schema name
\ddl_schema name CREATE
\ddl_schema name TABLE
\ddl_schema name CONSTRAINT
\ddl_schema name INDEX
\ddl_schema name VIEW
\ddl_schema name SEQUENCE
\ddl_schema name SYNONYM
\ddl_schema name PROCEDURE
\ddl_schema name PACKAGE
\ddl_schema name GRANT
\ddl_schema name COMMENT
```

Description

It outputs the DDL statements for the current state of the schema object.

The objects stored in the recycle bin are not output.

Table 40-24 \ddl_schema commands

Command	Description
\ddl_schema name	It performs all options below.
\ddl_schema name CREATE	It outputs the CREATE SCHEMA statement of the schema object.
\ddl_schema name TABLE	It outputs the CREATE TABLE statements of the tables which belong to the schema.
\ddl_schema name CONSTRAINT	It outputs the ALTER TABLE statements of the constraints which belong to the schema.
\ddl_schema name INDEX	It outputs the CREATE INDEX statements of the indexes which belong to the schema.
\ddl_schema name VIEW	It outputs the CREATE VIEW statements of the views which belong to the schema.
\ddl_schema name SEQUENCE	It outputs the CREATE SEQUENCE statements of the sequences which belong to the schema.
\ddl_schema name SYNONYM	It outputs the CREATE SYNONYM statements of the sequences which belong to the schema.
\ddl_schema name PROCEDURE	It outputs the CREATE PROCEDURE/ FUNCTION statements of the sequences which belong to the schema.
\ddl_schema name PACKAGE	It outputs the CREATE PACKAGE/PACKAGE BODY statements of the package which

Command	Description
	belong to the schema.
\ddl_schema name GRANT	It outputs the GRANT .. ON SCHEMA statement for the privilege information on the schema.
\ddl_schema name COMMENT	It outputs the COMMENT ON SCHEMA statement corresponding to the comment on the schema.

Examples

The following is an example of executing \ddl_schema CREATE.

```
gSQL> \ddl_schema h_user CREATE
SET SESSION AUTHORIZATION "SYS";
CREATE SCHEMA "H_USER"
    AUTHORIZATION "H_USER"
;
```

The following is an example of executing \ddl_schema CONSTRAINT.

```
gSQL> \ddl_schema h_user CONSTRAINT
SET SESSION AUTHORIZATION "H_USER";
ALTER TABLE "H_USER"."REGION"
    ADD CONSTRAINT "H_USER"."REGION_PK"
    PRIMARY KEY
    (
        "R_REGIONKEY" ASC NULLS LAST
    )
    INDEX "REGION_PK_INDEX"
    PCTFREE  10
    INITRANS 4
    MAXTRANS 8
    STORAGE
    (
        INITIAL 524288
        NEXT    262144
        MINSIZE 524288
        MAXSIZE 35184372088832
    )
    TABLESPACE "TEMP_TBS"
    NOT DEFERRABLE
    INITIALLY IMMEDIATE
```

```
;
SET SESSION AUTHORIZATION "H_USER";
ALTER TABLE "H_USER"."NATION"
    ADD CONSTRAINT "H_USER"."NATION_PK"
    PRIMARY KEY
    (
        "N_NATIONKEY" ASC NULLS LAST
    )
    INDEX "NATION_PK_INDEX"
    PCTFREE  10
    INITRANS 4
    MAXTRANS 8
    STORAGE
    (
        INITIAL 524288
        NEXT    262144
        MINSIZE 524288
        MAXSIZE 35184372088832
    )
    TABLESPACE "TEMP_TBS"
    NOT DEFERRABLE
    INITIALLY IMMEDIATE
;
SET SESSION AUTHORIZATION "H_USER";
ALTER TABLE "H_USER"."SUPPLIER"
    ADD CONSTRAINT "H_USER"."SUPPLIER_PK"
    PRIMARY KEY
    (
        "S_SUPPKEY" ASC NULLS LAST
    )
    INDEX "SUPPLIER_PK_INDEX"
    PCTFREE  10
    INITRANS 4
    MAXTRANS 8
    STORAGE
    (
        INITIAL 524288
        NEXT    262144
        MINSIZE 524288
        MAXSIZE 35184372088832
    )
```

```
    TABLESPACE "TEMP_TBS"
    NOT DEFERRABLE
    INITIALLY IMMEDIATE
;
```

\ddl_public_synonym

Syntax

```
\ddl_public_synonym name
\ddl_public_synonym name CREATE
```

Description

It outputs the DDL statements for the current state of the public synonym object.

Table 40-25 `\ddl_public_synonym` commands

Command	Description
`\ddl_public_synonym`	It performs all options below.
`\ddl_public_synonym name CREATE`	It outputs the CREATE PUBLIC SYNONYM statement of the public synonym object.

Example

The following is an example of executing `\ddl_public_synonym CREATE`.

```
gSQL> \ddl_public_synonym pubsyn CREATE
SET SESSION AUTHORIZATION "SYS";
CREATE PUBLIC SYNONYM "PUBSYN" FOR "PUBLIC"."T1"
;
COMMIT;
```

\ddl_table

Syntax

```
\ddl_table name
\ddl_table name CREATE
\ddl_table name CONSTRAINT
\ddl_table name INDEX
\ddl_table name IDENTITY
\ddl_table name SUPPLEMENTAL
\ddl_table name GRANT
\ddl_table name COMMENT
```

Description

It outputs the DDL statements for the current state of the table object.

The objects stored in the recycle bin are not output.

Table 40-26 `\ddl_table` commands

Command	Description
`\ddl_table name`	It performs all options below.
`\ddl_table name CREATE`	It outputs the CREATE TABLE statement of the table object.
`\ddl_table name CONSTRAINT`	It outputs the ALTER TABLE statements of the constraints created in the table.
`\ddl_table name INDEX`	It outputs the CREATE INDEX statements of the indexes created in the table.
`\ddl_table name IDENTITY`	It outputs the ALTER TABLE statement for the restart value if the table has an identity column.
`\ddl_table name SUPPLEMENTAL`	It outputs the ALTER TABLE statement if the supplemental log option is set on the table.
`\ddl_table name GRANT`	It outputs the GRANT .. ON TABLE statement for the privilege information on the table.
`\ddl_table name COMMENT`	It outputs the COMMENT ON TABLE statement corresponding to the comment on the table.

Examples

The following is an example of executing `\ddl_table CREATE`.

```
gSQL> \ddl_table h_user.orders CREATE
SET SESSION AUTHORIZATION "H_USER";
CREATE TABLE "H_USER"."ORDERS"
    (
```

```
        "O_ORDERKEY" NUMBER( 10, 0 )
      , "O_CUSTKEY" NUMBER( 10, 0 )
      , "O_ORDERSTATUS" CHARACTER( 1 OCTETS )
      , "O_TOTALPRICE" NUMBER( 12, 2 )
      , "O_ORDERDATE" DATE
      , "O_ORDERPRIORITY" CHARACTER( 15 OCTETS )
      , "O_CLERK" CHARACTER( 15 OCTETS )
      , "O_SHIPPRIORITY" NUMBER( 10, 0 )
      , "O_COMMENT" CHARACTER VARYING( 79 OCTETS )
    )
    PCTFREE  10
    PCTUSED  60
    INITRANS 4
    MAXTRANS 8
    STORAGE
    (
        INITIAL 524288
        NEXT    262144
        MINSIZE 524288
        MAXSIZE 35184372088832
    )
    TABLESPACE "TEST_TBS"
;
```

The following is an example of executing `\ddl_table GRANT`.

```
gSQL> \ddl_table h_user.nation GRANT
SET SESSION AUTHORIZATION "H_USER";
GRANT
    DELETE ON TABLE "H_USER"."NATION"
    TO "TEST"
;
SET SESSION AUTHORIZATION "H_USER";
GRANT
    SELECT ( "N_NATIONKEY" ) ON TABLE "H_USER"."NATION"
    TO "C_USER"
;
SET SESSION AUTHORIZATION "H_USER";
GRANT
    SELECT ( "N_NAME" ) ON TABLE "H_USER"."NATION"
    TO "C_USER"
```

```
;
```

\ddl_constraint

Syntax

```
\ddl_constraint name
\ddl_constraint name ALTER
\ddl_constraint name COMMENT
```

Description

It outputs the DDL statements for the current state of the constraint object.

The objects stored in the recycle bin are not output.

Table 40-27 `\ddl_constraint` commands

Command	Description
`\ddl_constraint name`	It performs all options below.
`\ddl_constraint name ALTER`	It outputs the ALTER TABLE statement of the constraint object.
`\ddl_constraint name COMMENT`	It outputs the COMMENT ON CONSTRAINT statement corresponding to the comment on the constraint.

Example

The following is an example of executing `\ddl_constraint ALTER`.

```
gSQL> \ddl_constraint h_user.lineitem_pk ALTER
SET SESSION AUTHORIZATION "H_USER";
ALTER TABLE "H_USER"."LINEITEM"
    ADD CONSTRAINT "H_USER"."LINEITEM_PK"
    PRIMARY KEY
    (
        "L_ORDERKEY" ASC NULLS LAST
      , "L_LINENUMBER" ASC NULLS LAST
    )
    INDEX "LINEITEM_PK_INDEX"
    PCTFREE  10
```

```
    INITRANS 4
    MAXTRANS 8
    STORAGE
    (
        INITIAL 524288
        NEXT    262144
        MINSIZE 524288
        MAXSIZE 35184372088832
    )
    TABLESPACE "TEMP_TBS"
    NOT DEFERRABLE
    INITIALLY IMMEDIATE
;
```

\ddl_index

Syntax

```
\ddl_index name
\ddl_index name CREATE
\ddl_index name COMMENT
```

Description

It outputs the DDL statements for the current state of the index object.

The objects stored in the recycle bin are not output.

Table 40-28 \ddl_index commands

Command	Description
\ddl_index name	It performs all options below.
\ddl_index name CREATE	It outputs the CREATE INDEX statement of the index object.
\ddl_index name COMMENT	It outputs the COMMENT ON INDEX statement corresponding to the comment on the index.

Example

The following is an example of executing `\ddl_index CREATE`.

```
gSQL> \ddl_index public.idx2 CREATE
SET SESSION AUTHORIZATION "TEST";
CREATE INDEX "PUBLIC"."IDX2"
    ON "PUBLIC"."T1"
    (
        "C3" ASC NULLS LAST
      , "C1" DESC NULLS FIRST
    )
    PCTFREE  10
    INITRANS 4
    MAXTRANS 8
    STORAGE
    (
        INITIAL 524288
        NEXT    262144
        MINSIZE 524288
        MAXSIZE 35184372088832
    )
    TABLESPACE "MEM_TEMP_TBS"
;
```

\ddl_view

Syntax

```
\ddl_view name
\ddl_view name CREATE
\ddl_view name GRANT
\ddl_view name COMMENT
```

Description

It outputs the DDL statements for the current state of the view object.

Table 40-29 \ddl_view commands

Command	Description
\ddl_view name	It performs all options below.
\ddl_view name CREATE	It outputs the CREATE VIEW statement of the view object.
\ddl_view name GRANT	It outputs the GRANT .. ON TABLE statement for the privilege information on the view.
\ddl_view name COMMENT	It outputs the COMMENT ON TABLE corresponding to the comment on the view.

Example

The following is an example of executing \ddl_view CREATE.

```
gSQL> \ddl_view h_user.revenue CREATE
SET SESSION AUTHORIZATION "H_USER";
CREATE OR REPLACE FORCE VIEW "H_USER"."REVENUE"
    (supplier_no, total_revenue)
    AS SELECT
    l_suppkey,
    ROUND( sum(l_extendedprice * (1 - l_discount)), 2)
FROM
    lineitem
WHERE
      l_shipdate >= date '1996-01-01'
  AND l_shipdate < date '1996-01-01' + interval '3' month
GROUP BY
    l_suppkey
;
```

\ddl_sequence

Syntax

```
\ddl_sequence name
\ddl_sequence name CREATE
\ddl_sequence name RESTART
\ddl_sequence name GRANT
\ddl_sequence name COMMENT
```

Description

It outputs the DDL statements for the current state of the sequence object.

Table 40-30 \ddl_sequence commands

Command	Description
\ddl_sequence name	It performs all options below.
\ddl_sequence name CREATE	It outputs the CREATE SEQUENCE statement of the sequence object.
\ddl_sequence name RESTART	It outputs the ALTER SEQUENCE statement corresponding to the restart value of the sequence object.
\ddl_sequence name GRANT	It outputs the GRANT .. ON SEQUENCE statement for the privilege information on t he sequence.
\ddl_sequence name COMMENT	It outputs the COMMENT ON SEQUENCE statement corresponding to the comment on the sequence.

Examples

The following is an example of executing \ddl_sequence CREATE.

```
gSQL> \ddl_sequence h_user.h_seq CREATE
SET SESSION AUTHORIZATION "H_USER";
CREATE SEQUENCE "H_USER"."H_SEQ"
    START WITH 1
    INCREMENT BY 1
    MAXVALUE 9223372036854775807
    MINVALUE 1
    NO CYCLE
    CACHE 20
;
```

The following is an example of executing \ddl_sequence RESTART.

```
gSQL> \ddl_sequence h_user.h_seq RESTART
SET SESSION AUTHORIZATION "H_USER";
ALTER SEQUENCE "H_USER"."H_SEQ"
    RESTART WITH 21
;
```

\ddl_synonym

Syntax

```
\ddl_synonym name
\ddl_synonym name CREATE
```

Description

It outputs the DDL statements for the current state of the synonym object.

Table 40-31 `\ddl_synonym` commands

Command	Description
`\ddl_synonym`	It performs all options below.
`\ddl_synonym name CREATE`	It outputs the CREATE SYNONYM statement of the synonym object.

Example

The following is an example of executing `\ddl_synonym CREATE`.

```
gSQL> \ddl_synonym syn CREATE
SET SESSION AUTHORIZATION "TEST";
CREATE SYNONYM "PUBLIC"."SYN" FOR "PUBLIC"."T1"
;
COMMIT;
```

\ddl_procedure

Syntax

```
\ddl_procedure name
\ddl_procedure name CREATE
```

Description

It outputs the DDL statements for the current state of the stored procedure or the function object.

Table 40-32 \ddl_procedure commands

Command	Description
\ddl_procedure	It performs all options below.
\ddl_procedure name CREATE	It outputs CREATE PROCEDURE/ FUNCTION statement of the stored procedure/ function object.

Example

The following is an example of executing \ddl_procedure CREATE.

```
gSQL> \ddl_procedure proc1 CREATE
SET SESSION AUTHORIZATION "TEST";
CREATE OR REPLACE PROCEDURE "PUBLIC"."PROC1"
is
begin
 null;
end;
/
COMMIT;
```

\ddl_package

Syntax

```
\ddl_package name
\ddl_package name CREATE
```

Description

It outputs the DDL statements for the current state of the package spec or the package body object.

Table 40-33 \ddl_package commands

Command	Description
\ddl_package	It performs all options below.
\ddl_package name CREATE	It outputs CREATE PACKAGE/PACKAGE BODY statement of the package spec and the package body object.

Examples

The following is an example of executing `\ddl_package CREATE`.

```
gSQL> \ddl_package pkg1 CREATE
SET SESSION AUTHORIZATION "TEST";
CREATE OR REPLACE PACKAGE "PUBLIC"."PKG1"
is
  v1 integer;
  procedure proc1( a1 integer );
 end;
/
CREATE OR REPLACE PACKAGE BODY "PUBLIC"."PKG1"
is
  procedure proc1( a1 integer )
  is
  begin
    null;
  end;
end;
/
COMMIT;
```

\desc

Syntax

```
\desc table_name
\desc schema_name.table_name
```

Description

It queries the structure information of the table.

A table may be described only with the table name as follows or described together with the schema na me. If the schema name is not specified, the schema name of the table is determined by the **Schema Path** of the user.

- \desc t1
- \desc public.t1

If a table name is created by using **Identifiers** when creating a table as follows, the double quotes (") is used to describe it.

- Creating a table
  - CREATE TABLE "TaBle#@^*" (id INTEGER);
- \desc "TaBle#@^*"
- \desc "PUBLIC"."TaBle#@^*"

The execution result includes the following information of the table.

- Column information
  - Column name
  - Data type
  - Whether NULL is allowed
- Index information
  - Index name
  - Storage space for index
  - Index type
  - Whether UNIQUE is allowed
  - Key column name
- Constraint information
  - Constraint name
  - Constraint type
  - The related index
  - Constraint column

Examples

The following is an example of querying the information of the t1 table.

```
gSQL> \desc t1
COLUMN_NAME TYPE                    IS_NULLABLE
----------- ----------------------- -----------
ID          NUMBER(10,0)            FALSE
NAME        CHARACTER VARYING(128)  TRUE
ADDR        CHARACTER VARYING(1024) TRUE
INDEX_NAME           TABLESPACE_NAME INDEX_TYPE IS_UNIQUE COLUMNS
-------------------- --------------- ---------- --------- -------
T1_PRIMARY_KEY_INDEX MEM_TEMP_TBS    BTREE      TRUE      ID
T1_IDX_NAME          MEM_TEMP_TBS    BTREE      FALSE     NAME
CONSTRAINT_NAME CONSTRAINT_TYPE ASSOCIATED_INDEX     COLUMNS
--------------- --------------- -------------------- -------
```

```
T1_PRIMARY_KEY  PRIMARY KEY     T1_PRIMARY_KEY_INDEX ID
```

The following is an example of querying the information of the t1 table by describing together with the s chema name public.

```
gSQL> \desc public.t1
COLUMN_NAME TYPE                    IS_NULLABLE
----------- ----------------------- -----------
ID          NUMBER(10,0)            FALSE
NAME        CHARACTER VARYING(128)  TRUE
ADDR        CHARACTER VARYING(1024) TRUE
INDEX_NAME           TABLESPACE_NAME INDEX_TYPE IS_UNIQUE COLUMNS
-------------------- --------------- ---------- --------- -------
T1_PRIMARY_KEY_INDEX MEM_TEMP_TBS    BTREE      TRUE      ID
T1_IDX_NAME          MEM_TEMP_TBS    BTREE      FALSE     NAME
CONSTRAINT_NAME CONSTRAINT_TYPE ASSOCIATED_INDEX     COLUMNS
--------------- --------------- -------------------- -------
T1_PRIMARY_KEY  PRIMARY KEY     T1_PRIMARY_KEY_INDEX ID
```

The following is an example of querying the information of the table created by using the delimited identi fier.

```
gSQL> CREATE TABLE "TaBle#@^*" ( id INTEGER );
Table created.
gSQL> \desc "TaBle#@^*"
COLUMN_NAME TYPE         IS_NULLABLE
----------- ------------ -----------
ID          NUMBER(10,0) TRUE
```

If the delimited identifier is not used as follows, then an error occurs.

```
gSQL> \desc TaBle#@^*
ERR-42000(40000): syntax error
\desc TaBle#@^*
...........^  ^
Error at line 1
```

\dynamic sql :var

Syntax

```
\dynamic sql :var
```

Description

It executes the SQL statement stored in the host variable var.
It is similar in concept to **Embedded Dynamic SQL** which executes SQL statements that are not defined in an embedded SQL, but It is used when performing SQL statement which is not defined.

It is performed in the following order.

1. Declare a host variable. (Refer to **\var**.)
   \var var_stmt VARCHAR(1024)
2. Assign an SQL statement as the value of the host variable. (Refer to **\exec :var := value**.)
   \exec :var_stmt := 'SELECT # FROM t1'
3. Execute the dynamic SQL.
   \dynamic sql :var_stmt

Examples

The following is an example of declaring the var_stmt host variable and executing the dynamic SQL by ass igning an SQL statement.

- Assigning the SELECT statement to the host variable

```
gSQL> \exec :var_stmt := 'SELECT * FROM t1'
```

- Executing the dynamic SQL

```
gSQL> \dynamic sql :var_stmt
ID NAME   ADDR
-- ------ ------------
 1 leekmo Seoul, Korea
1 row selected.
```

Two single quote (') are described like as the usage of the **\exec :var := value** command when a string exists in an SQL statement as the INSERT statement below.

- Declaring a host variable

```
gSQL> \var var_stmt VARCHAR(1024)
```

- Assigning the INSERT statement to the host variable

```
gSQL> \exec :var_stmt := 'INSERT INTO t1(id, name, addr) VALUES ( 1, ''leekmo'', ''Seoul, Korea'' )'
```

- Executing the dynamic SQL.

```
gSQL> \dynamic sql :var_stmt
1 row created.
```

\edit

Description

It edits the SQL statement by using the text editor. The used text editor is specified in the EDITOR environment variable, and if the EDITOR environment variable does not exist, vi is used by default.

When using the editing feature, gsql hands over control by executing the editor. The user can freely edit the SQL statements with the editor. When the editor ends, the control is handed over to gsql, and its content will be added as the last history.

The SQL statement edited by the editor may be executed through the last history execution command \\.

> ⚠ Only a single SQL statement can be edited with the SQL editor. An error occurs when executing multiple SQL statements.

The followings can be edited by using \edit.

- The most recently performed SQL statement
- the SQL statement stored in a file
- The SQL statement stored in the gsql history

The following chapter describes the usage.

\edit

Syntax

```
\edit
\ed
```

Description

It edits the most recently executed SQL statement by using the text editor. If any SQL statement has not b
een executed, the text editor is run without any content.

Example

The following is an example of editing the most recently executed SQL statement.

- Editing the most recently executed SQL statement

```
gSQL> SELECT * FROM T1;
C1
--
 1
11
2 rows selected.
gSQL> \EDIT
SELECT * FROM T1 WHERE C1 < 10;
```

- Executing the edited SQL statement

```
gSQL> \\
C1
--
 1
1 row selected.
```

\edit 'file_name'

Syntax

```
\edit 'file_name'
\ed   'file_name'
```

Description

It edits the given file_name by using the text editor.
The file_name is enclosed with the single quote (').
file_name can use either an absolute path or a relative path as follows. When file_name uses a relative pa th, it searches for the file_name based on the path of gsql execution.

- Using the absolute path

```
gSQL> \edit '/home/goldilocks/sample.sql'
```

- Using the relative path

```
gSQL> \edit 'sample.sql'
```

Example

The following is an example of editing the SQL statement stored in the file.

- Editing the most recently executed SQL statement

```
gSQL> SELECT * FROM T1;
C1
--
 1
11
2 rows selected.
gSQL> \EDIT 'sample.sql'
SELECT * FROM T1 WHERE C1 < 10;
```

- Executing the edited SQL statement

```
gSQL> \\
C1
--
 1
1 row selected.
```

\edit [history] {n}

Syntax

```
\edit [history] {n}
\ed   [history] {n}
```

Description

It edits the SQL statement corresponding to the number from the SQL execution history which can be qu eried by using **\history**.
The value of number should be the ID value which is the result of executing **\history**.

Example

The following is an example of editing the SQL statement stored in the gsql history.

```
gSQL> SELECT * FROM T1;
C1
--
 1
11
2 rows selected.
gSQL> \history
ID SQL
-- ----------------
 1 SELECT * FROM T1
```

- Editing the SQL statement whose ID value is 1 from the stored history

```
gSQL> \edit 1
SELECT * FROM T1 WHERE C1 < 5;
```

- Executing the edited SQL statement

```
gSQL> \\
C1
--
 1
1 row selected.
```

\exec

Syntax

```
\exec
```

Description

It executes the SQL statement which is prepared by **\prepare sql**.
\exec operates similarly to the **SQLExecute** function of ODBC and PreparedStatement::**execute** function of JDBC.
The prepared SQL statements can be repeatedly executed by using \exec.

Examples

The following is an example of preparing an INSERT statement and repeatedly executing it.

- Preparing the INSERT statement

```
gSQL> \prepare sql INSERT INTO t1(id, addr) VALUES ( seq.NEXTVAL, 'N/A' );
SQL prepared.
```

- Executing the prepared statement

```
gSQL> \exec
1 row created.
```

- Executing the prepared statement

```
gSQL> \exec
1 row created.
```

- The results of querying the table are as follows.

```
gSQL> SELECT * FROM t1;
ID ADDR
-- ----
 1 N/A
 2 N/A
2 rows selected.
```

The following is an example of preparing the SELECT statement with the host variable and repeatedly exe cuting it by changing the host variable value.

- Declaring the host variable

```
gSQL> \var v_id INTEGER
```

- Preparing the SELECT statement which used the host variable

```
gSQL> \prepare sql SELECT * FROM t1 WHERE ID = :v_id;
SQL prepared.
```

- Assigning the value 1 to the host variable

```
gSQL> \exec :v_id := 1
```

- Executing the prepared SELECT statement

```
gSQL> \exec
ID ADDR
-- ----
 1 N/A
1 row selected.
```

- Assigning the value 2 to the host variable

```
gSQL> \exec :v_id := 2
```

- Executing the prepared SELECT statement

```
gSQL> \exec
ID ADDR
-- ----
 2 N/A
1 row selected.
```

\exec :var := value

Syntax

```
\exec :variable := <value expression>
```

Description

It assigns the value to the host variable.
This command assigns a value to the host variable in the same way as executing **SELECT .. INTO** as follows.

```
gSQL> \exec :v_value := 1234
gSQL> SELECT 1234 INTO :v_value FROM DUAL;
```

The host variable must have been declared with **\var**.

The host variable must be used together with colon (:), and the assignment operator (:=) is not allowed to omit (:). Simple values or operators may appear in <value expression> entered to the host variable. For more information, refer to **Expressions**.

The value assigned to the host variable should be compatible with the data type of the host variable. For more information, refer to **Type Conversion**.

The value assigned to the host variable may be queried by using **\print**.

```
\exec :v_value := 1234
\print v_value
```

The string should be enclosed in the single quote (') as follows when inserting the string into the host variable.

- Simple string
  - Value: abcd
  - \exec :v_value := 'abcd'
- The string which includes the single quote
  - Value: Tom's House
  - The single quote within the string is represented with the two single quotes ('') as follows.
  - \exec :v_value := 'Tom''s House'
- The SQL statement including the single quote
  - Value: INSERT INTO t1 VALUES (1, 'Tom''s House')
  - The single quote within the string is represented with the two single quotes ('') as follows.
  - \exec :v_value := 'INSERT INTO t1 VALUES (1, ''Tom''s House'')'

Examples

The following is an example of assigning the various types of string to the host variable.

- Declaring the host variable

```
gSQL> \var v_value VARCHAR(1024)
```

- Assigning the simple string

```
gSQL> \exec :v_value := 'abcd'
gSQL> \print v_value
V_VALUE
-------
abcd
```

- Assigning the string including the single quote (')

```
gSQL> \exec :v_value := 'Tom''s House'
gSQL> \print v_value
V_VALUE
-----------
Tom's House
```

- Assigning the SQL statement including the single quote (')

```
gSQL> \exec :v_value := 'INSERT INTO t1 VALUES ( 1, ''Tom''s House'' )'
gSQL> \print v_value
V_VALUE
----------------------------------------
INSERT INTO t1 VALUES ( 1, 'Tom's House' )
```

The following is an example of using the operation result when assigning the value to the host variable.

- Declaring the host variable

```
gSQL> \var v1 INTEGER
gSQL> \var v2 INTEGER
```

- Assigning the value to v1

```
gSQL> \exec :v1 := 100
```

- Assigning the result of operation with v1 to v2

```
gSQL> \exec :v2 := :v1 + 1000
```

- Querying the value of host variables v1 and v2

```
gSQL> \print v1
 V1
---
100
gSQL> \print v2
  V2
----
1100
```

The following is an example of assigning the value to the host variable and using the host variable in the SELECT statement.

- Declaring the host variable

```
gSQL> \var v_id INTEGER
```

- Assigning the value to the host variable

```
gSQL> \exec :v_id := 1
```

- Executing the SELECT statement by using the host variable

```
gSQL> SELECT * FROM t1 WHERE id = :v_id;
ID ADDR
-- ----
 1 N/A
1 row selected.
```

The following is an example of obtaining the value to the host variable via **SELECT .. INTO**.

- Declaring the host variable

```
gSQL> \var v_id INTEGER
```

- A value is not specified.

```
gSQL> \print v_id
V_ID
----
```

```
null
```

- Assigning the value to v_id by executing the SELECT INTO statement

```
gSQL> SELECT MAX(id) INTO :v_id FROM t1;
V_ID
----
   2
1 row selected.
```

- Checking the host variable value

```
gSQL> \print v_id
V_ID
----
   2
```

\exec sql

Syntax

```
\exec sql <sql_statement>
```

Description

It executes the SQL statement described in <sql_statement>.
It operates as same as the execution of the SQL statement at the prompt as follows.

```
gSQL> SELECT * FROM dual;
DUMMY
-----
X
1 row selected.
gSQL> \exec sql SELECT * FROM dual;
DUMMY
-----
X
1 row selected.
```

The SQL statement is executed with the PREPARE/EXECUTE method using **\prepare sql** and **\exec**. On the other hand, \exec sql executes the SQL statement with the DIRECT EXECUTE method corresponding to **SQLExecDirect** function of ODBC and to Statement::**execute** function of JDBC.

Example

The following is an example of executing the SELECT statement.

```
gSQL> \exec sql SELECT * FROM dual;
DUMMY
-----
X
1 row selected.
```

\explain plan

Syntax

```
\explain plan <sql_statement>
\explain plan on <sql_statement>
\explain plan only <sql_statement>
```

Description

It outputs the execution plan of the SELECT statement described in <sql_statement>.

- \explain plan on
  - It executes the SQL statement and outputs the execution plan along with the query results.
- \explain plan only
  - It does not execute the SQL statement and outputs the execution plan without the query results.
- \explain plan
  - If on or only is not specified, the default value is on.

For more information about the execution plan, refer to **SQL Execution Plan**.

Examples

If it is used together with ON as follows, it executes the SQL statement and outputs the execution plan wi
th the query results.

```
gSQL> \explain plan on SELECT * FROM t1 WHERE id = 1;
ID NAME   ADDR
-- ------ ------------
 1 leekmo Seoul, Korea
1 row selected.
>>>  start print plan
< Execution Plan >
================================================================================
¦  IDX  ¦  NODE DESCRIPTION                                ¦       ROWS ¦
--------------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                ¦            ¦
¦    1  ¦    TABLE ACCESS ("T1")                         ¦          1 ¦
================================================================================
     1  -  READ COLUMNS : ID, NAME, ADDR
             PHYSICAL FILTER : ID = 1
<<<  end print plan
```

If it is used together with ONLY as follows, it does not execute the SQL statement and outputs the execut ion plan without the query results.

```
gSQL> \explain plan only SELECT * FROM t1 WHERE id = 1;
>>>  start print plan
< Execution Plan >
================================================================================
¦  IDX  ¦  NODE DESCRIPTION                                ¦       ROWS ¦
--------------------------------------------------------------------------------
¦    0  ¦  SELECT STATEMENT                                ¦            ¦
¦    1  ¦    TABLE ACCESS ("T1")                         ¦          0 ¦
================================================================================
     1  -  READ COLUMNS : ID, NAME, ADDR
             PHYSICAL FILTER : ID = 1
<<<  end print plan
```

\help

Syntax

```
\help
```

Description

It briefly displays the list of commands which begin with (\) in gsql interactive mode.

Examples

The following is an example of using \help.

```
gSQL> \help
\help
\q[uit]
\i[mport] {'FILE'}               Import SQL
\ed[it] [{'FILE'|[HISTORY] num}] Edit SQL statement
\\                               Executes the most recent history entry
\{n}                             Executes n'th history entry
\hi[story]                       Show history entries
\desc     {[schema.]table_name}  Show table description
\idesc    {[schema.]index_name}  Show index description
\spo[ol]  ['filename' | OFF]     Stores query results in a file
\ho[st]   [command]              Executes an operating system command
\set vertical    {ON|OFF}
\set time        {ON|OFF}
\set timing      {ON|OFF}
\set color       {ON|OFF}
\set error       {ON|OFF}
\set autocommit  {ON|OFF}
\set autotrace   {ON|TRACEONLY|OFF}
\set serveroutput {ON|OFF}
\set heading     {ON|OFF}
\set linesize    {n}      0 < n <= 100000
\set pagesize    {n}      0 < n <= 100000
\set colsize     {n}      0 < n <= 104857600
\set numsize     {n}      0 < n <= 50
\set ddlsize     {n}      0 < n <= 100000
\set history     {n}      n <= 100000 ( if n < 0, clear history buffer )
\var             {host_var_name} {INTEGER|BIGINT|VARCHAR(n)}
\exec            [{:host_var_name} := {constant}]
\exec sql        {sql string}
\prepare sql     {sql string}
\dynamic sql     {host_var_name}
\explain plan    [{ON|ONLY}] {sql string}
```

```
\print           [{host_var_name}]
\ddl_db
\ddl_tablespace    {name}
\ddl_auth          {name}
\ddl_schema        {name}
\ddl_table         {[schema.]name}
\ddl_constraint    {[schema.]name}
\ddl_index         {[schema.]name}
\ddl_view          {[schema.]name}
\ddl_sequence      {[schema.]name}
\ddl_synonym       {[schema.]name}
\ddl_public_synonym {name}
\startup         {[nomount|mount|open]}
\shutdown        {[abort|immediate|transactional|normal]}
\cstartup        {[nomount|mount|open]}
\cshutdown       {[abort|normal]}
\connect         [userid password] [as {sysdba|admin}]
```

\host

Syntax

```
\host [command]
\ho [command]
```

Description

It executes the command of the operational system without terminating gsql program.
If HOST is input without command, then the prompt of the operational system is displayed and it is able t o continuously input the command of the operational system.
Instead of HOST, "$" is input in Windows, and "!" is input in UNIX.

Example

The following is an example of executing ls *.sql statement which is a command of UNIX operational syst em.

```
gSQL> \host ls *.sql
DictionarySchema.sql  InformationSchema.sql  PerformanceViewSchema.sql
```

\history

Syntax

```
\history
\hi
```

Description

It displays the list of the SQL statements executed after executing gsql program.
It manages only the successfully executed SQL statements and it does not include the failed SQL statemen t nor the interactive command which begins with gsql(\).
The previously executed SQL statements can be executed again referring to the list through **** or **\{n}**.
The number of manageable SQL statements is controlled by using **\set history**.

Example

The following is an example of querying the SQL statement execution history and executing the SQL state ment of number 8 again.

```
gSQL> \history
ID SQL
-- ------------------------------------------------------------------
 1 drop table t1
 2 create table t1 ( id integer, name varchar(128), addr varchar(1024) )
 3 create index t1_idx on t1(id)
 4 insert into t1 values ( 1, 'leekmo', 'Seoul, Korea' )
 5 insert into t1 values ( 2, 'mkkim', 'Seoul, Korea' )
 6 insert into t1 values ( 3, 'xcom73', 'Inchon, Korea' )
 7 commit
 8 select * from dual
gSQL> \8
DUMMY
-----
X
1 row selected.
```

\import

Syntax

```
\import 'file_name'
\i 'file_name'
```

Description

It imports the SQL statement included in file_name.
The file_name is enclosed with the single quote (').
file_name can use either an absolute path or a relative path as follows. When file_name uses a relativepat
h, it searches for the file_name based on the path of gsql execution.

- Using the absolute path

```
gSQL> \import '/home/GOLDILOCKS/sample.sql'
```

- Using the relative path

```
gSQL> \import 'sample.sql'
```

Examples

The following is an example of importing the SQL statement from the file by using the absolute path.

```
gSQL> \import '/home/GOLDILOCKS/sample.sql'
DROP TABLE IF EXISTS t1;
Table dropped.
CREATE TABLE t1
(
    id   INTEGER,
    name VARCHAR(128),
    addr VARCHAR(128)
);
Table created.
INSERT INTO t1
       VALUES ( 1, 'leekmo', 'Seoul, Korea'  ),
              ( 2, 'mkkim' , 'Seoul, Korea'  ),
              ( 3, 'xcom73', 'Inchon, Korea' );
```

```
3 rows created.
COMMIT;
Commit complete.
```

The following is an example of importing the SQL statement from the file by using the relative path.

```
gSQL> \import 'sample.sql'
DROP TABLE IF EXISTS t1;
Table dropped.
CREATE TABLE t1
(
    id   INTEGER,
    name VARCHAR(128),
    addr VARCHAR(128)
);
Table created.
INSERT INTO t1
       VALUES ( 1, 'leekmo', 'Seoul, Korea'  ),
              ( 2, 'mkkim' , 'Seoul, Korea'  ),
              ( 3, 'xcom73', 'Inchon, Korea' );
3 rows created.
COMMIT;
Commit complete.
```

\idesc

Syntax

```
\idesc index_name
\idesc schema_name.index_name
```

Description

It queries the structure information of the index.

An index can be described alone as follows or it can be described together with the schema name. If the schema name is not specified, the schema name of the index is determined by the user's **Schema Path**.

- \idesc t1_idx_name
- \idesc public.t1_idx_name

The execution result includes the following key column information of the index.

- Key column name
- Key column location
- Key column sort order (ascending/ descending)
- NULL position of the key column(FIRST/ LAST)

Example

The following is an example of querying the information of the index t1_idx_name.

```
gSQL> \idesc t1_idx_name
COLUMN_NAME ORDINAL_POSITION IS_ASCENDING_ORDER IS_NULLS_FIRST
----------- ---------------- ------------------ --------------
NAME                       1 TRUE               FALSE
```

\{n}

Syntax

```
\number
```

Description

It executes the SQL statement corresponding to the number from the SQL execution history which can be queried by using **`\history`**.
The value of number should be the ID value which is the result of executing **`\history`**.

Examples

The following is an example of executing SQL statement which used `\history`, then executing the DROP TABLE statement whose ID value is 1.

```
gSQL> \history
ID SQL
-- --------------------------------------------
 1 DROP TABLE IF EXISTS t1
 2 CREATE TABLE t1
   (
```

```
       id   INTEGER PRIMARY KEY,
       name VARCHAR(128),
       addr VARCHAR(128)
   )
 3 CREATE INDEX t1_idx_name ON t1(name)
 4 INSERT INTO t1
          VALUES ( 1, 'leekmo', 'Seoul, Korea'  ),
                 ( 2, 'mkkim' , 'Seoul, Korea'  ),
                 ( 3, 'xcom73', 'Inchon, Korea' )
 5 COMMIT
 6 select * from t1
gSQL> \1
Table dropped.
```

\prepare sql

Syntax

```
\prepare sql <sql_statement>
```

Description

It prepares the SQL statement described in <sql_statement>. The prepared SQL statement can be execute d repeatedly with **\exec**.

\prepare sql operates similarly to the **SQLPrepare** functions of ODBC and Connection::**prepareStatement** function of JDBC.

Example

The following is an example of preparing the SELECT statement and repeatedly executing it.

- Preparing the SELECT statement

```
gSQL> \prepare sql SELECT * FROM t1 WHERE id > 2;
SQL prepared.
```

- Executing the prepared SQL statement

```
gSQL> \exec
ID NAME   ADDR
-- ------ -------------
 3 xcom73 Inchon, Korea
1 row selected.
gSQL> INSERT INTO t1 VALUES ( 4, 'GOLDILOCKS', 'Better Place' );
1 row created.
```

- Executing the prepared SQL statement again

```
gSQL> \exec
ID NAME   ADDR
-- ------ -------------
 3 xcom73 Inchon, Korea
 4 goldilocks  Better Place
2 rows selected.
```

\print

Syntax

```
\print
\print variable
```

Description

It queries the value of host variable declared with **\var** as follows. If the value of host variable is not set, it is NULL.

```
gSQL> \var v1 INTEGER
gSQL> \print v1
  V1
----
null
```

If the name of host variable is not specified as follows, it queries all declared host variables. VAR_ELAPSED_TIME_ is a built-in host variable which manages the execution time of SQL statement as follows.

```
gSQL> \print
NAME                VALUE
------------------- -----
VAR_ELAPSED_TIME__  null
V1                    21
V2                    10
```

Example

The following is an example of declaring the host variable, assigning the value to the host variable then querying it.

- Declaring the host variable

```
gSQL> \var v1 INTEGER
gSQL> \var v2 INTEGER
gSQL> \var v3 INTEGER
```

- Querying all host variables

```
gSQL> \print
NAME                VALUE
------------------- -----
VAR_ELAPSED_TIME__  null
V1                  null
V2                  null
V3                  null
```

- Assigning the value to the host variable

```
gSQL> \exec :v1 := 10
gSQL> \exec :v2 := 20
gSQL> \exec :v3 := :v1 + :v2
```

- Querying the host variable v3

```
gSQL> \print v3
V3
--
30
```

- Querying all host variables

```
gSQL> \print
NAME                VALUE
------------------- -----
VAR_ELAPSED_TIME__  null
V1                    10
V2                    20
V3                    30
```

\quit

Syntax

```
\quit
\q
```

Description

It quits gsql.
When quitting gsql, all uncommitted transactions are committed.

Example

```
gSQL> \quit
%
```

\set autocommit

Syntax

```
\set autocommit on
\set autocommit off
```

Description

It sets whether to automatically commit after executing the SQL statement.

- `\set autocommit on`
  - It automatically commits after executing the SQL statement.
- `\set autocommit off`
  - It does not automatically commit after executing the SQL statement.
- The default value of autocommit is OFF.

Examples

The following is a usage example when setting the AUTOCOMMIT value to ON.

- Setting the autocommit to on

```
gSQL> \set autocommit on
```

- The INSERT statement is automatically committed.

```
gSQL> INSERT INTO t1 ( id, name, addr ) VALUES ( 1, 'leekmo', 'Seoul, Korea' );
1 row created.
```

- It is not affected by rollback.

```
gSQL> ROLLBACK;
Rollback complete.
gSQL> SELECT * FROM t1;
ID NAME   ADDR
-- ------ ------------
 1 leekmo Seoul, Korea
1 row selected.
```

The following is a usage example when setting the AUTOCOMMIT value to OFF.

- Setting the autocommit to off

```
gSQL> \set autocommit off
```

- The transaction in not committed after executing the INSERT statement.

```
gSQL> INSERT INTO t1 ( id, name, addr ) VALUES ( 1, 'leekmo', 'Seoul, Korea' );
1 row created.
```

- The transaction is rolled back.

```
gSQL> ROLLBACK;
Rollback complete.
```

- The INSERT statement is rolled back without any result.

```
gSQL> SELECT * FROM t1;
no rows selected.
```

\set autotrace

Syntax

```
\set autotrace on
\set autotrace traceonly
\set autotrace off
```

Description

It sets whether to output the execution plan.

- `\set autotrace on`
  - It executes SQL statement, and outputs the execution plan together with the query result.
- `\set autotrace traceonly`
  - It does not execute SQL statement, and outputs the execution plan without the query result.
- `\set autotrace off`
  - It does not output the execution plan.
  - The default value is off.

For more information, refer to **SQL execution plan**.

Example

When using ON as follows, it executes SQL statement and outputs the execution plan together with the query result.

```
gSQL> \set autotrace on
gSQL> SELECT * FROM t1 WHERE id = 1;
ID NAME   ADDR
-- ------ ------------
 1 leekmo Seoul, Korea
```

```
1 row selected.
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                                |       ROWS |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                |            |
|    1  |    TABLE ACCESS ("T1")                           |          1 |
================================================================================
     1  -  READ COLUMNS : ID, NAME, ADDR
              PHYSICAL FILTER : ID = 1
<<<  end print plan
```

When using TRACEONLY as follows, it does not execute SQL statement, and outputs the execution plan without the query result.

```
gSQL> \set autotrace traceonly
gSQL> SELECT * FROM t1 WHERE id = 1;
>>>  start print plan
< Execution Plan >
================================================================================
|  IDX  |  NODE DESCRIPTION                                |       ROWS |
--------------------------------------------------------------------------------
|    0  |  SELECT STATEMENT                                |            |
|    1  |    TABLE ACCESS ("T1")                           |          0 |
================================================================================
     1  -  READ COLUMNS : ID, NAME, ADDR
              PHYSICAL FILTER : ID = 1
<<<  end print plan
```

\set color

Syntax

```
\set color on
\set color off
```

Description

It sets whether to output each row of the query results in a different color on the terminal to distinguish t hem.

- `\set color on`
  - Each row is output in a different color.
- `\set color off`
  - Each row is output in the same color.
- The default value is OFF.

If the length of each row is too long so that it overflows the terminal window, then the row is output ove r multiple lines. In this case, each row is hard to be distinguished, so it reduces the readability It is used to distinguish between the rows and increase the readability on the terminal.

Example

The following is an example of setting the color to ON.

```
gSQL> \set color on
gSQL> SELECT id, addr FROM t1;
ID ADDR
-- -------------
 1 Seoul, Korea
 2 Seoul, Korea
 3 Inchon, Korea
3 rows selected.
```

\set colsize

Syntax

```
\set colsize number
```

Description

It sets the maximum length of the data when outputting LONG VARCHAR, LONG VARBINARY data.

- The colsize value is a positive integer between 1 and 104,857,600.(100M, the maximum length of L ONG VARCHAR)

- The default value of colsize is 8,192.

The string length of the LONG VARCHAR column is too long, so the readability of the query execution is l ow as follows. In this case, the readability of LONG VARCHAR column can be increased by reducing the c olsize, or it may be output as long as the desired length by enlarging the colsize.

```
gSQL> SELECT view_name, text FROM all_views WHERE view_name LIKE 'ALL_%' FETCH 3;
VIEW_NAME
----------------
TEXT
--------------------------------------------------------------------------------
ALL_ALL_TABLES
SELECT
       auth.AUTHORIZATION_NAME                ❶ OWNER
     , sch.SCHEMA_NAME                        ❷ TABLE_SCHEMA

... Ellipsis ...
                                                    WHERE      , pvcol.GRANTOR_ID
     , pvcol.GRANTEE_ID
     , pvcol.PRIVILEGE_TYPE_ID

3 rows selected.
```

Example

The following is an example of increasing the readability of the LONG VARCHAR column by reducing the colsize.

```
gSQL> \set colsize 200
gSQL> SELECT view_name, text FROM all_views WHERE view_name LIKE 'ALL_%' FETCH 3;
VIEW_NAME        TEXT
---------------- ------------------------------------------------------------
ALL_ALL_TABLES   SELECT
                         auth.AUTHORIZATION_NAME                ❶ OWNER
                       , sch.SCHEMA_NAME                        ❷ TABLE_SCHEMA
                       , tab.TABLE_NAME                         ❸ TABLE_NAME
                       , spc.TAB
ALL_COL_COMMENTS SELECT
                         auth.AUTHORIZATION_NAME
                       , sch.SCHEMA_NAME
                       , tab.TABLE_NAME
```

```
                      , col.COLUMN_NAME
                      , col.COMMENTS
                   FROM
                        DICTIONARY_SCHEMA.WHOLE_COLUMNS AS col
                      , DICTIONARY_S
ALL_COL_PRIVS    SELECT
                        grantor.AUTHORIZATION_NAME
                      , grantee.AUTHORIZATION_NAME
                      , owner.AUTHORIZATION_NAME
                      , sch.SCHEMA_NAME
                      , tab.TABLE_NAME
                      , col.COLUMN_NAME
                      , pvcol.PRIVILEGE_TY
3 rows selected.
```

\set ddlsize

Syntax

```
\set ddlsize number
```

Description

It sets the buffer size to output the statement when outputting the DDL statements by using the followin g commands.

- \ddl_db
- \ddl_tablespace
- \ddl_auth
- \ddl_schema
- \ddl_table
- \ddl_constraint
- \ddl_index
- \ddl_view
- \ddl_sequence
- \ddl_synonym
- \ddl_public_synonym
- \ddl_procedure
- \ddl_package

- The value of ddlsize is a positive integer between 1 and 1,0485,760 (10M).
- The default value of ddlsize is 10,000.

If an error occurs due to the lack of buffer space as follows, DDL statement may be output by increasing t he value of ddlsize.

```
gSQL> \set ddlsize 1000
gSQL> \ddl_view dictionary_schema.all_tables
ERR-HY000(40052): not enough DDLSIZE.
use command: \set ddlsize {n}
gSQL> \set ddlsize 100000
gSQL> \ddl_view dictionary_schema.all_tables
SET SESSION AUTHORIZATION "SYS";
CREATE OR REPLACE FORCE VIEW "DICTIONARY_SCHEMA"."ALL_TABLES"
... Ellipsis ...
```

Example

The following is an example of changing the ddlsize.

```
gSQL> \set ddlsize 100000
gSQL>
```

\set error

Syntax

```
\set error on
\set error off
```

Description

It sets whether to output the error message.

- `\set error on`
  - It outputs the error message.
- `\set error off`
  - It does not output the error message.
- The default value of error is ON.

If an error occurs during executing the SQL statement, gsql outputs the following information.

- SQLSTATE: SQL standard state code
- Error code: GOLDILOCKS error code
- Error message

The following example describes that the error of the SQL statement occurs, the SQLSTATE value in ERR-42000(16040) is 42000, and the error code is 16040 within ().

```
gSQL> SELECT * FROM invalid_table;
ERR-42000(16040): table or view does not exist :
SELECT * FROM invalid_table
              *
ERROR at line 1:
```

Example

The following is an example of disabling the error message.

- Disabling the error message

```
gSQL> \set error off
```

- Outputting only the value of SQLSTATE and the error code

```
gSQL> SELECT * FROM invalid_table;
ERR-42000(16040)
```

\set heading

Syntax

```
\set heading {ON|OFF}
```

Description

It sets whether to output the header in the query result.

Example

- Set not to output the header message.

```
gSQL> \set heading off
```

- The following is an example of a query result in which the header message is not output.

```
gSQL> select * from dual;
X
1 row selected.
```

\set history

Syntax

```
\set history number
```

Description

It sets the number of SQL statements to manage the history information.

- The history value is a positive integer between 1 and 100,000.
- If the history value is set to a negative value, all SQL history are removed.
- The default value of history is 128.

If the history value is reduced, the old SQL statement is removed.
The history information is used when executing the previously executed SQL statement again through **** and **\{n}**.

Example

The following is an example of setting the number of history to 100.

```
gSQL> \set history 100
gSQL>
```

The following is an example of setting the number of history to smaller than the number of stored SQL st
atements.

- Five SQL statements are stored.

```
gSQL> \history
ID SQL
-- ---------------------------------------------
 1 DROP TABLE IF EXISTS t1
 2 CREATE TABLE t1
   (
       id   INTEGER PRIMARY KEY,
       name VARCHAR(128),
       addr VARCHAR(128)
   )
 3 CREATE INDEX t1_idx_name ON t1(name)
 4 INSERT INTO t1
         VALUES ( 1, 'leekmo', 'Seoul, Korea'  ),
                ( 2, 'mkkim' , 'Seoul, Korea'  ),
                ( 3, 'xcom73', 'Inchon, Korea' )
 5 COMMIT
```

- The history value is reduced to 3.

```
gSQL> \set history 3
```

- Three SQL statements are stored.

```
gSQL> \history
ID SQL
-- ---------------------------------------------
 3 CREATE INDEX t1_idx_name ON t1(name)
 4 INSERT INTO t1
         VALUES ( 1, 'leekmo', 'Seoul, Korea'  ),
                ( 2, 'mkkim' , 'Seoul, Korea'  ),
                ( 3, 'xcom73', 'Inchon, Korea' )
 5 COMMIT
```

The following is an example of removing all SQL statements stored in the history.

```
gSQL> \history
ID SQL
-- -------------------------
 1 SELECT * FROM dual
 2 SELECT * FROM user_tables
```

- All SQL history are removed by using a negative value.

```
gSQL> \set history -1
gSQL> \history
gSQL>
```

\set linesize

Syntax

```
\set linesize number
```

Description

It sets the maximum size of a single line when outputting the query result.

- The value of linesize is a positive integer between 1 and 10,000.
- The default value of linesize is 80.

Each row of the query result is output on a single line basis. If there are many columns in a row or the ro w length is bigger than linesize, a row is output to the multiple lines so the readability is decreased.

```
gSQL> SELECT * FROM dict_columns WHERE table_name = 'USER_TABLES' FETCH 3;
TABLE_SCHEMA      TABLE_NAME  COLUMN_NAME
----------------- ----------- ---------------
COMMENTS
----------------------------------------
DICTIONARY_SCHEMA USER_TABLES TABLE_SCHEMA
Schema of the table
DICTIONARY_SCHEMA USER_TABLES TABLE_NAME
Name of the table
DICTIONARY_SCHEMA USER_TABLES TABLESPACE_NAME
Name of the tablespace containing the table
3 rows selected.
```

In this case, a row may be controlled to be output in a single line by changing the linesize value.

Example

The following is an example of increasing the readability by setting the linesize larger.

```
gSQL> \set linesize 400
gSQL> SELECT * FROM dict_columns WHERE table_name = 'USER_TABLES' FETCH 3;
TABLE_SCHEMA      TABLE_NAME  COLUMN_NAME     COMMENTS
----------------- ----------- --------------- ----------------------------------------
DICTIONARY_SCHEMA USER_TABLES TABLE_SCHEMA    Schema of the table
DICTIONARY_SCHEMA USER_TABLES TABLE_NAME      Name of the table
DICTIONARY_SCHEMA USER_TABLES TABLESPACE_NAME Name of the tablespace containing the table
3 rows selected.
```

\set numsize

Syntax

```
\set numsize number
```

Description

It sets the maximum number of digit for the output of a numeric value.

- The numsize value is a positive integer between 1 and 50.
- The default value of numsize is 20.

If the number of digit of a numeric value exceeds the numsize range as follows, it is output in an expone ntial form.

```
gSQL> SELECT num, ( num * num ) AS result FROM t1;
          NUM               RESULT
------------- --------------------
1234567890123 1.52415787532276E+24
1 row selected.
```

By changing the value of numsize, these numeric value is output in a numeric form, not in an exponent f orm.

Example

The following is an example of outputting all digits of the numeric value by setting the numsize larger.

```
gSQL> \set numsize 50
gSQL> SELECT num, ( num * num ) AS result FROM t1;
          NUM                   RESULT
------------ ------------------------
1234567890123 15241578753227558009551 29
1 row selected.
```

\set pagesize

Syntax

```
\set pagesize number
```

Description

It sets the number of the rows which is to be consisted in a single page.

- The value of pagesize is a positive integer between 1 and 10,000.
- The default value of pagesize is 20.

It is set when there are many rows of the query results or when a page should consist of a certain number of row.

Example

The following is an example of consisting a page in 10 rows unit.

```
gSQL> \set linesize 120
gSQL> \set pagesize 10
gSQL> SELECT column_name, comments FROM dict_columns  WHERE table_name = 'SEQUENCES';
COLUMN_NAME       COMMENTS
----------------- ------------------------------------------------------------------------
OWNER_ID          authorization identifier who owns the table of the sequence generator
SCHEMA_ID         schema identifier of the sequence generator
SEQUENCE_ID       sequence generator identifier
SEQUENCE_TABLE_ID table id of sequence for naming resolution
```

```
TABLESPACE_ID     tablespace identifier of the sequence generator
PHYSICAL_ID       physical identifier of the sequence generator
SEQUENCE_NAME     sequence generator name
DTD_IDENTIFIER    unsupported feature
START_VALUE       the start value of the sequence generator
MINIMUM_VALUE     the minimum value of the sequence generator
COLUMN_NAME      COMMENTS
---------------- ----------------------------------------------------------------
MAXIMUM_VALUE    the maximum value of the sequence generator
INCREMENT        the increment of the sequence generator
CYCLE_OPTION     The values of CYCLE_OPTION have the following meanings:
                 - TRUE : The cycle option of the sequence generator is CYCLE.
                 - FALSE : The cycle option of the sequence generator is NO CYCLE.

CACHE_SIZE       number of sequence numbers to cache
CREATED_TIME     created time of the sequence generator
MODIFIED_TIME    last modified time of the sequence generator
COMMENTS         comments of the sequence generator
SEQUENCE_CATALOG catalog name of the sequence
SEQUENCE_OWNER   owner name of the sequence
SEQUENCE_SCHEMA  schema name of the sequence
COLUMN_NAME             COMMENTS
----------------------- ------------------------------------------------------------
SEQUENCE_NAME           sequence name
DATA_TYPE               the standard name of the data type
NUMERIC_PRECISION       the numeric precision of the numerical data type
NUMERIC_PRECISION_RADIX the radix ( 2 or 10 ) of the precision of the numerical data type
NUMERIC_SCALE           the numeric scale of the exact numerical data type
START_VALUE             the start value of the sequence generator
MINIMUM_VALUE           the minimum value of the sequence generator
MAXIMUM_VALUE           the maximum value of the sequence generator
INCREMENT               the increment of the sequence generator
CYCLE_OPTION            cycle option
COLUMN_NAME               COMMENTS
------------------------- -------------------------------------------
CACHE_SIZE                number of sequence numbers to cache
DECLARED_DATA_TYPE        the data type name that a user declared
DECLARED_NUMERIC_PRECISION the precision value that a user declared
DECLARED_NUMERIC_SCALE    the scale value that a user declared
CREATED_TIME              created time of the sequence generator
MODIFIED_TIME             last modified time of the sequence generator
```

```
COMMENTS                  comments of the sequence generator
37 rows selected.
```

\set serveroutput

Syntax

```
\set serveroutput on
\set serveroutput off
```

Description

It controls automatic output feature for the message writeten by functions of which DBMS_OUTPUT pack age provides in gsql or gsqlnet

- `\set serveroutput on`
  - It automatically outputs messages accumulated in the server by DBMS_OUTPUT.PUT_LINE after e xecuting SQL.
- `\set serveroutput off`
  - It does not use DBMS_OUTPUT package.
- The default value of serveroutput is OFF.

Basically, the maximum size of accumulated in the server is 20000 bytes.

Example

The following is an example of outputting the messages accumulated in a user defined function which w as used in SQL statement.

```
gSQL> create or replace function my_msg( msg varchar(100) )
return integer
is
begin
  dbms_output.put_line( 'my message is : ' || msg );
  return length( msg );
end;
/
2 3 4 5 6 7 8
Function created.
gSQL> commit;
```

```
Commit complete.
gSQL> \set serveroutput on
gSQL> select my_msg( 'Hello World!' ) from dual;
MY_MSG( 'Hello World!' )
------------------------
                      12
my message is : Hello World!
1 row selected.
```

\set sqlprompt

Syntax

```
\set sqlprompt "prompt_sql"
```

Description

It sets SQL for prompt of gsql or gsqlnet.

- Maximum 127 characters are allowed in SQL statement.
- Maximum 31 characters are allowed in the prompt returning to SQL statement.

Even when SQL is set, the prompt is not changed until connecting to the database.

If SQL statement is incorrect or the returning data is longer than 31 characters, then the user SQL statem ent is output following [SQLPROMPT ERROR] statement.

Example

The following is an example of outputting the user name and the cluster member name in the prompt.

```
gSQL> \set sqlprompt "current_user || '@' || cluster_member_name || '>'"
TEST@G1N1>
```

\set time

Syntax

```
\set time on
\set time off
```

Description

It sets whether to output the current time.

- `\set time on`
  - It outputs the current time.
- `\set time off`
  - It does not output the current time.
- The default value of time is OFF.

Example

The following is an example of outputting the current time.

```
gSQL> \set time on
12:45:34 gSQL> SELECT * FROM dual;
DUMMY
-----
X
1 row selected.
Elapsed time: 0.07600 ms
```

\set timing

Syntax

```
\set timing on
\set timing off
```

Description

It sets whether to output the execution time of SQL statement.

- `\set timing on`

    - The execution time is output.
- `\set timing off`
    - The execution time is not output.
- The default value of timing is OFF.

The unit of execution time is ms (millisecond).

Example

The following is an example of outputting the execution time of SQL statement.

```
gSQL> \set timing on
gSQL> SELECT * FROM dual;
DUMMY
-----
X
1 row selected.
Elapsed time: 0.07600 ms
```

\set vertical

Syntax

```
\set vertical on
\set vertical off
```

Description

It sets whether to output the value of column in line unit.

- `\set vertical on`
    - A column is output in a line unit.
- `\set vertical off`
    - A row is output in a line unit.
- The default value of vertical is OFF.

If each row has an individual information, the readability can be increased by outputting the query result i n a column unit.

If `\set vertical on` is set, then each row is separated by a blank line, and one line represents a single valu

e, then it is consisted in the following form.

```
column name # data value
```

Example

The following is an example of outputting a query result in a column unit.

```
gSQL> \set vertical on
gSQL> SELECT * FROM v$system_stat FETCH 10;
                STAT_NAME # SYSTEM_SAR
               STAT_VALUE # 3
                 COMMENTS # system available resource( 0:none, 1:session 2:database )
                STAT_NAME # MAX_ENVIRONMENT_COUNT
               STAT_VALUE # 128
                 COMMENTS # maximum environment count
                STAT_NAME # FREE_ENVIRONMENT_ID
               STAT_VALUE # 81
                 COMMENTS # available environment identifier
                STAT_NAME # MAX_SESSION_COUNT
               STAT_VALUE # 128
                 COMMENTS # maximum session count
                STAT_NAME # FREE_SESSION_ID
               STAT_VALUE # 82
                 COMMENTS # available session identifier
                STAT_NAME # MAX_PROCESS_COUNT
               STAT_VALUE # 128
                 COMMENTS # maximum process count
                STAT_NAME # FREE_PROCESS_ID
               STAT_VALUE # 1
                 COMMENTS # available process identifier
                STAT_NAME # CACHE_ALIGNED_SIZE
               STAT_VALUE # 64
                 COMMENTS # cache aligned size
                STAT_NAME # CPU_COUNT
               STAT_VALUE # 8
                 COMMENTS # count of CPUs
                STAT_NAME # SYSTEM_TIME
               STAT_VALUE # 1408676900172925
                 COMMENTS # system time
10 rows selected.
```

\shutdown

Syntax

```
\shutdown
\shutdown abort
\shutdown immediate
\shutdown transactional
\shutdown normal
```

Description

It shuts down the GOLDILOCKS server.
It should be connected as SYSDBA or ADMIN role to perform \shutdown. For more information about con necting as SYSDBA, refer to **Startup and Shutdown Server**.

- \shutdown normal
  - It blocks the connection from the new session and waits for all currently connected sessions to be terminated, then performs the checkpoint and shuts down the server.
- \shutdown transactional
  - It blocks the start of a new transaction and waits for all currently running transactions to be termi nated, then performs the checkpoint and shuts down the server.
- \shutdown immediate
  - It blocks the execution of a new unit operation(ex FETCH or EXECUTE, etc.), waits for all currentl y running unit operations to be terminated, then rolls back all transactions and performs the chec kpoint and shuts down the server.
- \shutdown abort
  - The server is forcibly shut down immediately regardless of the status of the currently connected s ession.
- \shutdown
  - It is as same as \shutdown normal.

In GOLDILOCKS cluster system, if the current phase is GLOBAL OPEN when using \shutdown to shut dow n a single node, then abort is the only option, but if the current phase is LOCAL OPEN or below, then all options are available.
If using an option other than abort on GLOBAL OPEN phase, then the following error occurs.

```
% gsql sys gliese --as sysdba
Connected to GOLDILOCKS Database.
gSQL> \shutdown immediate
ERR-42000(16430): in the GLOBAL OPEN phase, the cluster database can be closed only with ABORT
```

```
option :
ALTER SYSTEM CLOSE DATABASE IMMEDIATE
                             *
ERROR at line 1:
gSQL>
```

Example

The following is an example of shutdowning GOLDILOCKS.

```
% gsql --as sysdba
Connected to GOLDILOCKS Database.
gSQL> \shutdown
Shutdown success
gSQL>
```

\spool

Description

It sets all results output by using gsql to be stored in the terminal and file.

\spool 'filename'

Syntax

```
\spool 'filename' [CREATE ¦ REPLACE ¦ APPEND]
\spo   'filename' [CREATE ¦ REPLACE ¦ APPEND]
```

Description

It starts the spool function. All output results performed by gsql are stored in the given filename since exe cuting this statement.

The following options describe how to open the file.

- CREATE: It creates the file. If a file already exists, an error occurs.
- REPLACE: It removes the contents of an existing file, and newly starts storing. If a file does not exist, i t performs as same as CREATE.
- APPEND: It continues to store the contents from the end of an existing file. If a file does not exist, it p

erforms as same as CREATE.

If these options are not given, it is operated in REPLACE mode by default.


When starting a new spool during spooling, the existing spool stops and a new spool starts.

Example

The following is an example of starting spooling.

- Creating the result.txt file and storing the gsql execution result

```
gSQL> \SPOOL 'result.txt' CREATE
gSQL> SELECT * FROM T1 WHERE C1 < 10;
gSQL> \SPOOL OFF
```

- Continuing to store the result at the end of the existing result.txt file

```
gSQL> \SPOOL 'result.txt' APPEND
gSQL> SELECT * FROM T1 WHERE C1 >= 10;
gSQL> \SPOOL OFF
```

- Removing the contents of the existing result.txt file and storing a new result

```
gSQL> \SPOOL 'result.txt' REPLACE
gSQL> SELECT * FROM T1;
gSQL> \SPOOL OFF
```

\spool OFF

Syntax

```
\spool OFF
\spo   OFF
```

Description

It ends the current spool. If the spool is not in use, it does not perform any transaction.

Example

The following is an example of starting the spool then ending it.

```
gSQL> \SPOOL 'result.txt'
gSQL> SELECT * FROM T1;
```

- Ending spooling

```
gSQL> \SPOOL OFF
```

\spool

Syntax

```
\spool
\spo
```

Description

It represents the current spool state. If the spool is in use, it informs which file is spooling. Otherwise, it in forms that the spool is not in use.

Example

The following is an example of editing the SQL statement stored in the gsql history.

```
gSQL> \SPOOL 'a.txt'
```

- It informs that it is spooling in a.txt.

```
gSQL> \SPOOL

currently spooling to a.txt

gSQL> \SPOOL OFF
```

- It informs that it is not spooling.

```
gSQL> \SPOOL

not spooling currently
```

\startup

Syntax

```
\startup
\startup nomount
\startup mount
\startup open
```

Description

It starts up the GOLDILOCKS server.
It should be connected as SYSDBA or ADMIN role to perform \startup.
For more information about connecting as SYSDBA, refer to **Startup and Shutdown Server**.

- \startup nomount
  - It starts up the server on the NOMOUNT phase.
- \startup mount
  - It starts up the server on the MOUNT phase.
- \startup open
  - It starts up the server on the OPEN phase.
- \startup
  - It is as same as \startup open.

The server start up is divided into NOMOUNT, MOUNT, OPEN phase. For more information, refer to **Multi-level Startup**.
Execute **ALTER SYSTEM {MOUNT | OPEN} DATABASE** statement to move on to the next phase after exec uting \startup.

Example

The following is an example of starting up GOLDILOCKS.

```
% gsql --as SYSDBA
Connected to an idle instance.
gSQL> \startup
Startup success
```

\var

Syntax

```
\var variable_name data_type
```

Description

It declares the host variable.
When declaring the host variable which is as same as variable_name, the existing host variable is remove d and a new one is declared. The maximum length of variable_name is 128 bytes.
The data_type of host variable is as same as the data type of GOLDILOCKS.
For more information, refer to **Data Type**.
The host variable assigns the value by using **\exec :var := value** , and its value is queried by using **\prin t**.

```
gSQL> \var v1 INTEGER
gSQL> \exec :v1 := 1
gSQL> \print v1
V1
--
 1
```

The host variable can be used as an input or output parameter in the SQL statement. When the host varia ble is used in the SQL statement or **\exec :var := value**, a colon (:) sign should be put in front of the hos t variable to indicate that it is a host variable.

Example

The following is an example of declaring the host variable and using it as an input argument of the SQL st atement.

```
gSQL> \var v1 INTEGER
gSQL> \exec :v1 := 1
gSQL> SELECT * FROM t1 WHERE id = :v1;
ID NAME   ADDR
-- ------ ------------
 1 leekmo Seoul, Korea
1 row selected.
```

The following is an example of using the host variable as an output argument of the SQL statement.

```
gSQL> \var v_name VARCHAR(128)
gSQL> SELECT name INTO :v_name FROM t1 WHERE id = 1;
V_NAME
------
leekmo
1 row selected.
```

The following is an example of declaring the host variable and using it as the input and output arguments of the SQL statement.

In the following **UPDATE name RETURNING .. INTO** statement, the host variable of SET clause and WHERE clause, :v_id, is used as an input argument and :v_id of INTO clause is used as an output argument.

```
gSQL> \var v_name VARCHAR(128)
gSQL> \var v_id INTEGER
gSQL> \exec :v_id := 1
gSQL> UPDATE t1 SET id = 100 + :v_id WHERE id = :v_id RETURNING id INTO :v_id;
V_ID
----
 101
1 row updated.
gSQL> \print v_id
V_ID
----
 101
```

41.

gloader/gloadernet(Upload/download Tool)

41.1 Overview of gloader and gloadernet

gloader is a utility which downloads or uploads data of GOLDILOCKS in table unit.

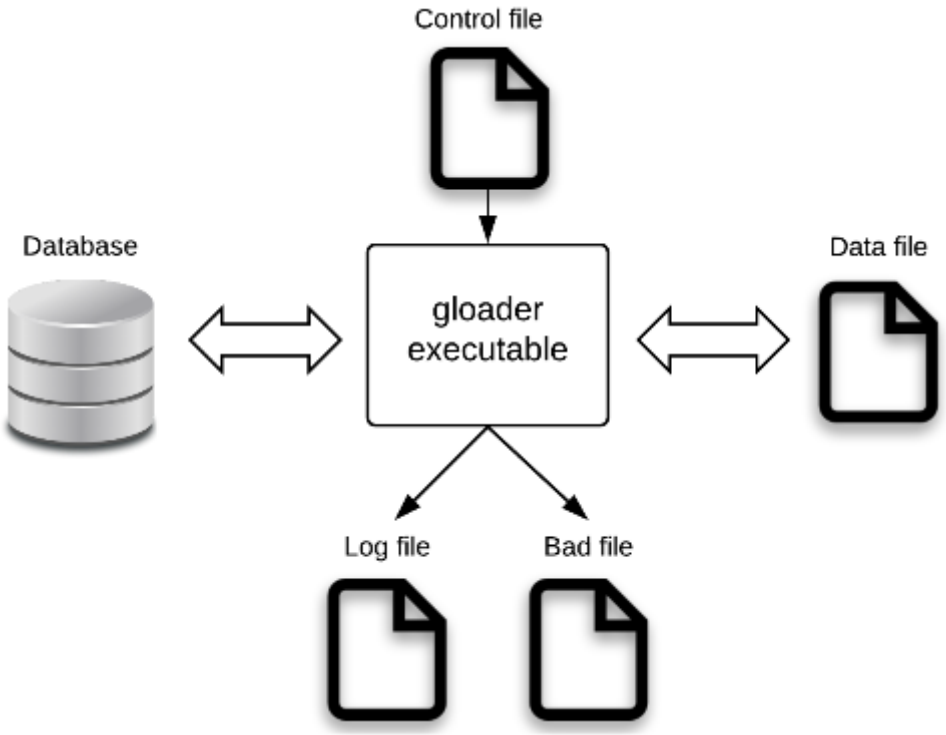
Table 41-1 Execution files

Name	Description
gloader	It is used in Direct Attach (D/A) environment.
gloadernet	It is used in Client/ Server (C/S) environment.

Environment

gloader should be connected to the database and it requires attention to all required files while using glo ader.

Figure 1 gloader environment



The control file and datafile are required to upload the data, then the log file is generated as a result.
The control file is required to download the data, then the datafile, log file, and bad file are generated as the results.

Control File

The control file is a file for operating gloader and it includes the following information. (Refer to **Control File Syntax**.)

- Table name
- Schema name
- The delimiter between columns in a row
- The qualifier notifying the start and end of the data
- The delimiter between rows
- Character set
- Whether to trim the whitespace character
- Where clause

DataFile

The datafile should be prepared when gloader uploads the data, and it is created when gloader downloads the data.
The datafile supports text format and binary format.

- The datafile in text format has an advantage of which the file contents can be checked and directly updated.
- The datafile in binary format can be performed faster comparing to the datafile in text format.

gloader uses direct I/O for the data file by default. gloader arbitrarily adjusts the file size if the file size is not an array appropriate for direct I/O when uploading the data file by using direct I/O.

Log File

Log file is a file which stores the following errors and results which occur while operating gloader.

- The row number and cause of the error
- The operating results of gloader

Bad File

Bad file is a file which stores the rows in which an error occurred while gloader uploads the data. The delimiter between columns and rows, and the qualifier which are used to store the bad file should be user-defined.

Example

The following is an example of downloading and uploading data by using gloader.

A table is created by using the SQL statement as follows.

```
$ cat test.sql
CREATE TABLE TEST
(
TEST_NAME VARCHAR(60),
TEST_NUM  INTEGER,
TEST_TIME TIMESTAMP(0) WITH TIME ZONE
);
INSERT INTO TEST VALUES
( 'NAME', 1, '1999-01-08 04:05:06.789 -8:00' );
INSERT INTO TEST VALUES
( 'NAME', 2, '1999-01-08 04:05:06.789 -8:00' );
INSERT INTO TEST VALUES
( 'NAME', 3, '1999-01-08 04:05:06.789 -8:00' );
COMMIT;
```

The control file is used as follows.

```
$ cat test.ctl
TABLE TEST
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
```

- Export: It downloads the data.

```
$ gloader test test --export --control test.ctl --data test.dat --no-prompt
 COMPLETED IN EXPORTING TABLE: PUBLIC.TEST, 3 RECORDS
$ cat test.dat
"NAME","1","1999-01-08 04:05:07. -08:00"
"NAME","2","1999-01-08 04:05:07. -08:00"
"NAME","3","1999-01-08 04:05:07. -08:00"
$ cat test.log
cat test.log
COMPLETED IN EXPORTING TABLE: PUBLIC.TEST, 3 RECORDS [ Start Time: 2010-1-1 01:01:01 End Time:
2010-1-1 01:01:01 Taken Time: 56496 micro-sec ]
```

- Import: It uploads the data.

```
$ cat import.dat
"NAME","1","1999-01-08 04:05:07. -08:00"
"NAME","2","1999-01-08 04:05:07. -08:00"
"NAME","3","1999-01-08 04:05:07. -08:00"
"FAIL","FAIL","FAIL"
$ gloader test test --import --control test.ctl --data import.dat --no-prompt
 COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 4 RECORDS, SUCCEEDED 3 RECORDS
$ gsql test test
gSQL> select * from test;
TEST_NAME TEST_NUM TEST_TIME
--------- -------- -------------------------------
NAME             1 1999-01-08 04:05:07.000000 -08:00
NAME             2 1999-01-08 04:05:07.000000 -08:00
NAME             3 1999-01-08 04:05:07.000000 -08:00
NAME             1 1999-01-08 04:05:07.000000 -08:00
NAME             2 1999-01-08 04:05:07.000000 -08:00
NAME             3 1999-01-08 04:05:07.000000 -08:00
6 rows selected.
$ cat import.log
Err Rec(4) Col(2): 22018(12006): data value is not a numeric literal
COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 4 RECORDS, SUCCEEDED 3 RECORDS [ Start Time:
2010-1-1 01:01:01 End Time: 2010-1-1 01:01:01 Taken Time: 56496 micro-sec ]
$ cat import.bad
"FAIL","FAIL","FAIL"
```

41.2 Using gloader

Datafile Type

Text Datafile

It is represented with a string which can be checked and edited by the user. A user can directly create, edi t the file, or can download the data from the existing tables in the database. A user also can use the data in a text format downloaded from another DBMS products.
The description for the representation of the text type datafile is recorded in the control file.

Binary Datafile

The file consists of binary data. A user can not directly create or edit the binary datafile. The file is generat ed when downloading the data from the existing tables in the database.

The binary type file is uploaded faster than the text type because the data is written to the file appropriat e to the data structure type defined in GOLDILOCKS.

When GOLDILOCKS databases' versions are different one another, then it is not recommended to upload/ download by using a binary datafile. Also, it may be required to chang a column size when uploading/ downloading data between databases whose string sets are different.

Downloading Data

Downloading in Text File

Simple Download

The following is the structure and data of the table to be downloaded.

```
$ cat test.sql
CREATE TABLE TEST ( I1 INTEGER PRIMARY KEY, I2 VARCHAR(10), I3 VARBINARY(10) );
INSERT INTO TEST VALUES( 1, 'LKH', X'10' );
INSERT INTO TEST VALUES( 2, 'KMM', X'A0' );
INSERT INTO TEST VALUES( 3, 'ksj', X'CD' );
COMMIT;
```

```
$ gsql test test
gSQL> SELECT * FROM TEST;
I1 I2  I3
-- --- --
 1 LKH 10
 2 KMM A0
 3 ksj CD
3 rows selected.
```

The following is the contents of the control file which is created to download the table data.

```
$ cat test.ctl
TABLE  PUBLIC.test
FIELDS TERMINATED BY ','
```

The data in the table T1 is downloaded through gloader as follows.

```
$ gloader test test --export --control test.ctl --data test.dat
 COMPLETED IN EXPORTING TABLE: PUBLIC.test, 3 RECORDS
```

The data in the table TEST is downloaded to the datafile as follows.

```
$ ls
test.ctl test.dat test.log
$ cat test.dat
1,LKH,10
2,KMM,A0
3,ksj,CD
```

Whitespace Character

The following is the structure and data of the table to be downloaded.

```
$ cat test.sql
CREATE TABLE TEST ( I1 INTEGER PRIMARY KEY, I2 VARCHAR(10), I3 VARBINARY(10) );
INSERT INTO TEST VALUES( 1, ' L K H ', X'10' );
INSERT INTO TEST VALUES( 2, 'KIM
MM', X'A0' );
INSERT INTO TEST VALUES( 3, ' KIM S
J ', X'CD' );
COMMIT;
$ gsql test test
```

```
gSQL> SELECT * FROM TEST;
I1 I2      I3
-- ------- --
 1  L K H  10
 2 KIM     A0
   MM
 3  KIM S  CD
   J
3 rows selected.
```

- The control file without using OPTIONALLY ENCLOSED BY statement
    - The following is the contents of the control file which is created to download the table data.

```
$ cat test.ctl
TABLE  PUBLIC.test
FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
```

    - The data in the table TEST is downloaded to the datafile as follows.

```
$ ls
test.ctl test.dat test.log
$ cat test.dat
1, L K H ,10
2,KIM
MM,A0
3, KIM S
J ,CD
```

The control file should be used with the OPTIONALLY ENCLOSED BY statement for the data including the white space to maintain the downloaded data and the uploaded data as same.

- The control file using OPTIONALLY ENCLOSED BY statement
    - The following is the contents of the control file which is created to download the table data.

```
$ cat test.ctl
TABLE  PUBLIC.test
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
```

```
LINES TERMINATED BY '\n'
```

- The data in the table TEST is downloaded to the datafile as follows.

```
$ ls
test.ctl test.dat test.log
$ cat test.dat
"1"," L K H ","10"
"2","KIM
MM","A0"
"3"," KIM S
J ","CD"
```

Time Related Data Type

DATE, TIME, TIME WITH TIME ZONE, TIMESTAMP, TIMESTAMP WITH TIME ZONE are output in the property default format when downloaded.

The following is each data type of DATE, TIME WITH TIME ZONE, TIMESTAMP WITH TIME ZONE.

```
$ gsql test test
gSQL> SELECT PROPERTY_VALUE, INIT_VALUE FROM V$PROPERTY WHERE PROPERTY_NAME LIKE
'NLS_DATE_FORMAT';
PROPERTY_VALUE                   INIT_VALUE
-------------------------------- --------------------------------
YYYY-MM-DD                       YYYY-MM-DD
1 row selected.
gSQL> SELECT PROPERTY_VALUE, INIT_VALUE FROM V$PROPERTY WHERE PROPERTY_NAME LIKE
'NLS_TIME_WITH_TIME_ZONE_FORMAT';
PROPERTY_VALUE                   INIT_VALUE
-------------------------------- --------------------------------
HH24:MI:SS.FF6 TZH:TZM           HH24:MI:SS.FF6 TZH:TZM
1 row selected.
gSQL> SELECT PROPERTY_VALUE, INIT_VALUE FROM V$PROPERTY WHERE PROPERTY_NAME LIKE
'NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT';
PROPERTY_VALUE                   INIT_VALUE
-------------------------------- --------------------------------
YYYY-MM-DD HH24:MI:SS.FF6 TZH:TZM YYYY-MM-DD HH24:MI:SS.FF6 TZH:TZM
1 row selected.
```

The following is the structure and data of the table to be downloaded.

```
$ cat test.sql
CREATE TABLE TEST ( I1 INTEGER, I2 DATE, I3 TIME WITH TIME ZONE, I4 TIMESTAMP WITH TIME ZONE
);
INSERT INTO TEST VALUES ( 1, '1999-12-31', '01:01:01', '1999-12-31 01:01:01.789 -8:00' );
INSERT INTO TEST VALUES ( 2, '2000-01-01', '23:12:12', '2000-01-01 23:12:06.0 +8:00' );
INSERT INTO TEST VALUES ( 3, '2000-12-31', '23:12:12', '2000-12-31 23:12:12.0 -8:00' );
COMMIT;
$ gsql test test
gSQL> SELECT * FROM T1;
I1 I2         I3                     I4
-- ---------- ---------------------- ---------------------------------
 1 1999-12-31 01:01:01.000000 +09:00 1999-12-31 01:01:01.789000 -08:00
 2 2000-01-01 23:12:12.000000 +09:00 2000-01-01 23:12:06.000000 +08:00
 3 2000-12-31 23:12:12.000000 +09:00 2000-12-31 23:12:12.000000 -08:00
3 rows selected.
```

The following is the contents of the control file which is created to download the table data.

```
$ cat test.ctl
TABLE  PUBLIC.test
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
```

The data in the table TEST is downloaded to the datafile as follows.

```
$ gloader test test -e -c test.ctl -d test.dat
 COMPLETED IN EXPORTING TABLE: PUBLIC.TEST, 3 RECORDS
$ ls
test.ctl test.dat test.log
$ cat test.dat
"1","1999-12-31 00:00:00","01:01:01.000000 +09:00","1999-12-31 01:01:01.789000 -08:00"
"2","2000-01-01 00:00:00","23:12:12.000000 +09:00","2000-01-01 23:12:06.000000 +08:00"
"3","2000-12-31 00:00:00","23:12:12.000000 +09:00","2000-12-31 23:12:12.000000 -08:00"
```

Downloading in Binary File

Simple Download

The following is the structure and data of the table to be downloaded.

```
$ cat test.sql
CREATE TABLE TEST ( I1 INTEGER PRIMARY KEY, I2 VARCHAR(10), I3 VARBINARY(10) );
INSERT INTO TEST VALUES( 1, 'LKH', X'10' );
INSERT INTO TEST VALUES( 2, 'KMM', X'A0' );
INSERT INTO TEST VALUES( 3, 'ksj', X'CD' );
COMMIT;
$ gsql test test
gSQL> SELECT * FROM TEST;
I1 I2  I3
-- --- --
 1 LKH 10
 2 KMM A0
 3 ksj CD
3 rows selected.
```

The following is the contents of the control file which is created to download the table data.

```
$ cat test.ctl
TABLE  PUBLIC.test
FIELDS TERMINATED BY ','
```

The data in the table TEST is downloaded through gloader as follows.

```
$ gloader test test --export --control test.ctl --data test.dat --format binary
 COMPLETED IN EXPORTING TABLE: PUBLIC.test, 3 RECORDS
```

The following file is generated after executing gloader.

```
$ ls
test.ctl test.dat test.log
```

A user can not directly check or edit the binary file.

Complex Download

The following is the structure and data of the table to be downloaded.

```
$ gsql test test
gSQL>\desc TEST
COLUMN_NAME TYPE                        IS_NULLABLE
----------- --------------------------- -----------
```

```
I1          NUMBER(10,0)               TRUE
I2          DATE                       TRUE
I3          TIME(6) WITH TIME ZONE     TRUE
I4          TIMESTAMP(6) WITH TIME ZONE TRUE
gSQL> SELECT COUNT(*) FROM TEST;
COUNT(*)
--------
 1572864
1 row selected.
```

The following is the contents of the control file which is created to download the table data.

```
$ cat test.ctl
TABLE  PUBLIC.test
FIELDS TERMINATED BY ','
```

- Downloading in a single file
    - The data in the table TEST is downloaded through gloader as follows.

```
$ gloader test test --export --control test.ctl --data test.dat --format binary
loaded 1000 records into PUBLIC.TEST
loaded 2000 records into PUBLIC.TEST
... Ellipsis ...
loaded 1571000 records into PUBLIC.TEST
loaded 1572000 records into PUBLIC.TEST
 COMPLETED IN EXPORTING TABLE: PUBLIC.TEST, TOTAL 1572864 RECORDS
```

    - The following file is generated after executing gloader.

```
$ ll test.*
-rw-r--r-- 1 test test       71 2014-08-28 12:13 t1.ctl
-rw-r--r-- 1 test test 59930624 2014-08-28 12:49 t1.dat
-rw-r--r-- 1 test test      159 2014-08-28 12:49 t1.log
```

- Downloading in multiple files
    - The data in the table TEST is downloaded through gloader as follows.

```
$ gloader test test --export --control test.ctl --data test.dat --format binary --filesize
31461376
loaded 1000 records into PUBLIC.TEST
loaded 2000 records into PUBLIC.TEST
```

```
... Ellipsis ...
loaded 1571000 records into PUBLIC.TEST
loaded 1572000 records into PUBLIC.TEST
 COMPLETED IN EXPORTING TABLE: PUBLIC.TEST, TOTAL 1572864 RECORDS
```

- The following file is generated after executing gloader.

```
$ ll test.*
-rw-r--r-- 1 test test       71 2014-08-28 12:13 test.ctl
-rw-r--r-- 1 test test 31461376 2014-08-28 13:02 test.dat
-rw-r--r-- 1 test test 28474880 2014-08-28 13:02 test.dat.001
-rw-r--r-- 1 test test      159 2014-08-28 12:49 test.log
```

Uploading Data

Uploading Text File

Simple Upload

The following is the table to be uploaded.

```
gSQL> \DESC TEST
COLUMN_NAME TYPE                        IS_NULLABLE
----------- --------------------------- -----------
I1          NUMBER(10,0)                TRUE
I2          DATE                        TRUE
I3          TIME(6) WITH TIME ZONE      TRUE
I4          TIMESTAMP(6) WITH TIME ZONE TRUE
gSQL> SELECT * FROM TEST;
no rows selected.
```

The following is the datafile to be uploaded.

```
$ cat test.dat
1,1999-12-31 00:00:00,01:01:01.000000 +09:00,1999-12-31 01:01:01.789000 -08:00
2,2000-01-01 00:00:00,23:12:12.000000 +09:00,2000-01-01 23:12:06.000000 +08:00
3,2000-12-31 00:00:00,23:12:12.000000 +09:00,2000-12-31 23:12:12.000000 -08:00
```

The datafile is uploaded with gloader and the result is output as follows.

```
$ gloader test test --import --control test.ctl --data test.dat
 COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 3 RECORDS, SUCCEEDED 3 RECORDS
$ gsql test test
gSQL> select * from test;
I1 I2         I3                     I4
-- ---------- ---------------------- ---------------------------------
 1 1999-12-31 01:01:01.000000 +09:00 1999-12-31 01:01:01.789000 -08:00
 2 2000-01-01 23:12:12.000000 +09:00 2000-01-01 23:12:06.000000 +08:00
 3 2000-12-31 23:12:12.000000 +09:00 2000-12-31 23:12:12.000000 -08:00
3 rows selected.
```

Whitespace Character

The following is the table to be uploaded.

```
gSQL> \DESC TEST
COLUMN_NAME TYPE                 IS_NULLABLE
----------- -------------------- -----------
I1          NUMBER(10,0)         TRUE
I2          CHARACTER VARYING(10) TRUE
I3          BINARY VARYING(10)   TRUE
gSQL> SELECT * FROM TEST;
no rows selected.
```

- The datafile downloaded without using OPTIONALLY ENCLOSED BY statement in the control file
    - The datafile is downloaded without using OPTIONALLY ENCLOSED BY statement in the control fil e as follows. (Refer to **Downloading in Text File**.)

```
$ cat test.dat
1, L K H ,10
2,KIM
MM,A0
3, KIM S
J ,CD
```

    - The datafile is uploaded with gloader and the result is output as follows. Even though New Line ('\n') exists in the data of the second and third records, but a qualifier notifying the start and end of the column data is not set, so it is recognized as a row identifier.

```
$ gloader test test --import --control test.ctl --data test.dat
 COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 5 RECORDS, SUCCEEDED 3 RECORDS
$ gsql test test
gSQL> select * from test;
I1 I2     I3
-- ------ ----
 1 L K H  10
 2 KIM    null
 3 KIM S  null
3 rows selected.
```

- The datafile downloaded using OPTIONALLY ENCLOSED BY statement in the control file
    - The datafile is downloaded by using the OPTIONALLY ENCLOSED BY statement of the control file

as follows. (Refer to **Downloading in Text File**.) Differently from the result above, New Line ('\n') i s treated as a part of data in the result below.

```
$ cat test.dat
"1"," L K H ","10"
"2","KIM
MM","A0"
"3"," KIM S
J ","CD"
```

- The datafile is uploaded with gloader and the result is output as follows.

```
$ gloader test test --import --control test.ctl --data test.dat
 COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 5 RECORDS, SUCCEEDED 3 RECORDS
$ gsql test test
gSQL> select * from test;
I1 I2      I3
-- ------- --
 1  L K H  10
 2 KIM     A0
   MM
 3  KIM S  CD
   J
3 rows selected.
```

Uploading Binary File

Simple Upload

The following is the structure of the table to be uploaded.

```
gSQL> \DESC TEST
COLUMN_NAME TYPE                 IS_NULLABLE
----------- -------------------- -----------
I1          NUMBER(10,0)         TRUE
I2          CHARACTER VARYING(10) TRUE
I3          BINARY VARYING(10)    TRUE
gSQL> SELECT * FROM TEST;
no rows selected.
```

The following is the content of the control file written to download the table data.

```
$ cat test.ctl
TABLE  PUBLIC.test
FIELDS TERMINATED BY ','
```

The following is a file to be uploaded with gloader. *test.dat* was already downloaded when **Downloading in Binary File**.

```
$ ls
test.dat
```

The datafile is uploaded to the table TEST and the result is output as follows.

```
$ gloader test test --import --control test.ctl --data test.dat --format binary
 COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 3 RECORDS, SUCCEEDED 3 RECORDS
$ gsql test test
gSQL> SELECT * FROM TEST;
I1 I2  I3
-- --- --
 1 LKH 10
 2 KMM A0
 3 ksj CD
3 rows selected.
```

Complex Upload

The following is the structure of the table to be uploaded.

```
gSQL> \DESC TEST
COLUMN_NAME TYPE                  IS_NULLABLE
----------- --------------------- -----------
I1          NUMBER(10,0)          TRUE
I2          CHARACTER VARYING(10) TRUE
I3          BINARY VARYING(10)    TRUE
gSQL> SELECT * FROM TEST;
no rows selected.
```

The following is the datafile to be uploaded.

```
$ ll test.*
 -rw-r--r-- 1 test test       71 2014-08-28 12:13 test.ctl
 -rw-r--r-- 1 test test 31461376 2014-08-28 13:02 test.dat
 -rw-r--r-- 1 test test 28474880 2014-08-28 13:02 test.dat.001
```

```
 -rw-r--r-- 1 test test      159 2014-08-28 12:49 test.log
```

When uploading multiple downloaded files with --filesize, gloader should be separately performed for ea ch datafile.

```
$ gloader test test --import --control test.ctl --data test.dat --format binary
 COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 860728 RECORDS, SUCCEEDED 2285000 RECORDS,
ERRORED 0 RECORDS
$ gloader test test --import --control test.ctl --data test.dat.001 --format binary
 COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 860728 RECORDS, SUCCEEDED 860728 RECORDS,
ERRORED 0 RECORDS
```

The upload result is as follows.

```
gSQL> select count(*) from test;
COUNT(*)
--------
 3145728
1 row selected.
```

Controlling Upload Unit

The data can be uploaded faster by using the options which is related to performance.
For more information, refer to **--array**, **--commit**, **--atomic**.

The following is the datafile with approximately 380,000 records.

```
$ ll test.dat
-rw-r--r-- 1 test test 75092480 2014-08-28 15:59 test.dat
```

The following is the control file which is used for uploading.

```
$ cat test.ctl
TABLE  PUBLIC.test
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
```

Array Binding and Commit Cycle

The following gloader command uploads records to be bound in 5,000 unit and uploads records to be co mmitted in 20,000 unit.

```
$ gloader test test --import --control test.ctl --data test.dat --array 5000 --commit 20000
loaded 5000 records into PUBLIC.TEST
loaded 10000 records into PUBLIC.TEST
loaded 15000 records into PUBLIC.TEST
... Ellipsis ...
loaded 3810000 records into PUBLIC.TEST
loaded 3815000 records into PUBLIC.TEST
loaded 3818244 records into PUBLIC.TEST
 COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 3818243 RECORDS, SUCCEEDED 3818243 RECORDS
```

The following is the result of executing gloader.

```
gSQL> SELECT I1, I2, I3 FROM TEST FETCH 3;
I1 I2 I3
-- ------- --
1 L K H 10
2 KIM A0
3 KIM S CD
3 rows selected.
gSQL> SELECT COUNT(*) FROM TEST;
COUNT(*)
--------
3818246
1 row selected.
```

Array Binding and Atomic Option

The following gloader commands uploads records to be bound in 5000 unit with atomic INSERT, and 1,0 00 records are failed.

```
$ gloader test test --import --control test.ctl --data test.dat --array 5000 --atomic
loaded 5000 records into PUBLIC.TEST
loaded 10000 records into PUBLIC.TEST
loaded 15000 records into PUBLIC.TEST
... Ellipsis ...
loaded 38175000 records into PUBLIC.TEST
 COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 3818243 RECORDS, SUCCEEDED 3817243 RECORDS
```

The following is the result of executing gloader.

```
gSQL> SELECT I1, I2, I3 FROM TEST FETCH 3;
I1 I2 I3
```

```
-- ------- --
1 L K H 10
2 KIM A0
3 KIM S CD
3 rows selected.
gSQL> SELECT COUNT(*) FROM TEST;
COUNT(*)
--------
3817243
1 row selected.
```

The following is the result for the cause of the upload failure and they are recorded in the log file. The upl oad is failed because the non-numeric data is stored in the first column of the first record.

```
$ cat test.log
Err Rec(1) Col(1): 22018(12006): data value is not a numeric literal
Err Rec(1) Col(-1): HY000(19041): Failed to atomic execution
Err Rec(1001) Col(1): 22018(12006): data value is not a numeric literal
Err Rec(1001) Col(-1): HY000(19041): Failed to atomic execution
COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 3818243 RECORDS, SUCCEEDED 3817243 RECORDS [
Start Time: 2014-8-28 16:46:42 End Time: 2014-8-28 16:46:52 Taken Time: 10084582 micro-sec ]
```

The following is the result of the records which failed to upload and they are recorded in the bad file. 100 0 records were stored because it is uploaded in 500 array units.

```
"s1"," L K H ","10"
"2","KIM","A0"
"3"," KIM S J ","CD"
... Ellipsis ...
```

Array, commit options do not affect the execution result, but success or failure of INSERT in the atomic operation is treated in an array unit, so if a record is failed to upload, all records in a unit to which the records belong are treated as INSERT failure.
The cause of the first failed records of the array is recorded in the log file and the causes for fail ed record later is not recorded.

Parallel Upload

gloader improves the performance of GOLDILOCKS by dividing the operation into parts and uploading th em in thread unit. (Refer to --**parallel**.)

The following is the datafile with approximately 380,000 records.

```
$ ll test.dat
-rw-r--r-- 1 test test 75092480 2014-08-28 15:59 test.dat
```

The following is the control file which is used for uploading.

```
$ cat test.ctl
TABLE  PUBLIC.test
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
```

The following gloader commands uploads records which is bound to a thread performing four uploads. It is uploaded in 5000 unit with INSERT, and 100 records are failed.

```
$ gloader test test --import --control test.ctl --data test.dat --array 5000 --parallel 4
loaded 5000 records into PUBLIC.TEST
loaded 10000 records into PUBLIC.TEST
loaded 15000 records into PUBLIC.TEST
... Ellipsis ...
loaded 3810000 records into PUBLIC.TEST
loaded 3815000 records into PUBLIC.TEST
 COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 3818243 RECORDS, SUCCEEDED 3818143 RECORDS
```

The following is the result of executing gloader.

```
gSQL> SELECT I1, I2, I3 FROM TEST FETCH 3;
I1 I2 I3
-- ------- --
1 L K H 10
2 KIM A0
3 KIM S CD
3 rows selected.
gSQL> SELECT COUNT(*) FROM TEST;
COUNT(*)
--------
3818143
1 row selected.
```

The cause of failure during the upload is recorded in the log file as follows.

```
$ cat test.log
Err Rec(1) Col(1): 22018(12006): data value is not a numeric literal
Err Rec(3) Col(1): 22018(12006): data value is not a numeric literal
Err Rec(5) Col(1): 22018(12006): data value is not a numeric literal
... Ellipsis ...
Err Rec(1001) Col(1): 22018(12006): data value is not a numeric literal
... Ellipsis ...
COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 3818243 RECORDS, SUCCEEDED 3818143 RECORDS [
Start Time: 2014-8-28 16:46:42 End Time: 2014-8-28 16:46:52 Taken Time: 10084582 micro-sec ]
```

The following is a result of failed records recorded on the bad file during the upload. 100 records are stor ed.

```
"s1"," L K H ","10"
"s2","KIM","A0"
"s3"," KIM S J ","CD"
... Ellipsis ...
```

When uploading the data with multiple threads, the records in which errors occur after INSERT are stored in the log file and bad file. In this case, the order of records may not be as same as th e order in the data files.

Troubleshooting for Uploading

The record upload failure may occur by various causes. The failed record is stored in the bad file, and the i nformation about the cause of failure is stored in the log file.

Failure due to Redundant Constraint

The redundant data is already in the constrained table to be uploaded. (primary k ey or unique index)

The following is the structure of the table to be uploaded.

```
gSQL>\desc T1
COLUMN_NAME TYPE                  IS_NULLABLE
----------- --------------------- -----------
I1          NUMBER(10,0)          TRUE
I2          CHARACTER VARYING(10) TRUE
I3          BINARY VARYING(10)    TRUE
gSQL> SELECT * FROM T1;
I1 I2  I3
-- --- --
 1 LKH 10
 2 KMM A0
 3 ksj CD
3 rows selected.
```

The following is the datafile to be uploaded.

```
$ cat t1.dat
"1","LKH"
"4","SOS"
"5","OKO"
```

The followings are the upload result by using gloader, and the created log file and bad file.
The upload is failed because the record violated a primary key constraint.

```
$ gloader test test -i -c t1.ctl -d t1.dat
 COMPLETED IN IMPORTING TABLE: PUBLIC.T1, TOTAL 3 RECORDS, SUCCEEDED 2 RECORDS
$
$ cat t1.log
```

```
Err Rec(1) Col(-1): 40002(16057): unique constraint (PUBLIC.T1_PRIMARY_KEY) violated
COMPLETED IN IMPORTING TABLE: PUBLIC.T1, TOTAL 3 RECORDS, SUCCEEDED 2 RECORDS [ Start Time:
2014-8-26 16:19:51 End Time: 2014-8-26 16:19:51 Taken Time: 15455 micro-sec ]
$
$ cat t1.bad
"1","LKH"
```

Failure due to Date/time Format

The format can be set for the data type such as DATE, TIME, TIME WITH TIME ZONE, TIMESTAMP, TIMES TAMP WITH TIME ZONE, and it may cause the failure of upload using gloader.

The following is the table to be uploaded.

```
gSQL> \DESC T1
COLUMN_NAME TYPE IS_NULLABLE
----------- ------------------------- -----------
I1 NUMBER(10,0) TRUE
I2 TIMESTAMP(6) WITH TIME ZONE TRUE
gSQL> SELECT * FROM T1;
no rows selected.
```

The format of the TIMESTAMP WITH TIME ZONE on the server is as follows.

```
gSQL> SELECT PROPERTY_VALUE, INIT_VALUE FROM V$PROPERTY WHERE PROPERTY_NAME LIKE
'NLS_TIMESTAMP_WITH_TIME_ZONE_FORMAT';
PROPERTY_VALUE                    INIT_VALUE
--------------------------------- ---------------------------------
YYYY-MM-DD HH24:MI:SS.FF6 TZH:TZM YYYY-MM-DD HH24:MI:SS.FF6 TZH:TZM
```

The following is the datafile to be uploaded.

```
$ cat t1.dat
"4","20000108 00:00:00"
"5","20000108 04:05:06"
"6","20000108 04:05:06"
```

The following is the result of executing upload by using gloader.

```
$ gloader test test -i -c t1.ctl -d t1.dat


 COMPLETED IN IMPORTING TABLE: PUBLIC.T1, TOTAL 3 RECORDS, SUCCEEDED 0 RECORDS
```

The cause and records of upload failure are stored in the log file and the bad file as follows.

```
$ cat t1.log
Err Rec(1) Col(2): HY000(12136): literal does not match format string
Err Rec(2) Col(2): HY000(12136): literal does not match format string
Err Rec(3) Col(2): HY000(12136): literal does not match format string
COMPLETED IN IMPORTING TABLE: PUBLIC.T1, TOTAL 3 RECORDS, SUCCEEDED 0 RECORDS [ Start Time:
2014-8-27 13:56:29 End Time: 2014-8-27 13:56:29 Taken Time: 20670 micro-sec ]
$ cat t1.bad
"4","20000108 00:00:00"
"5","20000108 04:05:06"
"6","20000108 04:05:06"
```

Troubleshooting

The problem occurs when the data type format of the datafile and that of the data used on the server are different. The data type format of the datafile should be set to solve the problem. In this case, .odbc.ini is used.

The data type format is set in .odbc.ini file as follows.

```
$ cat .odbc.ini
[GOLDILOCKS]
HOST = 127.0.0.1
PORT = 21123
DATE_FORMAT = YYYYMMDD
TIME_FORMAT = HH24MISS
TIME_WITH_TIME_ZONE_FORMAT = HH24MISS TZHTZM
TIMESTAMP_FORMAT = YYYYMMDD HHMISS
TIMESTAMP_WITH_TIME_ZONE_FORMAT = YYYYMMDD HH:MI:SS
```

The following is the result of setting .odbc.ini and performing gloader again.

```
$ gloader test test -i -c t1.ctl -d t1.dat
 COMPLETED IN IMPORTING TABLE: PUBLIC.T1, TOTAL 3 RECORDS, SUCCEEDED 3 RECORDS
```

- The data type format set in .odbc.ini is applied to the entire column, and it can not be separ ately set.
- If the data type format is set in .odbc.ini, it is also applied when downloading to gloader.

Failure Due to Lack of Capacity

gloader is performed and failed as follows.

```
$ gloader test test -i -c t1.ctl -d t2.dat
loaded 4000 records into PUBLIC.T1
COMPLETED IN IMPORTING TABLE: PUBLIC.T1, TOTAL 4200 RECORDS, SUCCEEDED 0 RECORDS
```

It is failed due to lack of the space for datafile in the tablespace.

```
$ cat t1.log
Err Rec(1) Col(-1): HY000(14015): there is no extendible datafile in tablespace 'MEM_DATA_TBS'
Err Rec(2) Col(-1): HY000(14015): there is no extendible datafile in tablespace 'MEM_DATA_TBS'
... Ellipsis ...
COMPLETED IN IMPORTING TABLE: PUBLIC.T1, TOTAL 4200 RECORDS, SUCCEEDED 0 RECORDS [ Start Time:
2014-8-27 15:7:28 End Time: 2014-8-27 15:7:29 Taken Time: 317885 micro-sec ]
```

Troubleshooting

The datafile of a tablespace should be extended or added to solve the problem.
For more information, refer to **ALTER TABLESPACE**.

Datafile Analysis Failure

The field terminator, the qualifier, and the line terminator described in the control file may be different in the text datafile because of misuse.

The following is the structure of the table object to be uploaded.

```
gSQL>\desc T1
COLUMN_NAME TYPE                  IS_NULLABLE
----------- --------------------- -----------
I1          NUMBER(10,0)          TRUE
I2          CHARACTER VARYING(10) TRUE
I3          BINARY VARYING(10)    TRUE
```

The following is the control file to be uploaded.

```
$ cat t1.ctl
TABLE  T1
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
```

The following is the datafile to be uploaded.

```
$ cat t1.dat
'1',"LKH","aa"
"2","SOS"","aa"
"5","OKO","00"
```

The following is the result of executing upload by using gloader.

```
$ gloader test test --import --control t1.ctl --data t1.dat
 COMPLETED IN IMPORTING TABLE: PUBLIC.T1, TOTAL 3 RECORDS, SUCCEEDED 1 RECORDS
```

The cause and records of upload failure are as follows.

```
$ cat t1.log
Err Rec(1) Col(1): 22018(12006): data value is not a numeric literal
COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 3 RECORDS, SUCCEEDED 1 RECORDS [ Start Time:
2014-8-28 18:6:13 End Time: 2014-8-28 18:6:13 Taken Time: 18679 micro-sec ]
$
$ cat t1.bad
"2","SOS"","aa"
'1',"LKH","aa"
```

Troubleshooting

The records fail to analyze the data due to qualifier or delimiter which are used improperly. Therefore, to solve this problem, the field terminator, the qualifier and the delimiter in the control file and the datafile s hould be checked, then the datafile should be edited based on the records of the bad file.

The records which are failed to parse in the data analysis process are stored in the bad file, not i n the log file.

Uploading to Another Database Whose Character Set Is Different

Adjusting the column size may be required when downloading/ uploading data in binary type between d atabases whose character sets are different one another.

The following is the structure of the table object to be uploaded.

```
gSQL>\desc T1
COLUMN_NAME TYPE                 IS_NULLABLE
----------- -------------------- -----------
```

```
I1          NUMBER(10,0)          TRUE
I2          CHARACTER VARYING(36) TRUE
```

The following is the table data, and it is a VARCHAR type, its size is 36 and is completely filled.

```
SELECT * FROM T1;
I1    I2
---- ------------------------------------
   1 일이삼사오육칠팔구십일이삼사오육칠팔
```

The following error occurs when downloading data above from UHC database then uploading to UTF8 d atabase which has the same schema.

```
$ gloader test test -i -f binary -T T1 -d t1.dup
ERR-HY000(42023): byte length of data greater than column length.
ERROR: FAILED TO IMPORT TABLE PUBLIC.T1
```

Troubleshooting

The data upload succeeds when adjusting the length of column I2 as follows.
Declare I2 as VARCHAR(18 CHAR), or declare I2 as VARCAR(54) size.

```
gSQL>\desc T1
COLUMN_NAME TYPE                       IS_NULLABLE
----------- -------------------------- -----------
I1          NUMBER(10,0)               TRUE
I2          CHARACTER VARYING(18 CHAR) TRUE
```

```
gSQL>\desc T1
COLUMN_NAME TYPE                  IS_NULLABLE
----------- --------------------- -----------
I1          NUMBER(10,0)          TRUE
I2          CHARACTER VARYING(54) TRUE
```

41.3 Control File Syntax

```
TABLE [schema_name.]table_name[domain_name.]
FIELDS TERMINATED BY 'Field Terminator'
[OPTIONALLY ENCLOSED BY 'Open Qualifier' [AND 'Close Qualifier']]
[LINES TERMINATED BY 'Line Terminator']
[Characterset characterset_name]
[RTRIM [ON¦OFF]]
[LTRIM [ON¦OFF]]
[WHERE="conditional statement"]
```

A control file describes only one table. Therefore, each item above should be described in the co ntrol file only once, and if it is described redundantly, then the control parsing error occurs.

CHARACTERSET

Syntax

```
CHARACTERSET characterset_name
```

Description

It refers to the character set of the datafile to be downloaded or uploaded.
If the character set is not specified, the UTF8 which is the default character set of CHARACTER_SET in GO LDILOCKS property is used.

Example

The following is an example of which the character set of the datafile to be uploaded or downloaded is t he ASCII code.

```
% cat sample.ctl
CHARACTERSET ASCII
```

TABLE

Syntax

```
TABLE table_name
TABLE schema_name.table_name
TABLE table_name@domain_name
TABLE schema_name.table_name@domain_name
```

Description

It specifies the name of table to which upload or download the data, or specifies the schema name to which the table belongs, or the domain name.
The domain name can be used only for data download, and it downloads only the data of the corresponding member.
The string or double-quoted (") string is used in the table name.
If the table name is given as the command row argument in gloader as well, then the value of the command row argument takes precedence over the value in the control file.

Examples

The following is an example of specifying only the table name.

```
% cat sample.ctl
TABLE lineitem
```

The following is an example of specifying the schema PUBLIC and the table name. It has the same meaning as the example above.

```
% cat sample.ctl
TABLE PUBLIC.lineitem
```

The following is an example of specifying the schema PUBLIC and the table name of G1N1 member. It has the same meaning as the example above.

```
% cat sample.ctl
TABLE PUBLIC.lineitem@G1N1
```

The following is an example of specifying the table name created by the delimited identifier.

```
gSQL> CREATE TABLE "Tab*&^" ( id INTEGER );
```

```
% cat sample.ctl
TABLE "Tab*&^"
```

FIELDS TERMINATED BY

Syntax

```
FIELDS TERMINATED BY 'Field Terminator'
```

Description

The field terminator is used as a delimiter between columns in the record data of the data file. Setting the field terminator in the control file can not be omitted.
The field terminator can be set with one or more strings, and should not be redundant with a qualifier or line terminator, and it should not use the subset string.
There is not any constraintfor the string to be set as a field terminator. However, \ or % should be added to in front of each n, t, r when setting NEW LINE or TAB, CARRIAGE RETURN character.
If the field terminator is given as the command row argument in gloader as well, then the value of the co mmand row argument takes precedence over the value in the control file.

Example

The following is an example of setting the field terminator as COMMA and NEW LINE.

```
% cat sample.ctl
FIELDS TERMINATED BY ',\n'
```

```
% cat sample.ctl
FIELDS TERMINATED BY ',%n'
```

OPTIONALLY ENCLOSED BY

Syntax

```
OPTIONALLY ENCLOSED BY 'Open Qualifier' [AND 'Close Qualifier']
```

Description

Qualifier is used as a delimiter to represent the start and end of the column.
Qualifier is a single character, and the first qualifier is a open qualifier, the last qualifier is a close qualifier.
The same characters can be used to set an open qualifier and a close qualifier, or the different characters can be used to represent the start and end of the column.
Characters set as a qualifier can not be used in a field terminator nor in a line terminator.
If a close qualifier literally belongs to a column data, two close qualifiers are used to represent a single valid data.
If an open qualifier is set omitting a close qualifier, then characters as same as those in an open qualifier is set in a close qualifier.

If OPTIONALLY ENCLOSED BY statement does not exist, the column data is distinguished by the field terminator.
If the qualifier is given as the command row argument in gloader as well, then the value of the command row argument takes precedence over the value in the control file.

If OPTIONALLY ENCLOSED BY statement does not exist, the results of the uploading and downloading data may be different. (Refer to **Troubleshooting Uploading**.)

Example

The following is an example of using double quotes (") and a single quote (') as a open qualifier and a close qualifier each.

```
% cat sample.ctl
OPTIONALLY ENCLOSED BY '"' AND "'"
```

LINES TERMINATED BY

Syntax

```
LINES TERMINATED BY 'Line Terminator'
```

The line terminator is used as a delimiter between records in the data file.
The line terminator can be set with one or more strings, and should not be redundant with a qualifier or a field terminator, and it should not use the subset string.

When omitting the line terminator setting, then NEW LINE ('\n' or '%n') is used by default.
If the line terminator is given as the command row argument in gloader as well, then the value of thecom mand row argument takes precedence over the value in the control file.

Description

• Differently from Unix, CARRIAGE RETURN and NEW LINE are written together instead of a si ngle NEW LINE in the data file exported from Windows OS. When importing data by using this data file, LINES TERMINATED BY in the control file should be explicitly set like as '\r\n' so that t he data is normally imported.

• It is recommended to set a field terminator and a line terminator with a different string each. The more mutual string including the first character exist the poorer the import performance du e to the internal comparing. In other words, when a field terminator and a line terminator are s et with different strings each, then the shorter the string the better the performnace.

Example

The following is an example of using '^^\t\r\n' as a line terminator

```
% cat sample.ctl
LINES TERMINATED BY '^^\t\r\n'
```

LTRIM

Syntax

```
LTRIM ON¦OFF
```

Description

It determines a left trim.
The default value is OFF, and when it is set to OFF, then the left WHITESPACE is considered data.
When it is set to ON, then the left WHITESPACE is ignored.

> OPTIONALLY ENCLOSED BY is applied only when the syntax does not exist.
> If OPTIONALLY ENCLOSED BY is used and the column is enclosed with delimiters in the data file, then RTRIM and LTRIM is OFF.

Example

The following is an example of setting LTRIM to ON.

```
% cat sample.ctl
LTRIM ON
```

RTRIM

Syntax

```
RTRIM ON¦OFF
```

Description

It determines a right trim.
The default value is OFF, and when it is set to OFF, then the right WHITESPACE is considered data.
When it is set to ON, then the right WHITESPACE is ignored.

> OPTIONALLY ENCLOSED BY is applied only when the syntax does not exist.
> If OPTIONALLY ENCLOSED BY is used and the column is enclosed with delimiters in the data file, then RTRIM and LTRIM is OFF.

Example

The following is an example of setting RTRIM to ON.

```
% cat sample.ctl
RTRIM ON
```

WHERE

Syntax

```
WHERE="conditional_statement"
```

Description

It uses a conditional clause when downloading data.

Example

The following is an example of using WHERE.

```
% cat sample.ctl
WHERE="I2 > 3"
```

41.4 gloader Argument References

Usage

```
$ gloader --help
Usage
    gloader user password mode data [control] [format] [options]
    user                user name
    password            password
mode: gloader's mode.
  --export              export data
  --import              import data
data:
  --data                data file
options:
  --control             control file
  --format              file format(text¦binary, Default text)
  --log                 log file
  --bad                 bad file
  --dsn                 dsn string
  --array               number of rows in bind array(Default 1000)
  --filesize            max file size
  --commit              number of commit unit(Default 5000)
  --comment             commenting on commit
  --atomic              use atomic function
  --parallel            use parallel in import
  --propagation         enabling or disabling a redo log propagation(ON¦OFF)
  --errors              number of error count to allow(Default 100)
  --AsTIMESTAMP         bind DATE as TIMESTAMP
  --buffered            buffered disk io(Default direct io)
  --tablename           [schema_name.]table_name[@domain_name]
  --fieldterm           field terminator
  --lineterm            line terminator
  --qualifier           qualifier(column data encloser)
  --where               export only rows selected by given WHERE condition
  --group-id            importing distributed data by group id using global connections in a
clustered environment
  --directio-size       direct io size(Default 512)
```

```
--no-copyright        suppresses the display of the banner
--silent              suppresses the display of the result message
--help                print help message
```

Mandatory Argument

The arguments are entered in an order of username and password to connect to the database.
gloader operation mode, control file, datafile are entered as arguments.

Argument	Description
user_name	It is the user name. The maximum length of user name is 128.
password	It is password. The maximum length of password is 128.

--export

Description

It specifies for gloader to download the data in the database.

Example

```
$ gloader test test --export -c sample.ctl -d sample.dat
```

--import

Description

It specifies for gloader to upload the data in the database.

Example

```
$ gloader test test --import -c sample.ctl -d sample.dat
```

--control

Description

It specifies the control file path.

If --tablename is given as an argument, the control file can be omitted. The table name is mandatory to e

xecute gloader, so the table name should be given via a control file or an argument. If the table name is n ot given as an argument, TABLE item should be set via a control file.

When a control file is omitted, a field terminator, a qualifier, a line terminator can be given as an argume nt. If these delimiters are not given as an argument, then delimiters in CSV form is used by default.

Example

The following is an example of using a control file whose name is sample.ctl.

```
$ gloader test test -i --control sample.ctl -d sample.dat
```

--data

Description

It specifies the data file path to download or upload.

Example

The following is an example of uploading a control file whose name is sample.dat.

```
$ gloader test test -i -c sample.ctl --data sample.dat
```

Optional Argument

--tablename

Description

It specifies the table name in [Schemaname.]Tablename[@domain_name] form.

The table name should be given via setting TABLE of an argument or a control file. In other words, if a co ntrol file is omitted, the table name argument is mandatory.
If a tablename argument is given when TABLE item of a control file is set, then argument value takes prec edence over the value in the control file.

Example

The following is an example of uploading data in CSV form by giving a tablename argument because the control file argument is omitted

```
$ gloader test test -i --tablename PUBLIC.T1 --data sample.dat
```

The following is an argument of omitting a tablename argument because TABLE item is omitted in the co ntrol file.

```
$ cat sample.ctl | grep TABLE
  TABLE T1
$ gloader test test -i -c sample.ctl --data sample.dat
```

--format

Description

It specifies the datafile format.
The datafile format may be text or binary, and the default value is text when the format is not set.

Example

The following is an example of downloading the data to sample.dat in binary form.

```
$ gloader test test --export --control sample.ctl --data sample.dat --format binary
```

The followings are sample.dat and sample.log which are created after executing gloader as above.

```
$ ls
sample.ctl sample.dat sample.log
```

--log

Description

It specifies the logfile path.
gloader records the error and results occurred during the execution.
If the file name is not specified, the name is created as same as the datafile by changing its extension to l og.

Example

The following is an example of when the name of the log file is specified by using --log option.

```
$ gloader test test --export --control sample.ctl --data sample.dat --log SAMPLE.log
```

The followings are sample.dat, SAMPLE.log which are created after executing gloader as above.

```
$ ls
sample.ctl sample.dat SAMPLE.log
```

The following is an example of when the name of the log file is not specified by using --log option.

```
$ gloader test test --export --control sample.ctl --data sample.dat
```

The followings are sample.dat, sample.log which are created after executing gloader as above.

```
$ ls
sample.ctl sample.dat sample.log
```

--bad

Description

It specifies the bad file path.
It is valid only when gloader performs the import operation. Rows which can not be uploaded due to an error are stored in the bad file.
If the file name is not specified, the name is created as same as the datafile by changing its extension to bad.

Example

The following is an example of when the name of the bad file is specified by using --bad option.

```
$ gloader test test --import --control sample.ctl --data sample.dat --bad SAMPLE.bad
```

The followings are sample.dat, sample.log, SAMPLE.bad which are created after executing gloader as above.

```
$ ls
SAMPLE.bad sample.ctl sample.dat sample.log
```

The following is an example of when the name of the bad file is not specified by using --bad option.

```
$ gloader test test --export --control sample.ctl --data sample.dat
```

The followings are sample.dat, sample.log, sample.bad which are created after executing gloader as abov

e.

```
$ ls
sample.bad sample.ctl sample.dat sample.log
```

--dsn

Description

It is the dsn string. The maximum length is 128.
It is used to specify the format of the time-related data type represented in the data file to be downloaded or uploaded.
It is used to specify the server when using gloadernet in the Client/ Server (C/S) environment.
For more information, refer to **odbc.ini File**.

Example

The following is an example of .odbc.ini described to use --dsn option.

```
$ cat .odbc.ini
[GOLDILOCKS]
HOST = 127.0.0.1
PORT = 21123
DATE_FORMAT = YYYYMMDD
TIME_FORMAT = HH24MISS
TIME_WITH_TIME_ZONE_FORMAT = HH24MISS TZHTZM
TIMESTAMP_FORMAT = YYYYMMDD HHMISS
TIMESTAMP_WITH_TIME_ZONE_FORMAT = YYYYMMDD HH:MI:SS
```

The following is an example of uploading the data to the goldilocks server by using --dsn option.

```
$ gloadernet test test --dsn goldilocks --import --control sample.ctl --data sample.dat
```

The following is an example of defining the format of time-related data type of the data file by using --dsn option, and then downloading data.

```
$ gloader test test --dsn goldilocks --export --control sample.ctl --data sample.dat
```

For more information, refer to **Failure due to date/time format**.

--array

Description

It specifies the number of rows to be bound when importing the data.
The amount of memory is required in proportion to the number of array. If not specified, 1,000 rows are used.
The option is applied only when uploading the data.

Example

The following is an example of uploading the data file in 2,000 records unit.

```
$ gloader test test --import --control sample.ctl --data sample.dat --array 2000
```

--filesize

Description

The maximum size of the file can be set when gloader downloads the data. If the amount of data exceeds the maximum size, the file is generated by adding the permutation number to the file extension.
If not specified, the maximum file size is unlimited, and the minimum is 31,461,376 (30 Mbytes).
The option is valid only for the binary type data file.

Example

The following is an example of executing the commands, then the file size exceeds the specified maximum, so new files are generated as results.

```
$ gloader test test --export --control sample.ctl --data sample.dat --filesize 31461376
```

```
$ls -al
-rw-r--r-- 1 test test 31461376 sample.dat
-rw-r--r-- 1 test test 31461376 sample.dat.001
-rw-r--r-- 1 test test  6715392 sample.dat.002
```

--commit

Description

While gloader uploads the data, the transaction is in the no commit state. The commit cycle may be set b
y using --commit option.
If not set, the default value is 5,000 (rows).

Example

The following is an example of using --commit option.

```
$ gloader test test --import --control sample.ctl --data sample.dat --commit 10000
```

--comment

Description

When gloader commits the transaction, it specifies the comment on the transaction.

Example

The following is an example of using --comment option.

```
$ gloader test test --import --control sample.ctl --data sample.dat --comment import_sample
```

--atomic

Description

It is the option for performing array INSERT, and it is useful when uploading the data.
The performance is faster than the existing array insert, because atomic array INSERT processes the insert statements as many as the size of array in a single transaction.

Example

The following is an example of using --atomic option.

```
$ gloader test test --import --control sample.ctl --data sample.dat --array 1000 --atomic
```

--parallel

Description

It specifies the number of threads for parallel processing.
It is useful only when gloader uploads the data, and the performance gets faster when the number of thr

eads is increased by adjusting --parallel option.
The performance gets faster by increasing the number of threads, but it is recommended to set the numb er of threads according to the operational environment.
The default value is 1, and the maximum value is 32.

Example

The following is an example of using eight threads when uploading data. Ten threads are operated toget her including threads analyzing the data files and read only threads, besides the eight uploading threads.

```
$ gloader test test --import --control sample.ctl --data sample.dat --parallel 8
```

--propagation

Description

It determines whether to propagate the upload transaction log to another replicated server.
It can be set to ON or OFF.

Example

The following is an example of which the upload transaction log is not propagated to another replicated server.

```
$ gloader test test --import --control sample.ctl --data sample.dat --propagation off
```

--errors

Description

It sets the number of errors permitted when gloader uploads the data.
If not set, 100 (rows) are used. If it is set to 0, the --errors option is ignored.
If it is smaller than the size of --array option, the number of permitted error is the number of array.

Example

The following is an example of using --errors option.

```
$ gloader test test --import --control sample.ctl --data sample.dat --errors 10000
```

--AsTIMESTAMP

Description

It sets the DATE type data stored in TIMESTAMP format to be operated in TIMESTAMP format for backw ard compatibility.
TIMESTAMP_FORMAT is also applied to the DATE type data when using the --AsTIMESTAMP option.

Example

The following is an example of using --AsTIMESTAMP option.

```
$ gloader test test --import --control sample.ctl --data sample.dat --AsTIMESTAMP
```

--buffered

Description

Only the datafile uses the buffered IO instead of the direct IO.
The log file and bad file use the buffered IO.

Example

The following is an example of using --buffered option.

```
$ gloader test test --import --control sample.ctl --data sample.dat --buffered
```

--fieldterm

Description

It provides a field terminator.
If it is set in a control file as well, then the fieldterm argument value is preferentially used.
% is added in front of n, r, t each for NEW LINE, CARRIAGE RETURN, TAB.
Characters which is used as a shell meta character such as ', ", ₩, & is not recommended to use.

Example

The following is an example of using --fieldterm option.

```
$ gloader test test --i -c sample.ctl -d sample.dat --fieldterm ",,,"
$ gloader test test --i -c sample.ctl -d sample.dat --fieldterm ,,,
```

```
$ gloader test test --i -c sample.ctl -d sample.dat --fieldterm ',,,'
```

--lineterm

Description

It provides a line terminator.
If it is set in a control file as well, then the lineterm argument value is preferentially used.
The detailed usage is as same as that of the fieldterm argument.

Example

The following is an example of using --lineterm option.

```
$ gloader test test --i -T PUBLIC.test -d sample.dat --lineterm ",,,"
$ gloader test test --i -T PUBLIC.test -d sample.dat --lineterm ,,,
$ gloader test test --i -T PUBLIC.test -d sample.dat --lineterm ',,,'
```

--qualifier

Description

It provides a qualifier which is to be added to the start and the end of the column data. Only a single char acter can be set as a qualifier.
If it is set in a control file as well, then the qualifier argument value is preferentially used.
The detailed usage is as same as that of the fieldterm argument.

Example

The following is an example of using --qualifier option.

```
$ gloader test test --i -T PUBLIC.test -d sample.dat --qualifier "¦"
$ gloader test test --i -T PUBLIC.test -d sample.dat --qualifier '"'
```

--where

Description

It downloads the data by setting a conditional clause for export operation.
If the where clause is also set in a control file, then the where argument value is preferentially used.

Example

The following is an example of using --where option.

```
$ gloader test test --i -T PUBLIC.test -d sample.dat --where "I2 > 4"
```

--group-id

Description

It directly uploads the data to the corresponding group after sorting the data by group when uploading t he data in the sharded table in the cluster environment.
If this option is used when uploading the data to the non-sharded table, the option is not valid.
This option is operated only in C/S environment, so it is valid only in gloadernet and it can be used only w hen uploading text files.

--group-id option uses **GLOBAL CONNECTION** of ODBC, so properties related to the **Data Source Configu ration** should be set.

Example

The following is odbc.ini configuration file to use the global connection.

```
$cat .odbc.ini
[GOLDILOCKS]
HOST=127.0.0.1
PORT = 22581
LOCALITY_AWARE_TRANSACTION=1
LOCATOR_DSN = LOCATOR
[LOCATOR]
LOCATOR_FILE=.locator.ini
```

The following is an example of using *--group-id* option in gloadernet.

```
$ gloadernet test test --i -T PUBLIC.test -d sample.dat --group-id
COMPLETED IN IMPORTING TABLE: PUBLIC.TEST, TOTAL 20 RECORDS, SUCCEEDED 20 RECORDS
```

The following is an example of an error which occurred due to using *--group-id* option in gloader.

```
$ gloader test test --i -T PUBLIC.test -d sample.dat --group-id
ERR-HY010(19009): Function sequence error : The function should be called only when the
SQL_ATTR_LOCALITY_AWARE_TRANSACTION connection attribute is set.
```

--directio-size

Description

gloader uses direct IO by default. The default value of direct IO is 512, but this size can be modified by using *--directio-size*. The value for the size should be the value of 2 powers of 512.

Example

The following is an example of modifying the direct IO size in gloader.

```
$ gloader test test --i -T PUBLIC.test -d sample.dat --directio-size 1024
```

--no-copyright

Description

It does not output the copyright and version.

Example

The following is the result of executing gloader with --no-copyright option.

```
$ gloader test test --import --control sample.ctl --data sample.dat --no-copyright
COMPLETED IN EXPORTING TABLE: PUBLIC.t1, 3 RECORDS
$
```

--silent

Description

It does not output the results of executing gloader.

Example

The following is the result of executing gloader with --silent option.

```
$ gloader test test --import --control sample.ctl --data sample.dat --silent
$
```

--help

Description

It displays the help messages.
For more information, refer to **Usage**.

42.

gdump

42.1 Overview of gdump

Definition

gdump is a utility provided by GOLDILOCKS, and it dumps the following binary files managed by the data base in text form.

- Control file
- Datafile
- Log file
- Incremental backup file
- Property file
- Commit log file
- Log buffer file
- Pending log buffer file

Argument

gdump usage is divided into the mandatory argument and the optional argument, and the optional arguments are used differently according to the file types to be dumped.

```
gdump file_type file_name [options]
```

Mandatory Argument

The mandatory argument is used in an order of the file type and and the file path.

```
file_type:  CONTROL ¦ LOG ¦ DATA ¦ PROPERTY ¦ BACKUP ¦ COMMIT_LOG ¦ LOG_BUFFER ¦ PEND_BUFFER
file_name:  file name to dump
```

file_type Argument

- CONTROL: It is a control file. It is located in *<GOLDILOCKS_DATA>/wal*, and its file extension is *.ctl*.
- Log: It is a log file (online/archived redo log file). It is located in *<GOLDILOCKS_DATA>/wal* or in *<GOLDILOCKS_DATA>/archive_log*, and its file extension is *.log*.
- DATA: It is a datafile. It is located in *<GOLDILOCKS_DATA>/db*, and its file extension is *.dbf*.
- PROPERTY: It is a binary property file. It is located in *<GOLDILOCKS_DATA>/conf*, and the default file name is *goldilocks.properties.binary*.

- BACKUP: It is an incremental backup file. It is located in *<GOLDILOCKS_DATA>/backup*, and its file extension is *.inc*.
- COMMIT_LOG: It is a commit.log file located in *<GOLDILOCKS_DATA>/wal*.
- LOG_BUFFER: It is a file of which a log buffer located in the memory is stored. The log buffer is stored in a file by executing gsyncher.
- PEND_BUFFER: It is a file of which a pending log buffer located in the memory is stored. The pending log buffer is stored in a file by executing gsyncher.


An attempt to the dump of broken control file fails.

Optional Argument

```
-S  --silent                                silent
   file_type = CONTROL
      -s, --section  sys ¦ log ¦ db ¦ backup ¦ all  dump controlfile section(default all)
   file_type = DATA
      -h, --header                          dump datafile header
      -n, --number   INTEGER ( >= 0 )       (log or page) sequence number
      -f, --fetch    INTEGER ( >= 1 )       dump as much as count
   file_type = LOG
      -h, --header
      -n, --number   INTEGER ( >= 0 )       (log or page) sequence number
      -o, --offset   INTEGER                offset of lsn(usable if set lsn)
      -f, --fetch    INTEGER ( >= 1 )       dump as much as count
      -a, --all
   file_type = BACKUP
      -b, --body     header ¦ all           dump incremental body(default all)
      -t, --tbs                             dump specific tablespace body(usage if set body)
      -n, --number   INTEGER ( >= 0 )       (log or page) sequence number
      -f, --fetch    INTEGER ( >= 1 )       dump as much as count
```

Description

Name	Available file type	Description
--silent	All	It does not output copyright, version, the execution time.
--section	Control file	It refers to the section of control file to be dumped. • sys: System section • log: Log section • db: Database section

Name	Available file type	Description
		• backup: Incremental backup section • all: All sections (The default value)
--header	Datafile	It dumps only the header of the datafile. If not set, it does not display the header.
	Log file	It dumps only the header of the log file.
--number	Datafile	It is a page number to be dumped. (The default value is 0.)
	Log file	It is a log number to be dumped. (The default value is 0, and it dumps the log bigger than the specified number.)
	Incremental backup file	It is a page number to be dumped. (The default value is 0, and it dumps the page bigger than the specified number.)
--fetch	Datafile	It is a number of pages to be dumped. (The default value is 1.)
	Log file	It is a number of logs to be dumped. (The default value is the whole logs.)
	Incremental backup file	It is a number of pages to be dumped. (The default value is the whole pages.)
--offset	Log file	It is valid only when --number option argument is set, and it refers to the position apart as far as offset from the value set in number. (The default value is 0.)
--body	Incremental backup file	It displays the header of page (header) or the header and page (all). If not set, it does not display anything. (The default value is none).
--tbs	Incremental backup file	It is valid only when --body option is set. It specifies the certain tablespace number in the incremental backup file. If not set, it refers to all tablespace.
--all	Logl file	It dumps all including invalid logs.

An option is not used when dumping the property file.

42.2 Examples of Using gdump

Control File

- Dumping system section

```
$ gdump control control_0.ctl --section sys --silent
 [SYSTEM SECTION]
------------------------------------------------------------
  SERVER STATE                      : SERVICE
  DATA STORE MODE                   : TDS
  LAST CHECKPOINT LSN               : 136976
  INCREMENTAL BACKUP CHUNK COUNT    : 1
  INCREMENTAL BACKUP SECTION OFFSET : 512
  LOG  SECTION OFFSET               : 8704
  DB   SECTION OFFSET               : 13312
```

- Dumping log section

```
$ gdump control control_0.ctl --section log --silent
[LOG SECTION]
------------------------------------------------------------
   DATABASE CREATION TIME : 2014-08-25 18:37:11.299610
  [CHECKPOINT]
    LID                              :  0,54834,13
    LSN                              :  136976
    RECOVERY LSN                     :  136976
    ARCHIVELOG MODE                  :  ARCHIVELOG
    LAST INACTIVATED LOGFILE SEQUENCE :  -1
  [LOG STREAM]
    STATE          :  ACTIVE
    GROUP COUNT    :  4
    BLOCK SIZE     :  512
    FILE SEQUENCE  :  0
    [LOG GROUP #0]
      STATE         : CURRENT
      SIZE          : 104857600
      MEMBER COUNT  : 1
      FILE SEQUENCE : 0
```

```
      PREV LAST LSN : -1
      MEMBER #0     : "/home/lkh/work/product/Gliese/home/wal/redo_0_0.log"
```

- Dumping database section

```
$ gdump control control_0.ctl --section db --silent
 [DB SECTION]
------------------------------------------------------------
  [DATABASE]
    TRANSACTION_TABLE_SIZE : 1024
    UNDO_RELATION_COUNT    : 128
    TABLESPACE COUNT       : 4
    NEW TABLESPACE ID      : 4
  [TABLESPACE #0]
   NAME          :  DICTIONARY_TBS
   ATTRIBUTES    :  MEMORY | PERSISTENT | DICT
   STATE         :  CREATED
   LOGGING STATE :  LOGGING
   ONLINE STATE  :  ONLINE
   EXTENT_SIZE   :  8
   RELATION_ID   :  0
   [DATAFILE #0]
     SIZE  :  134209536
     STATE :  CREATED
     NAME  :  "/home/lkh/work/product/Gliese/home/db/system_dict.dbf"
```

- Dumping incremental backup section

```
$ gdump control control_0.ctl --section backup --silent
[INCREMENTAL BACKUP SECTION]
------------------------------------------------------------
[ BACKUP #0 ]
FILE PATH : /home/lkh/work/product/Gliese/home/backup/databaseD20140825T183902L0S0.inc
BACKUP LSN : 136976
BACKUP LEVEL : 0
BACKUP OBJECT : database
[ BACKUP #1 ]
FILE PATH : /home/lkh/work/product/Gliese/home/backup/controlD20140825T183904L0S0.inc
BACKUP LSN : 136976
BACKUP LEVEL : 0
BACKUP OBJECT : control
```

Datafile

- Dumping the first page of datafile *system_dict.dbf*

```
$ gdump data system_dict.dbf --silent
  TABLESPACE ID : 0
  DATAFILE ID : 0
  PAGE SEQUENCE ID : 0
------------------------------------------------------------------------------
  [PHYSICAL HEADER] TYPE(EXT_BLOCK_MAP), FREENESS(FREE), LSN(107938),
TIMESTAMP(1408959438472006), PARENT RID(0,-1,0), SEGMENT ID(0), MAX VIEW SCN(0), AGABLE
SCN(0), SELF ID(0,0)
------------------------------------------------------------------------------
 0000    0200000004000000 A2A5010000000000 4643D5EE70010500 0000FFFF00000000
 0020    0000000000000000 0000000000000000 0000000000000000 0300000000000000
 0040    0000000000000000 0000000000000000 0000000000000000 0200000001000000
 0060    0200000001000000 0320000001000000 0000000000000000 0000000000000000
 0080    0000000000000000 0000000000000000 0000000000000000 0000000000000000
 00A0    0000000000000000 0000000000000000 0000000000000000 0000000000000000
 00C0    0000000000000000 0000000000000000 0000000000000000 0000000000000000
....
```

- Dumping three pages from the 100th page of datafile *system_dict.dbf*

```
$ gdump data system_dict.dbf --number 100 --fetch 3 --silent
  TABLESPACE ID : 0
  DATAFILE ID : 0
  PAGE SEQUENCE ID : 100
------------------------------------------------------------------------------
  [PHYSICAL HEADER] TYPE(BITMAP_HEADER), FREENESS(INSERTABLE), LSN(348),
TIMESTAMP(1408959438472006), PARENT RID(0,57,4), SEGMENT ID(4294967296), MAX VIEW SCN(0),
AGABLE SCN(0), SELF ID(0,1
00)
------------------------------------------------------------------------------
 0000    0400000003000000 5C01000000000000 4643D5EE70010500 0000390004000000
 0020    0000000001000000 0000000000000000 0000000000000000 80C0317800000000
 0040    0000000000000000 0000000000000000 0000000064000000 0100000001000002
 0060    FFFFFFFFFFFFFFFF 0100000000000000 0000000000000000 0000000000000000
 0080    0000000000000000 0200000001000000 7303000072030000 6E00010000000000
 00A0    3700000000000000 0000000000000000 0000000000000000 0000000000000000
```

```
 00C0    0000000000000000 0000000000000000 0000000000000000 0000000000000000
....
 1FC0    0000000000000000 0000000000000000 0000000000000000 0000000000000000
 1FE0    0000000000000000 0000000000000000 0000000000000000 5C01000000000000
  TABLESPACE ID : 0
  DATAFILE ID : 0
  PAGE SEQUENCE ID : 101
-------------------------------------------------------------------------------
  [PHYSICAL HEADER] TYPE(BITMAP_HEADER), FREENESS(INSERTABLE), LSN(353),
TIMESTAMP(1408959438472006), PARENT RID(0,58,4), SEGMENT ID(4294967296), MAX VIEW SCN(0),
AGABLE SCN(0), SELF ID(0,1
01)
-------------------------------------------------------------------------------
 0000    0400000003000000 6101000000000000 4643D5EE70010500 00003A0004000000
 0020    0000000001000000 0000000000000000 0000000000000000 00BF317800000000
 0040    0000000000000000 0000000000000000 0000000065000000 0100000001000002
 0060    FFFFFFFFFFFFFFFF 0100000000000000 0000000000000000 0000000000000000
 0080    0000000000000000 0200000001000000 8303000082030000 7000010000000000
 00A0    3800000000000000 0000000000000000 0000000000000000 0000000000000000
 00C0    0000000000000000 0000000000000000 0000000000000000 0000000000000000
....
 1FC0    0000000000000000 0000000000000000 0000000000000000 0000000000000000
 1FE0    0000000000000000 0000000000000000 0000000000000000 6101000000000000
  TABLESPACE ID : 0
  DATAFILE ID : 0
  PAGE SEQUENCE ID : 102
-------------------------------------------------------------------------------
  [PHYSICAL HEADER] TYPE(BITMAP_HEADER), FREENESS(INSERTABLE), LSN(358),
TIMESTAMP(1408959438472006), PARENT RID(0,59,4), SEGMENT ID(4294967296), MAX VIEW SCN(0),
AGABLE SCN(0), SELF ID(0,1
02)
-------------------------------------------------------------------------------
 0000    0400000003000000 6601000000000000 4643D5EE70010500 00003B0004000000
 0020    0000000001000000 0000000000000000 0000000000000000 C0BD317800000000
 0040    0000000000000000 0000000000000000 0000000066000000 0100000001000002
 0060    FFFFFFFFFFFFFFFF 0100000000000000 0000000000000000 0000000000000000
 0080    0000000000000000 0200000001000000 9303000092030000 7200010000000000
 00A0    3900000000000000 0000000000000000 0000000000000000 0000000000000000
 00C0    0000000000000000 0000000000000000 0000000000000000 0000000000000000
....
```

- Dumping the header of datafile *system_dict.dbf*

```
$ gdump data system_dict.dbf --header --silent
  FILE                  : system_dict.dbf
  Tablespace Physical Id : 0
  Datafile Id           : 0
  Last Checkpoint Lsn    : 136976
  Creation TIME         : 2014-08-25 18:37:18.472006
```

Log File

- It dumps all of online redo log file *redo_0_0.log*.

```
$ gdump log redo_0_0.log --silent
============================================================
 [LOG FILE HEADER]
------------------------------------------------------------
 LOG_GROUP_ID    : 0
 BLOCK_SIZE      : 512
 FILE_SIZE       : 104857600
 FILE_SEQUENCE   : 0
 PREV_LAST_LSN   : -1
 CREATION TIME   : 2014-08-25 18:37:13.315283
============================================================
[LOG #0] : BLOCK(0), LSN(0), SIZE(1132), PIECE_COUNT(1), TRANS_ID(FFFFFFFFFFFF0001),
RID(0,-1,0)
[PIECE #0] : TYPE(MEMORY_FILE_CREATE), SIZE(1116), CLASS(DATAFILE), REDO_TYPE(CONTROL_FILE),
PROPAGATE_LOG(YES), RID(0,0,0)
 FFFFFFFF2F686F6D 652F6C6B682F776F 726B2F70726F6475 63742F476C696573    ..../hom e/lkh/wo
rk/produ ct/Glies
 652F686F6D652F64 622F73797374656D 5F646963742E6462 6600000000000000    e/home/d b/system
_dict.db f.......
 0000000000000000 0000000000000000 0000000000000000 0000000000000000    ........ ........
........ ........
[LOG #1] : BLOCK(3), LSN(1), SIZE(232), PIECE_COUNT(1), TRANS_ID(FFFFFFFFFFFF0001), RID(0,0,0)
[PIECE #0] : TYPE(MEMORY_TBS_CREATE), SIZE(216), CLASS(DATAFILE), REDO_TYPE(CONTROL_FILE),
PROPAGATE_LOG(YES), RID(0,0,0)
... Ellipsis ...
```

- It dumps three logs from a log which is away as far as five offsets from log sequence number 100 in *r*

edo_0_0.log.

```
$ gdump log redo_0_0.log --number 100 --offset 5 --fetch 3 --silent
============================================================
 [LOG FILE HEADER]
------------------------------------------------------------
 LOG_GROUP_ID    : 0
 BLOCK_SIZE      : 512
 FILE_SIZE       : 104857600
 FILE_SEQUENCE   : 0
 PREV_LAST_LSN   : -1
 CREATION TIME   : 2014-08-25 18:37:13.315283
============================================================
[LOG #0] : BLOCK(105), LSN(105), SIZE(24), PIECE_COUNT(1), TRANS_ID(FFFFFFFFFFFF0001),
RID(0,0,55)
[PIECE #0] : TYPE(INIT_RELATION_HEADER), SIZE(8), CLASS(ACCESS), REDO_TYPE(PAGE),
PROPAGATE_LOG(YES), RID(0,160,55)
 0A00000000000000                                                 ........
[LOG #1] : BLOCK(105), LSN(106), SIZE(240), PIECE_COUNT(4), TRANS_ID(FFFFFFFFFFFF0001),
RID(0,0,56)
[PIECE #0] : TYPE(INIT_PAGE), SIZE(88), CLASS(PAGE_ACCESS), REDO_TYPE(PAGE),
PROPAGATE_LOG(YES), RID(0,0,56)
 0400000004000000 0000000000000000 4643D5EE70010500 00000D0004000000    ........ ........
FC..p... ........
 0000000001000000 0000000000000000 0000000000000000 0000000000000000    ........ ........
........ ........
 0000000000000000 0000000000000000 0000000038000000                     ........ ........
....8...
[PIECE #1] : TYPE(MEMORY_BITMAP_UPDATE_LEAF_STATUS), SIZE(24), CLASS(SEGMENT),
REDO_TYPE(PAGE), PROPAGATE_LOG(YES), RID(0,13,4)
 0400000003000000 0000000000000000 0000000000000000                     ........ ........
........
[PIECE #2] : TYPE(BYTES), SIZE(8), CLASS(RECOVERY), REDO_TYPE(PAGE), PROPAGATE_LOG(YES),
RID(0,4,56)
 0400000003000000                                                 ........
[LOG #2] : BLOCK(105), LSN(107), SIZE(832), PIECE_COUNT(8), TRANS_ID(FFFFFFFFFFFF0001),
RID(0,-1,0)
[PIECE #0] : TYPE(INIT_PAGE), SIZE(88), CLASS(PAGE_ACCESS), REDO_TYPE(PAGE),
PROPAGATE_LOG(YES), RID(1,0,178)
 0100000004000000 0000000000000000 BED7F8EE70010500 0000FFFF00000000    ........ ........
....p... ........
```

```
 0000000000000000 0000000000000000 0000000000000000 0000000000000000    ........ ........
........ ........
 0000000000000000 0000000000000000 01000000B2000000                     ........ ........
........
```

Incremental Backup File

- It dumps incremental backup information.
  - Only header and tail of file in *databaseD20140825T183902L0S0.inc* are dumped as follows.

```
$ gdump backup databaseD20140825T183902L0S0.inc --silent
 INCREMENTAL FILE HEADER
-------------------------------------------------------------------------------
  OBJECT TYPE(DATABASE), TBS COUNT(3), BODY SIZE(62742528)
  LSN: PREV(0), MAX (136976), CHKPT(136976)
  CHKPT LID: File Seq No(0), Block Info1(877344), Block Info2(13)
-------------------------------------------------------------------------------
 INCREMENTAL FILE TAIL
-------------------------------------------------------------------------------
  TABLESPACE ID(000), BACKUP PAGE COUNT(05372), TABLESPACE OFFSET(8192)
  TABLESPACE ID(001), BACKUP PAGE COUNT(02090), TABLESPACE OFFSET(44015616)
  TABLESPACE ID(002), BACKUP PAGE COUNT(00197), TABLESPACE OFFSET(61136896)
```

- It dumps header and body of page for the entire page of incremental backup.
  - The entire page of all tablespaces in *databaseD20140825T183902L0S0.inc* is dumped as follows.

```
$ gdump backup databaseD20140724T123414L0S0.inc --body all
 INCREMENTAL FILE HEADER
-------------------------------------------------------------------------------
  OBJECT TYPE(DATABASE), TBS COUNT(3), BODY SIZE(62742528)
  LSN: PREV(0), MAX (136976), CHKPT(136976)
  CHKPT LID: File Seq No(0), Block Info1(877344), Block Info2(13)
-------------------------------------------------------------------------------
 INCREMENTAL FILE TAIL
-------------------------------------------------------------------------------
  TABLESPACE ID(000), BACKUP PAGE COUNT(05372), TABLESPACE OFFSET(8192)
  TABLESPACE ID(001), BACKUP PAGE COUNT(02090), TABLESPACE OFFSET(44015616)
  TABLESPACE ID(002), BACKUP PAGE COUNT(00197), TABLESPACE OFFSET(61136896)
  TABLESPACE ID : 0
  DATAFILE ID : 0
```

```
  PAGE SEQUENCE ID : 0
--------------------------------------------------------------------------------
  [PHYSICAL HEADER] TYPE(EXT_BLOCK_MAP), FREENESS(FREE), LSN(107938),
TIMESTAMP(1408959438472006), PARENT RID(0,-1,0), SEGMENT ID(0), MAX VIEW SCN(0), AGABLE
SCN(0), SELF ID(0,0)
--------------------------------------------------------------------------------
 0000    0200000004000000 A2A5010000000000 4643D5EE70010500 0000FFFF00000000
 0020    0000000000000000 0000000000000000 0000000000000000 0000000000000000
 0040    0000000000000000 0000000000000000 0000000000000000 0200000001000000
 0060    0200000001000000 0320000001000000 0000000000000000 0000000000000000
 0080    0000000000000000 0000000000000000 0000000000000000 0000000000000000
... Ellipsis ...
  TABLESPACE ID : 0
  DATAFILE ID : 0
  PAGE SEQUENCE ID : 1
--------------------------------------------------------------------------------
  [PHYSICAL HEADER] TYPE(BITMAP_HEADER), FREENESS(FREE), LSN(100016),
TIMESTAMP(1408959438472006), PARENT RID(0,-1,0), SEGMENT ID(4294967296), MAX VIEW SCN(0),
AGABLE SCN(0), SELF ID(0,1)
--------------------------------------------------------------------------------
 0000    0400000004000000 B086010000000000 4643D5EE70010500 0000FFFF00000000
 0020    0000000001000000 0000000000000000 0000000000000000 802A327800000000
 0040
... Ellipsis ...
  TABLESPACE ID : 2
  DATAFILE ID : 0
  PAGE SEQUENCE ID : 24580
--------------------------------------------------------------------------------
  [PHYSICAL HEADER] TYPE(EXT_MAP), FREENESS(FREE), LSN(24), TIMESTAMP(1408959441642963),
PARENT RID(0,-1,0), SEGMENT ID(0), MAX VIEW SCN(0), AGABLE SCN(0), SELF ID(2,24580)
--------------------------------------------------------------------------------
 0000    0300000004000000 1800000000000000 D3A505EF70010500 0000FFFF00000000
 0020    0000000000000000 0000000000000000 0000000000000000 0000000000000000
 0040    0000000000000000 0000000000000000 0200000004600000 1E001E0000001E00
... Ellipsis ...
```

- It dumps header and body of page whose page number is bigger than the specified number in incre mental backup.
  - The pages whose page number is bigger than 1,000 of all tablespaces in *databaseD20140825T183902L0S0.inc* are dumped as follows.

```
$ gdump backup databaseD20140724T123414L0S0.inc --number 10000 --body all
 INCREMENTAL FILE HEADER
--------------------------------------------------------------------------------
  OBJECT TYPE(DATABASE), TBS COUNT(3), BODY SIZE(62742528)
  LSN: PREV(0), MAX (136976), CHKPT(136976)
  CHKPT LID: File Seq No(0), Block Info1(877344), Block Info2(13)
--------------------------------------------------------------------------------
 INCREMENTAL FILE TAIL
--------------------------------------------------------------------------------
  TABLESPACE ID(000), BACKUP PAGE COUNT(05372), TABLESPACE OFFSET(8192)
  TABLESPACE ID(001), BACKUP PAGE COUNT(02090), TABLESPACE OFFSET(44015616)
  TABLESPACE ID(002), BACKUP PAGE COUNT(00197), TABLESPACE OFFSET(61136896)
  TABLESPACE ID : 2
  DATAFILE ID : 0
  PAGE SEQUENCE ID : 16387
--------------------------------------------------------------------------------
  [PHYSICAL HEADER] TYPE(EXT_MAP), FREENESS(FREE), LSN(22), TIMESTAMP(1408959441642963),
PARENT RID(0,-1,0), SEGMENT ID(0), MAX VIEW SCN(0), AGABLE SCN(0), SELF ID(2,16387)
--------------------------------------------------------------------------------
 0000    0300000004000000 1600000000000000 D3A505EF70010500 0000FFFF00000000
 0020    0000000000000000 0000000000000000 0000000000000000 0000000000000000
 0040    0000000000000000 0000000000000000 0200000003400000 0001000100000001
 0060    0000000000000000 0000000000000000 0000000000000000 0000000000000000
 0080    0000000000000000 0000000000000000 0000000000000000 0000000000000000
... Ellipsis ...
  TABLESPACE ID : 2
  DATAFILE ID : 0
  PAGE SEQUENCE ID : 24580
--------------------------------------------------------------------------------
  [PHYSICAL HEADER] TYPE(EXT_MAP), FREENESS(FREE), LSN(24), TIMESTAMP(1408959441642963),
PARENT RID(0,-1,0), SEGMENT ID(0), MAX VIEW SCN(0), AGABLE SCN(0), SELF ID(2,24580)
--------------------------------------------------------------------------------
 0000    0300000004000000 1800000000000000 D3A505EF70010500 0000FFFF00000000
 0020    0000000000000000 0000000000000000 0000000000000000 0000000000000000
 0040    0000000000000000 0000000000000000 0200000004600000 1E001E0000001E00
 0060    0000000000000000 0000000000000000 0000000000000000 0000000000000000
 0080    0000000000000000 0000000000000000 0000000000000000 0000000000000000
... The rest is omitted ...
```

- It dumps header and body of page for the specified number of the specified tablespace in incrementa l backup.

- The page of number 1,000 in the tablespace of number 0 in *database D20140825T183902L0S0.inc* is dumped as follows.

```
gdump backup databaseD20140825T183902L0S0.inc --tbs 0 --number 1000 --body all --fetch 1
--silent ¦ more
 INCREMENTAL FILE HEADER
--------------------------------------------------------------------------------
  OBJECT TYPE(DATABASE), TBS COUNT(3), BODY SIZE(62742528)
  LSN: PREV(0), MAX (136976), CHKPT(136976)
  CHKPT LID: File Seq No(0), Block Info1(877344), Block Info2(13)
--------------------------------------------------------------------------------
 INCREMENTAL FILE TAIL
--------------------------------------------------------------------------------
  TABLESPACE ID(000), BACKUP PAGE COUNT(05372), TABLESPACE OFFSET(8192)
  TABLESPACE ID(001), BACKUP PAGE COUNT(02090), TABLESPACE OFFSET(44015616)
  TABLESPACE ID(002), BACKUP PAGE COUNT(00197), TABLESPACE OFFSET(61136896)
  TABLESPACE ID : 0
  DATAFILE ID : 0
  PAGE SEQUENCE ID : 1000
--------------------------------------------------------------------------------
  [PHYSICAL HEADER] TYPE(UNFORMAT), FREENESS(FREE), LSN(6307), TIMESTAMP(1408959438472006),
PARENT RID(0,-1,0), SEGMENT ID(0), MAX VIEW SCN(0), AGABLE SCN(0), SELF ID(0,1000)
--------------------------------------------------------------------------------
 0000    0100000004000000 A318000000000000 4643D5EE70010500 0000FFFF00000000
 0020    0000000000000000 0000000000000000 0000000000000000 0000000000000000
 0040    0000000000000000 0000000000000000 00000000E8030000 0000000000000000
 0060    0000000000000000 0000000000000000 0000000000000000 0000000000000000
... The rest is omitted ...
```

Property File

- PAGE_CHECKSUM_TYPE which is the one of the property is set by using gsql as follows.

```
gSQL> ALTER SYSTEM SET PAGE_CHECKSUM_TYPE = 1 SCOPE = BOTH;
System altered.
```

- The binary property file of GOLDILOCKS is dumped as follows.

```
$ gdump property goldilocks.properties.binary
============================================================
FILE: goldilocks.properties.binary
TYPE: PROPERTY-BINARY
TIME: 2014-08-29 12:45:13.853448
============================================================
PAGE_CHECKSUM_TYPE = 1
============================================================
TIME: 2014-08-29 12:45:13.853627
============================================================
```

Commit Log

The commit log file is dumped as follows.

```
$ gdump commit_log commit.log
============================================================
FILE: commit.log
TYPE: COMMIT LOG FILE
TIME: 2017-03-23 11:47:42.555368
============================================================
============================================================
 [COMMIT LOG FILE HEADER]
------------------------------------------------------------
 FILE_SEQUENCE   : 0
 FILE_SIZE       : 104857600
 MAX_BLOCK_COUNT : 204800
 SIGNATURE       : 5DAABFECBDCD11E68785A3701A679D47
============================================================
[BLOCK(0), LOG_COUNT(1)]
 TRANS_ID(1.0.59899956), COMMIT_SCN(16603.645.17371), INDOUBT_BEHAVIOR(FORGET),
SYNC_GLOBAL_TABLE_SCN(TRUE), PREV_LOG_FILE_SEQ(-1), PREV_LOG_BLOCK_SEQ(-1),
PREV_LOG_SLOT_SEQ(-1)
```

Log Buffer File

The log buffer file created after executing gsyncher is dumped as follows.
For more information, refer to **gsyncher**.

```
$ gdump log_buffer logbuffer.log
=============================================================
FILE: logbuffer.log
TYPE: LOG BUFFER FILE
TIME: 2017-03-23 11:10:12.787400
=============================================================
=============================================================
 [LOG BUFFER FILE HEADER]
-------------------------------------------------------------
 REAR_FILE_BLOCK_SEQ_NO   : 203410
 FRONT_FILE_BLOCK_SEQ_NO  : 203410
 REAR_SBSN                : 797213
 REAR_LSN                 : 375933
 REAR_LID                 : (3, 3254496, 161)
 FRONT_SBSN               : 797212
 FRONT_LSN                : 375933
 FILE_SEQ_NO              : 3
 BUFFER_SIZE              : 10485760
 BUFFER_BLOCK_COUNT       : 20480
 BLOCK_OFFSET             : 13
 GROUP_COMMIT_LSN         : 336688
 LOG_SWITCHING_SBSN       : -1
=============================================================
```

Pending Log Buffer File

The pending log buffer file created after executing gsyncher is dumped as follows.
For more information, refer to **gsyncher**.

```
$ gdump pend_buffer pendbuffer.log
=============================================================
FILE: pendbuffer.log
TYPE: PEND BUFFER FILE
TIME: 2017-03-23 11:10:29.603646
```

```
============================================================
============================================================
 [PEND LOG BUFFER FILE HEADER]
 PEND_LOG_BUFFER COUNT : 4
------------------------------------------------------------
 PEND_LOG_BUFFER NO. : 0
 PEND_BUFFER_SIZE    : 1048576
 BUFFER_BLOCK_COUNT  : 16384
 FRONT_PBSN          : 102
 REAR_PBSN           : 102
 FRONT_LSN           : -1
 REAR_LSN            : 327055
------------------------------------------------------------
 PEND_LOG_BUFFER NO. : 1
 PEND_BUFFER_SIZE    : 1048576
 BUFFER_BLOCK_COUNT  : 16384
 FRONT_PBSN          : 0
 REAR_PBSN           : 0
 FRONT_LSN           : -1
 REAR_LSN            : -1
------------------------------------------------------------
 PEND_LOG_BUFFER NO. : 2
 PEND_BUFFER_SIZE    : 1048576
 BUFFER_BLOCK_COUNT  : 16384
 FRONT_PBSN          : 0
 REAR_PBSN           : 0
 FRONT_LSN           : -1
 REAR_LSN            : -1
------------------------------------------------------------
 PEND_LOG_BUFFER NO. : 3
 PEND_BUFFER_SIZE    : 1048576
 BUFFER_BLOCK_COUNT  : 16384
 FRONT_PBSN          : 0
 REAR_PBSN           : 0
 FRONT_LSN           : -1
 REAR_LSN            : -1
------------------------------------------------------------
============================================================
```

43.

tablediff

43.1 Overview of tablediff

Background

The GOLDILOCKS system operator should prepare for the unexpected failure of GOLDILOCKS synchronization of two tables using the tools such as cyclone or LogMirror. An unexpected failure of synchronization refers that a particular row exists only in one table or the column values of rows to be synchronized are different from each other. Cyclone and LogMirror do not notify a user whether synchronization is failed. It is required to verify the synchronization of the table and to perform synchronization, if necessary, at non-operation time.

Features

tablediff is a tool comparing two tables of GOLDILOCKS which was synchronized by using CDC in a row unit. It reports when a particular row exists only in a table or values are different each other, then performs synchronization. The configuration file for controlling various operations is input, and the log files to report the unsynchronized rows and the synchronization results are output.

The constraints are that the schemas of two tables should be same and each of them should have the primary key. It is not recommended to perform the tool for table of the currently executing transaction because the table can be updated in real time. However, it does not matter for the currently executing querying (SELECT statement). This tool is available only for the GOLDILOCKS tables.

This tool has two executing commands, which are TableDiff and TableSync.

TableDiff

```
java sunje.goldilocks.tool.diff.TableDiff [configure file]
```

TableDiff program verifies whether two tables are unsynchronized, and it may output the information(binary file) to perform synchronization for unsynchronized rows immediately or at a later time according to an option. The immediate synchronization can be operated for multiple threads.

TableSync

```
java sunje.goldilocks.tool.diff.TableSync [configure file]
```

Tablesync program performs synchronization by using the sync information which was previously left by T ableDiff. (Row comparison is not performed.) A simultaneous execution can be performed by driving mul ti threads.

Unsynchronized rows of two tables to which the configure file is not applied are stored in a bin file left by TableDiff. TableSync uses this bin file to forcibly synchronize two tables.

Characteristics

This tool is a java application and it is provided in a jar file form, so Java(1.6) is required to execute the to ol. Also, goldilocks6.jar is required because GOLDILOCKS JDBC driver is used. It can be remotely performe d because it is connected to GOLDILOCKS by using TCP/ IP, and it is performed at much faster rate than u sing JDBC, ODBC with the proprietary protocol.

The row comparison and synchronization can be simultaneously performed with multi threads. For multit hreading of the row comparison, equal dividing of the range of the key in the table should be manually p erformed. When the user splits the key range into ten ranges (The user can specify it in the configuration file), then ten threads compare the tables. On the other hand, the synchronization operation is performe d by threads of as many as it is specified.

File Configuration

tablediff program consists of a single file called as $GOLDILOCKS_HOME/bin/tablediff.jar. $GOLDILOCKS _HOME/lib/goldilocks6.jar file is also required for execution. Also, the configuration file is required as an i nput argument, and refer to the sample file, $GOLDILOCKS_HOME/conf/tablediff.conf.

43.2 Usage

Command Usage

Java (JRE 1.6 or JDK1.6) is required because tablediff is a java program. For this, tablediff.jar and goldiloc ks6.jar files should be included in CLASSPATH, or they should be specified with -classpath option of java. tablediff is executed as follows.

```
export
CLASSPATH=$CLASSPATH:$GOLDILOCKS_HOME/bin/tablediff.jar:$GOLDILOCKS_HOME/lib/goldilocks6.jar
java sunje.goldilocks.tool.diff.TableDiff [configure file]
```

Or

```
java -classpath $GOLDILOCKS_HOME/bin/tablediff.jar:$GOLDILOCKS_HOME/lib/goldilocks6.jar
sunje.goldilocks.tool.diff.TableDiff [configure file]
```

The result of executing TableDiff for the simple sample table is as follows.

```
gSQL> create table tab1 ( c1 integer primary key, c2 char(10) );
gSQL> create table tab2 ( c1 integer primary key, c2 char(10) );
gSQL> insert into tab1 values ( 1, 'HELLO');
gSQL> insert into tab2 values ( 1, 'HELLO');
gSQL> insert into tab1 values ( 2, 'WORLD');
gSQL> insert into tab2 values ( 2, 'world');
gSQL> insert into tab1 values ( 3, 'good');
gSQL> insert into tab2 values ( 4, 'good');
gSQL> commit;
shell> java sunje.goldilocks.tool.diff.TableDiff tablediff.conf
Total 4 rows processed
  > row diff            : 1, update target(success/failure): 1/0
  > key diff source only: 1, insert into target(success/failure): 1/0
  > key diff target only: 1, delete from target(success/failure): 1/0
TableDiff completed
elapsed time = 0.229 sec
```

- The contents of tablediff.conf

```
SOURCE_URL      = jdbc:goldilocks://127.0.0.1:22581/test
SOURCE_USER     = TEST
SOURCE_PASSWORD = test
SOURCE_SCHEMA   = PUBLIC
SOURCE_TABLE    = TAB1
TARGET_URL      = jdbc:goldilocks://127.0.0.1:22581/test
TARGET_USER     = TEST
TARGET_PASSWORD = test
TARGET_SCHEMA   = PUBLIC
TARGET_TABLE    = TAB2
OPERATION       = SYNC
TARGET_INSERT = ON
TARGET_UPDATE = ON
TARGET_DELETE = ON
SOURCE_INSERT = OFF
```

Property Option

This chapter describes the available property options in the configuration file of tablediff.

Property Options for SOURCE, TARGET Tables

These property options should necessarily be specified and they define the source and target tables. The s ource and target refer to table to be compared each.

- SOURCE_URL: It is JDBC connection URL of GOLDILOCKS in which the source table exists.
- SOURCE_USER: It is the user account of GOLDILOCKS in which the source table exists.
- SOURCE_PASSWORD: It is the password of GOLDILOCKS in which the source table exists.
- SOURCE_SCHEMA: It is the schema of source table.
- SOURCE_TABLE: It is the name of source table.
- TARGET_URL: It is JDBC connection URL of GOLDILOCKS in which the target table exists.
- TARGET_USER: It is the user account of GOLDILOCKS in which the target table exists.
- TARGET_PASSWORD: It is the password of GOLDILOCKS in which the target table exists.
- TARGET_SCHEMA: It is the schema of target table.
- TARGET_TABLE: It is the name of target table.

The following is an example.

```
SOURCE_URL      = jdbc:goldilocks://192.168.0.100:22581/test
SOURCE_USER     = TEST
SOURCE_PASSWORD = test
SOURCE_SCHEMA   = PUBLIC
SOURCE_TABLE    = T1
TARGET_URL      = jdbc:goldilocks://192.168.0.101:22581/test
TARGET_USER     = TEST
TARGET_PASSWORD = test
TARGET_SCHEMA   = PUBLIC
TARGET_TABLE    = T2
```

Operation

It determines the operations of TableDiff. The DIFF property verifies only the synchronization integrity and reports it, but the SYNC property verifies the integrity and simultaneously performs synchronization.

The integrity result file (whose file name is specified as DIFF_BIN_ FILE property) is generated when operating with DIFF and it is used to execute TableSync.

Synchronization Policy

Four properties are used for controlling the synchronization policy and all of them have the value of either ON or OFF.

- TARGET_INSERT: If the key in the source does not exist in the target, it inserts the key into the target.
- TARGET_UPDATE: If the value of the column which is not a key is different, it updates the target row.
- TARGET_DELETE: If the key does not exist in the source but exists in the target, it deletes the target row.
- SOURCE_INSERT: If the key does not exist in the source but exists in the target, it inserts that row into the source.

Both TARGET_DELETE and SOURCE_INSERT are not allowed to be ON together.

EXCLUDED_COLUMNS

It specifies the columns to be excluded from the comparison. The comma (,) is used as a delimiter and the column name should be specified. The key columns can not be excluded.

WHERE_CLAUSE

It sets the conditions for comparing rows in the table. For example, the condition, WHERE_CLAUSE = SALARY> = 1000000, refers that the integrity is verified only for the rows having the values of their column salary equal to or bigger than 1,000,000.

DISPLAY_ROW_UNIT

TableDiff program displays the progress status of table comparison, and it outputs to the console the number of rows processed each time whenever it processes a certain number of rows. The property sets the number of rows to be output. the default value is 100,000, and it can be omitted.

SYNC_OUT_FILE

It specifies the name of the file in which the synchronization result is to be recorded. If not specified, the result is recorded in tablesync.log. If multiple synchronized threads exist, a number is added at the end of the file name.

DIFF_OUT_FILE

It specifies the name of the file in which the row mismatch result is to be recorded. If not specified, the result is recorded in tablesync.log. This file is in a text form which is human-readable.

DIFF_BIN_FILE

If the operation property is set to DIFF, TableDiff records the synchronization information in a file with a name specified by this property. This file is used as an input argument by TableSync.

PROPAGATE_REDO_LOG

It determines whether to propagate the row synchronization log to another replicated server. The default value is OFF.

LOGGING_ON_SUCCESS

It sets whether to record logs even when DML(INSERT, UPDATE, DELETE) used for the row synchronization is successful. The default value is OFF. If it is OFF, then the synchronization performance becomes poor. If DML is failed, the logging information is recorded regardless of this property.

LOGGING_ON_DIFF

It sets whether to record logs when the mismatch occurs during comparing the row integrity. The default value is OFF.

JOB_QUEUE_SIZE

The row synchronization is performed by the operating thread. The threads perform the synchronization by getting the operation from the JOB QUEUE one by one. The main thread of TableDiff or TableSync ins erts the operation to JOB QUEUE. JOB QUEUE becomes full if the thread slowly performs synchronization. JOB_QUEUE_SIZE property sets the size of JOB QUEUE. If this value is big, JOB QUEUE never becomes full but it wastes a lot of memory. The default value is 100, and 100 is big enough to use without filling of th e queue.
The property is recommended not to change unless it is needed.

JOB_THREAD

It sets the number of threads for synchronization. If not set, the default value is 1. If there are multiple ro ws to be synchronized, the value should be set considering the number of CPU. The bigger the value, the faster the synchronization is performed.

JOB_UNIT_SIZE

The synchronization is performed in batch as much as the size specified by the property. The default value is 100.

DISPLAY_CALL_STACK

It sets whether to display call stack when an error occurs. The default value is OFF.

Replication Settings for Table Comparison

The row comparison of TableDiff is performed by a single thread by default. But multiple threads can perf orm comparisons by specifying the condition clause. The property name is PARTITION_RANGE[n], and if t he range for N properties is set such as the WHERE clause, n threads perform the comparison each.

For example, the following properties are specified for each of 12 threads to perform the comparison wh en the table includes the monthly data.

```
PARTITION_RANGE1 = MONTH=1
PARTITION_RANGE2 = MONTH=2
```

```
PARTITION_RANGE3 = MONTH=3
PARTITION_RANGE4 = MONTH=4
PARTITION_RANGE5 = MONTH=5
PARTITION_RANGE6 = MONTH=6
PARTITION_RANGE7 = MONTH=7
PARTITION_RANGE8 = MONTH=8
PARTITION_RANGE9 = MONTH=9
PARTITION_RANGE10 = MONTH=10
PARTITION_RANGE11 = MONTH=11
PARTITION_RANGE12 = MONTH=12
```

All conditions should be disjointed, and the union of all conditions should be as same as the total set. Als o, the column used in the condition should the front part of the primary key. (It means that the specified condition should be able to use the primary index.)
This property is applied only to TableDiff, and TableSync ignores this property.

44.

gsyncher

44.1 Overview of gsyncher

Definition

gsyncher is a utility provided by GOLDILOCKS, and which synchronizes the log of shared memory and the logfile of disk.
When the server is abnormally terminated, gsyncher synchronizes the latest log of shared memory and th e logfile, then records it.

Features

- gsyncher can not be executed during the server operation.
- The STARTUP stage of the server in which gsyncher can be executed is OPEN phase.
- When executing gsyncher, all other applications of the shared memory should be terminated.
- During gsyncher execution, the logfile can be switched and in this case, the controlfile is generated.
- gsyncher backups the controlfile and logfile in which the logs are reflected, in advance.
- gsyncher can be executed after restoring the damaged controlfile.

Usage

```
gsyncher [options]
```

Options

```
-l  --log              Log trace msg
-s  --silent           Do not print message
-f  --home             home directory
-c  --copy-right       Do not print copy right
-b  --backup-path      Backup directory
-h  --help             Print help message
```

44.2 Examples

- Example 1: gsyncher is executed during the server is normally operating.

```
$ gsyncher -l
[SHARED MEM] Attached to shm - Name(_STATIC), Key(21128)
[SHARED MEM] Detached from shm.
ERR-HY000(53002): gmaster is active
```

- Example 2: The server is not on the STARTUP stage on which gsyncher can be executed.

```
$ gsyncher -l
[SHARED MEM] Attached to shm - Name(_STATIC), Key(21128)
[SHARED MEM] Detached from shm.
ERR-HY000(53000): invalid phase(MOUNT): executable phase is OPEN
```

- Example 3: When executing gsyncher, the progress is displayed. The execution result is total buffer (0) bytes, and all logs are flushed to the logfile, so there is not a log to be synchronized.

```
$ gsyncher --log
[SHARED MEM] Attached to shm - Name(_STATIC), Key(21128)
[CLEAR PROCESS] Process 'gbalancer' is cleared.
[CLEAR PROCESS] Process 'gdispatcher' is cleared.
[CLEAR PROCESS] Process 'gdispatcher' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[FLUSH] Log buffer flushed - Log group from id (0) to (0), lsn from (130784) to (130784),
total buffer (0) bytes
[SHARED MEM] Detached from shm.
[FINI] Log sync complete.
```

    - gsyncher operations are recorded in logfile by using --log option.

```
$ cat $GOLDILOCKS_DATA/trc/gsyncher.trc
================================================
 gsyncher start
 TIME    : 2015-08-05 16:12:15.776633
================================================
[2015-08-05 16:12:15.776763 THREAD(29199,139846497756928)]
[INIT] Log started.
[2015-08-05 16:12:15.777113 THREAD(29199,139846497756928)]
[SHARED MEM] Attached to shm - Name(_STATIC), Key(21128)
[2015-08-05 16:12:15.777208 THREAD(29199,139846497756928)]
[CLEAR PROCESS] Process 'gbalancer' is cleared.
[2015-08-05 16:12:15.777297 THREAD(29199,139846497756928)]
[CLEAR PROCESS] Process 'gdispatcher' is cleared.
[2015-08-05 16:12:15.777358 THREAD(29199,139846497756928)]
[CLEAR PROCESS] Process 'gdispatcher' is cleared.
[2015-08-05 16:12:15.777416 THREAD(29199,139846497756928)]
[CLEAR PROCESS] Process 'gserver' is cleared.
[2015-08-05 16:12:15.777917 THREAD(29199,139846497756928)]
[CLEAR PROCESS] Process 'gserver' is cleared.
[2015-08-05 16:12:15.777993 THREAD(29199,139846497756928)]
[CLEAR PROCESS] Process 'gserver' is cleared.
[2015-08-05 16:12:15.778051 THREAD(29199,139846497756928)]
[CLEAR PROCESS] Process 'gserver' is cleared.
[2015-08-05 16:12:15.778107 THREAD(29199,139846497756928)]
[CLEAR PROCESS] Process 'gserver' is cleared.
[2015-08-05 16:12:15.778577 THREAD(29199,139846497756928)]
[CLEAR PROCESS] Process 'gserver' is cleared.
[2015-08-05 16:12:15.778652 THREAD(29199,139846497756928)]
[CLEAR PROCESS] Process 'gserver' is cleared.
[2015-08-05 16:12:15.778710 THREAD(29199,139846497756928)]
[CLEAR PROCESS] Process 'gserver' is cleared.
[2015-08-05 16:12:15.778766 THREAD(29199,139846497756928)]
[CLEAR PROCESS] Process 'gserver' is cleared.
[2015-08-05 16:12:15.778822 THREAD(29199,139846497756928)]
[CLEAR PROCESS] Process 'gserver' is cleared.
[2015-08-05 16:12:15.818007 THREAD(29199,139846497756928)]
[SHARED MEM] Detached from shm.
[2015-08-05 16:12:15.818100 THREAD(29199,139846497756928)]
[FINI] Log sync complete.
```

- Example 4: The logs from lsn 130786 to lsn 131012 are flushed to the log group 1.

```
$ gsyncher -l
[SHARED MEM] Attached to shm - Name(_STATIC), Key(21128)
[CLEAR PROCESS] Process 'gsql' is cleared.
[CLEAR PROCESS] Process 'gbalancer' is cleared.
[CLEAR PROCESS] Process 'gdispatcher' is cleared.
[CLEAR PROCESS] Process 'gdispatcher' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[FLUSH] Log buffer flushed - Log group from id (1) to (1), lsn from (130786) to (131012),
total buffer (84480) bytes
[SHARED MEM] Detached from shm.
[FINI] Log sync complete.
```

- Example 5: If the logfile switch occurs during gsyncher operation, it is reflected in the controlfile.

```
$ gsyncher -l
[SHARED MEM] Attached to shm - Name(_STATIC), Key(21128)
[CLEAR PROCESS] Process 'gsql' is cleared.
[CLEAR PROCESS] Process 'gbalancer' is cleared.
[CLEAR PROCESS] Process 'gdispatcher' is cleared.
[CLEAR PROCESS] Process 'gdispatcher' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gserver' is cleared.
[CLEAR PROCESS] Process 'gsql' is cleared.
[CLEAR PROCESS] Process 'gsql' is cleared.
```

```
[FLUSH] Log buffer flushed - Log group from id (2) to (3), lsn from (156412) to (178129),
total buffer (16644096) bytes
[CONTROLFILE] Saving controlfile caused by logfile switching.
[CONTROLFILE] '/media/solid/ssd_home/work/product/Gliese/home/wal/control_0.ctl' control file
is saved.
[CONTROLFILE] '/media/solid/ssd_home/work/product/Gliese/home/wal/control_1.ctl' control file
is saved.
[SHARED MEM] Detached from shm.
[FINI] Log sync complete.
```

45.

gmon

45.1 Overview of gmon

Definition

gmon is a database (gmaster) process monitoring utility provided by GOLDILOCKS.

Features

gmon monitors whether database process (gmaster) is operated, and when the process is abnormally ter minated, not by user's SHUTDOWN, then it executes **gsyncher** and terminates the process. It intends to re duce the loss of logs by executing gsyncher.

gmon can be set to be automatically operated by setting the server property GMON_AUTOSTART to 1, or a user can directly executes gmon.

A single gmaster process can monitor only one gmon process.

Usage

```
gmon [options]
```

Options

```
-s  --start        Start gmon
-t  --stop         Stop gmon
-u  --status       Get gmon status
-o  --home         gmaster home path
-d  --uds_dir      unix domain socket directory
-l  --silent       Suppress display message
-r  --no-copyright Suppress display copy right and version
-h  --help         Print help messages
```

start

It starts gmon.

stop

It stops gmon.

status

It checks the status of gmon process.

home

It is the database home directory. If it is omitted, then it uses the value set in GOLDILOCKS_DATA environ ment variable.

uds_dir

It sets the directory in which unix domain socket is created. The maximum length of the directory is 50 by tes. The default value is /tmp.

silent

It does not output the result message.

no-copyright

It does not output copyright and version.

help

It outputs the help message.

45.2 Examples

- Example 1: Execute gmon by using the default path ($GOLDILOCKS_DATA).

```
$ gmon --start
gmon is started.
```

- Example 2: Execute gmon by specifying home path. When a relative path is used, home path is specif ied based on $GOLDILOCKS_DATA.

```
$ gmon --start --home g1n1_home
gmon is started.
```

- Example 3: Execute gmon by specifying home path as an absolute path. When an absolute path is us ed, the specified path is specified as a home path.

```
$ gmon --start --home /g1n1_home
gmon is started.
```

- Example 4: Check the status of gmon process.

```
$ gmon --status --home /g1n1_home
gmon is running(19119).
```

- Example 5: Terminate the gmon process.

```
$ gmon --stop --home /g1n1_home
gmon is stopped.
```

46.

gtrclogger

46.1 Overview of gtrclogger

Definition

gtrclogger is a utility remotely collecting logs of GOLDILOCKS.

GOLDILOCKS basically records events and trace logs which occur in a server, in system.trc file of the speci fied directory of SYSTEM_LOGGER_DIR of property. (The default value is $GOLDILOCKS_DATA/trc)

If multiple different instances are being operated, trace logs occur in each server should be separately mo nitored. For that, gtrclogger makes trace logs of each different GOLDILOCKS instances to be collected in a single spot. gtrclogger receives trace logs of various GOLDILOCKS being operated in a different device with udp, and stores them in a file.

gtrclogger receives trace logs from the remote GOLDILOCKS and records them in a file. For that, the rem ote GOLDILOCKS should have the following transfer properties.

- TRACE_LOGGER
- TRACE_LOGGER_REMOTE_HOST
- TRACE_LOGGER_REMOTE_PORT

Features

- gtrclogger can be operated regardless of server operation.
- Even when it receives trace logs from various servers, it sequentially stores them in a file.
- A trace log is basically stored as $(GOLDILOCKS_DATA)/trc/system.rmt.trc.
- If system.rmt.trc is bigger than 10M bytes , it is backuped as system.rmt.trc_*date*_Index.

Usage

```
gtrclogger [options]
```

Options

```
-s  --start        start gtrclogger
-q  --stop         stop gtrclogger
-p  --port         udp port for receive log (1024 ~ 49151): default 21470
-d  --dir          write file directory: default $(GOLDILOCKS_DATA)/system.rmt.trc
-h  --help         show gtrclogger help messages
```

46.2 Examples

- Example 1: Execute gtrclogger by using the default port (21470) and the default directory($GOLDILOCKS_DATA/trc).

```
$ gtrclogger --start
gtrclogger is started successfully.
```

- Example 2: Receive a trace log with two ports (21471, 21472) and set the directory as $GOLDILOCKS_DATA/rmt_trc, then execute gtrclogger.

```
$ gtrclogger -s -p 21471 -p 21472 -d $GOLDILOCKS_HOME/rmt_trc
ERR-HY000(11040): No such object
(/home/lym1/workspace/product/Gliese/home/rmt_trc/system.rmt.trc)
$ mkdir $GOLDILOCKS_HOME/rmt_trc
$ gtrclogger -s -p 21471 -p 21472 -d $GOLDILOCKS_HOME/rmt_trc
gtrclogger is started successfully.
```

- Example 3: Terminate gtrclogger.

```
$ gtrclogger -q
Stop Done.
```

47.

glocator

47.1 Overview of glocator

Definition

glocator is a utility which provides locations to a client and manages them in GOLDILOCKS cluster system. The location information of cluster member nodes is provided to the glocator program through **gloctl**. glocator communicates with the client and gloctl via UDP, and it communicates with gagent via TCP.

glocator requires the location information such as a listener host, listener port and db home pat h, and this information is provided to the glocator through gloctl.

Usage

```
glocator [options]
```

Options

help

Description

It outputs a help message.

Example

```
$ glocator --help
Usage:
 glocator [options]
Options:
-c  --create       Create glocator environment
-s  --start        Start glocator
-t  --stop         Stop glocator
-f  --conf         Set configure file
-u  --status       Get glocator status
-l  --silent       Suppress display message
```

```
-r  --no-copyright Suppress display copy right and version
-h  --help         Print help message
```

create

Description

It creates the data file of glocator.

Example

```
$ glocator --create
glocator is created.
```

The data file of glocator is created in <GOLDILOCKS_DATA>/db directory.

```
$ ls
README  glocator.dat  system_data.dbf  system_dict.dbf  system_undo.dbf
```

start

Description

It starts glocator. If glocator having the same port already has started, then an error occurs.

Example

```
$ glocator --start
glocator is started.
```

stop

Description

It stops glocator which is in operation. The same port should be set to stop glocator in operation.

Example

```
$ glocator --stop
glocator is stopped.
```

conf

Description

It sets the configure file when starting glocator.

Example

```
$ glocator --start --conf goldilocks.glocator.conf
glocator is started.
```

status

Description

It outputs the status message of glocator which is in operation. The same port should be set to check the status of glocator in operation.

Example

```
$ glocator --status
Process ID: 26058
Configuration file: goldilocks.glocator.conf
Unix domain path: /tmp/unix-glocator.42581
Udp listen host: 0.0.0.0, Port: 42581
glocator is running.
```

sync

Description

It synchronizes the data of glocator set in ALTERNATE_LOCATORS before driving glocator.

There are two methods for the data synchronization, whichare SOURCE and BOTH. SOURCE method brin gs data from ALTERNATE_LOCATORS, and BOTH method merges two glocators.

Example

The following is an example of success of a sync option.

```
$ glocator --start --sync
glocator is started.
```

The following is an example of failure of a sync option. The value is not set in ALTERNATE_LOCATORS pr operty.

```
$ glocator --start --sync
ERR-HY000(60016): Need more alternate locator host information.
```

The following is an example of failure of a sync option. glocator set in ALTERNATE_LOCATORS property d oes not response.

```
$ glocator --start --sync
ERR-HY000(60016): Need more alternate locator host information.
```

silent

Description

It does not output the message of glocator about the execution.

Example

```
$ glocator --start --silent
```

no-copyright

Description

It does not output the message of glocator's copyright and version about the execution.

Example

```
$ glocator --start --no-copyright
glocator is started.
```

47.2 Using glocator

Data File

glocator data file should be created before starting glocator.

```
$ glocator --create
glocator is created.
```

The directory in which glocator data file is stored can be altered by editing configuration **LOCATION_FILE_DIR**. The default value is created in <GOLDILOCKS_DATA>/db. The data file can be altered by editing **LOCATION_FILE_NAME**. The default value is glocator.dat.

The maximum size and initial size of glocator data file can be altered by editing configuration **LOCATION_FILE_MAX_SIZE**and **LOCATION_FILE_SIZE**.

CSTARTUP and CSHUTDOWN

glocator can be used to CSTARTUP and CSHUTDOWN the GOLDILOCKS server.
For that, glocator should be in operation, and CSTARTUP or CSHUTDOWN should be executed through gsqlnet.
However, LOCATOR_DSN and the property should be set in odbc.ini of the device which executes gsqlnet.
For more information, refer to **GOLDILOCKS UNIX ODBC driver libraries**.

The following contents is set in odbc.ini to use glocator.

```
[GOLDILOCKS]
HOST=127.0.0.1
PORT=20101
UID=sys
PWD=gliese
LOCATOR_DSN=GLOCATOR
[GLOCATOR]
HOST=127.0.0.1
PORT=42581
```

If FILE property exists in DSN [GLOCATOR], then the file specified in FILE property takes precedence instead of using glocator. Therefore, FILE property should be excluded.

glocator should know the location information of member nodes to execute CSTARTUP and CSHUTDOWN.

Replication

Overview

glocator replication keeps data consistent, so it enables the stable service through the remote glocator when an error occurs.

Configuration

Each glocator should set ALTERNATE_LOCATORS property in a configure file to use the replication feature. Locator_name should be set in ALTERNATE_LOCATORS property and the specified Locator_name should be set in a configure file together with HOST, PORT properties.

The following is how to set ALTERNATE_LOCATORS property.

```
[LOCATOR]
ALTERNATE_LOCATORS = (locator_name1,locator_name2)
[locator_name1]
HOST= ip_address
PORT = port_num
[locator_name2]
HOST= ip_address
PORT = port_num
```

The following is an example of a configure file which sets ALTERNATE_LOCATORS property.

```
[LOCATOR]
PORT=42581
SYNC_RESPONSE_TIMEOUT = 2
SYNC_RETRY_COUNT = 2
LOCATION_FILE_NAME='glocator_1.dat'
ALTERNATE_LOCATORS=(LOCATOR_2,LOCATOR_3)
[LOCATOR_2]
```

```
HOST=127.0.0.1
PORT=42582
[LOCATOR_3]
HOST=127.0.0.1
PORT=42583
```

- [LOCATOR] is DSN which is read as default by glocator.
- For more information glocator-related ODBC and how to set gagent, refer to **odbc.ini File**, **ALTERNATE_LOCATORS**.

Synchronizing Data

The data should be consistent when the replicated glocator is in service. If glocator in operation does not exist, then all glocator can perform the normal start. If glocator in operation exists, an alternate glocator can be started by synchronizing the data using a **sync** option.

sync option has BOTH and SOURCE. BOTH merges glocator's own data and alternate glocator's data. SOURCE brings the data from the counterpart glocator. The sync operation of glocator is 1 : 1 correspondence task between a glocator and an alternate glocator. Therefore, if A and B are already being operated, and C is operated after syncing by using BOTH for the replication of three (A, B, C) glocators, then the data of A and B glocators may be different.

glocator in service synchronizes only the updated content. If the data synchronization fails due to the packet loss or other reasons, then the data may be different. In this case, update the data within that glocator by using **gloctl** program, or restart glocator with sync option.

47.3 Features of glocator

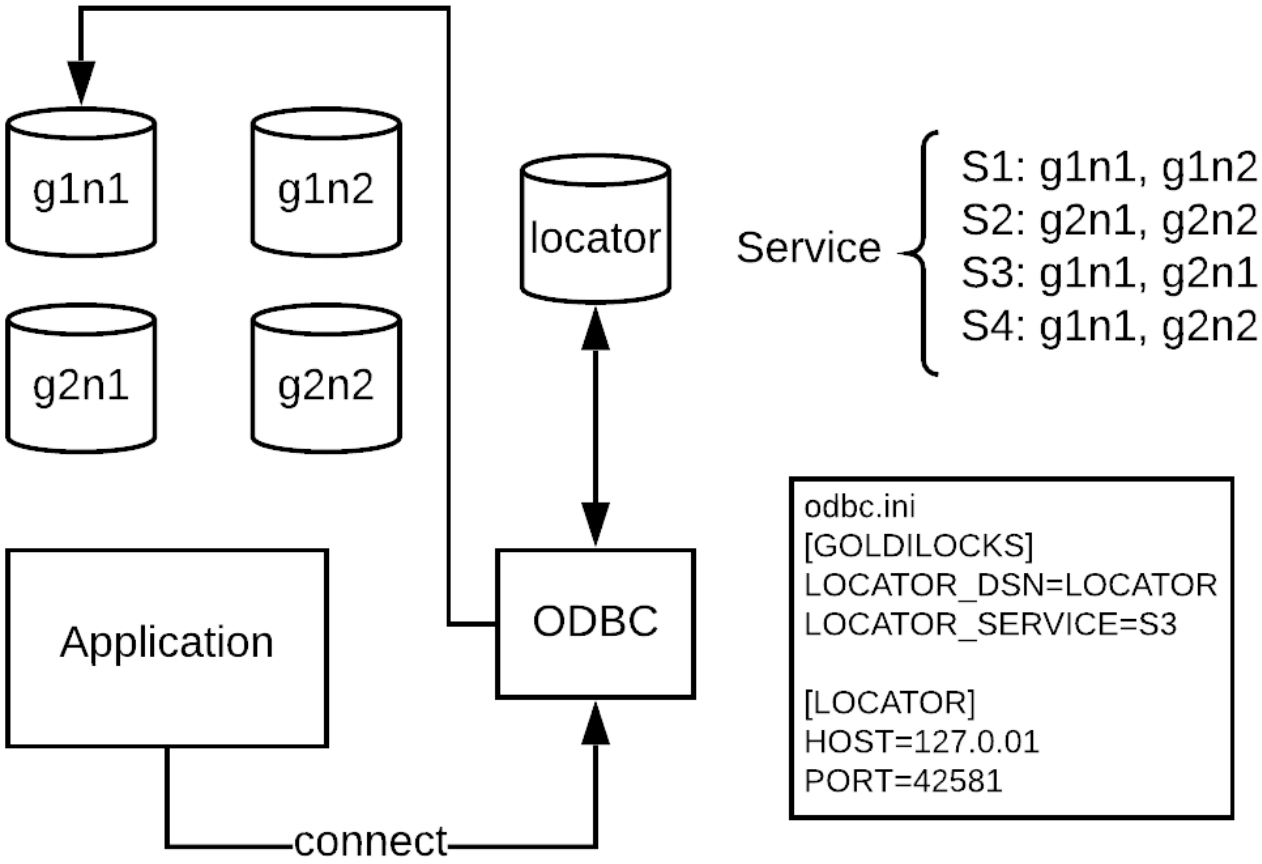
Connection Service

It provides a service feature which enables arbitrarily access any node among user-defined nodes even th ough the location information of a specific node in a cluster environment is unknown.

The service feature is that a user specifies nodes managed by glocator an arbitrary group. This service can be registered by using gloctl program. Service lists managed by glocator can also be viewed by using gloc tl program.

An application should be accessed by using ODBC driver, and LOCATOR_DSN and LOCATOR_SERVICE pr operty should be specified in **odbc.ini file**.

Figure 1 locator_service



The figure above describes that an application accesses to g1n1 belonging to service s3. The connecting s equence of ODBC driver by using the service is as same as the sequence of node registered in the service. If ODBC driver fails to connect to g1n1 in the example above, it will try to connect to the next node g2n1.

To use the service feature, the valid location information of a node should be input in glocator in advance.

Cluster Failover

glocator is helpful when a server proceeds the failover.

When the value of server property **CLUSTER_SPLIT_BRAIN_RESOLUTION_POLICY** is set to 1 or 2, and connection between nodes is disconnected in a cluster environment, a failover occurs, then each node proceeds the cluster failover and queries its viability to glocator through gagent.

The glocator which received a query determines the viability between two nodes which were disconnected, and transfer the result to the gagent which enquired the query.

Nodes received the viabilty results are terminated or proceeds the failover.

The cluster failover processing time of the server is relevant to various properties. Server property **LOCATOR_QUERY_TIMEOUT** sets the time waiting for the response after the server enquires to glocator, and the default value is 3 seconds.
Server property **CLUSTER_SPLIT_BRAIN_RETRY_COUNT** sets the number of enquiring again when glocator does not respond, and it is relevant to the cluster failover processing time. The default value is 1.

47.4 glocator Configuration

Configuration File and Environment Variable

glocator can use a file or environment variable to set the configuration.

An environment variable can be set by using the name of which 'LOCATOR_' is added as a prefix to a con figuration property name. For example, setting HOST in the configuration file has the same effect as speci fying $LOCATOR_HOST.

The contents of a configuration file takes precedence over the environment variable setting values.

glocator reads the configuration file by reading DSN as the default value of [LOCATOR].

glocator has $GOLDILOCKS_DATA/conf/goldilocks.glocator.conf file as its environment file. To alter the d riving environment of glocator, the file contents should be altered, or glocator should start after setting t he environment variables.

Configuration Properties

HOST

Item	Description
Name	HOST
Description	It is an IP address of which glocator binds.
Data type	ip address (ip v4)
Default value/ range	0.0.0.0

It is an IP address of which glocator binds for UDP communication.
It uses an IP address in IP v4 form.

PORT

Item	Description
Name	PORT
Description	It is a port of which glocator waits for Recv.
Data type	INT
Default value/ range	42581 / 1024~49151

It is a port of which glocator receives a packet through UDP communication.
The port from 1024 to 49151 can be used.

WORKER_COUNT

Item	Description
Name	WORKER_COUNT
Description	It is the number of threads processing a job.
Data type	INT
Default value/ range	1 / 1~8

It is the number of threads processing packets of which glocator received from a client or an internal proc ess.

MAX_NODE_COUNT

Item	Description
Name	MAX_NODE_COUNT
Description	It is the maximum number of connectable gagent.
Data type	INT
Default value/ range	64 / 1~8192

It is the maximum number of connectable gagent.

MESSAGE_QUEUE_SIZE

Item	Description
Name	MESSAGE_QUEUE_SIZE
Description	It is the queue size in which received packets are stored before processing them.
Data type	INT
Default value/ range	33554432 / 10485760~2147483648

It is the size of queue of which glocator stores packets received from a client or or an internal process bef ore processing them.
A packet is stored in a queue as a single item in message unit.

MESSAGE_ALLOCATOR_SIZE

Item	Description
Name	MESSAGE_ALLOCATOR_SIZE
Description	It is the size of an allocator which allocates an item to be stored in a message queue.
Data type	INT
Default value/ range	33554432 / 10485760~2147483648

It is the size of an allocator which is used to allocate an item (message) to be stored in a message queue.

PACKET_ALLOCATOR_SIZE

Item	Description
Name	PACKET_ALLOCATOR_SIZE
Description	It is the size of an allocator which allocates a packet when receiving packets through UDP c ommunication.
Data type	INT
Default value/ range	33554432 / 10485760~2147483648

It is the size of an allocator for a buffer which is allocated for glocator to receive packets.

SYSTEM_LOGGER_DIR

Item	Description
Name	SYSTEM_LOGGER_DIR
Description	It is the directory path of glocator trace log file.
Data type	String
Default value/ range	<GOLDILOCKS_DATA>/trc

It is the directory path in which system trace log file of glocator is stored. <GOLDILOCKS_DATA> of the de fault value is replaced with the environment variable value of $GOLDILOCKS_DATA.

SYSTEM_UDS_DIR

Item	Description
Name	SYSTEM_UDS_DIR
Description	It is the directory path in which unix domain socket file used in glocator is stored.
Data type	String
Default value/ range	/tmp (Maximum 60 byte)

It sets the directory path in which the unix domain socket file used in glocator is stored. The maximum le ngth of the directory should be set within 60 bytes.

LOCATION_FILE_DIR

Item	Description
Name	LOCATION_FILE_DIR
Description	It is the directory path in which the location file used in glocator is stored.
Data type	String
Default value/ range	<GOLDILOCKS_DATA>/db

It sets the directory path in which the location file used in glocator is stored.

LOCATION_FILE_NAME

Item	Description
Name	LOCATION_FILE_NAME
Description	It is the name of a location file which is used in glocator.
Data type	String
Default value/ range	glocator.dat

It sets the name of a location file which is used in glocator.
The default value is glocator.dat.

LOCATION_FILE_SIZE

Item	Description
Name	LOCATION_FILE_SIZE
Description	It is the initial size of a location file.
Data type	Int
Default value/ range	1048576 / 104576~2147483648

It sets the initial size of a location file used in glocator.

LOCATION_FILE_MAX_SIZE

Item	Description
Name	LOCATION_FILE_MAX_SIZE
Description	It is the maximum size of a location file.

Item	Description
Data type	Int
Default value/ range	10485760 / 104576~2147483648

It sets the maximum size of a location file used in glocator.

MESSAGE_TIMEOUT

Item	Description
Name	MESSAGE_TIMEOUT
Description	It sets the maximum time for glocator to wait to receive packets.
Data type	Int
Default value/ range	100 / 0 ~2147483648(단위 Second)

It sets the maximum time (seconds) for glocator to wait to receive packets. Packets exceeds the maximum time without being processed, are dumped.

ALTERNATE_LOCATORS

Item	Description
Name	ALTERNATE_LOCATORS
Description	It sets an alternate locator and replication of glocator.
Data type	String
Default value/ range	empty / 0 ~ 1024 bytes

It sets **Replication** of glocator.

SYNC_RETRY_COUNT

Item	Description
Name	SYNC_RETRY_COUNT
Description	It sets the number of retry when glocator fails to synchronize with an alternate locator.
Data type	Int
Default value/ range	1 / 0 ~ 5

It sets the number of redelivery of glocator synchronization.

glocator transfers the altered data to alternate locator when the data is altered, and it transfers the data again when it could not get any response.

SYNC_RESPONSE_TIMEOUT

Item	Description
Name	SYNC_RESPONSE_TIMEOUT
Description	It sets the time waits for the response for the synchronization of glocator.
Data type	Int
Default value/ range	5 / 1 ~ 20

It sets the time waits for the response for the synchronization data transferred from glocator.

KEEPALIVE_IDLE_TIME

Item	Description
Name	KEEPALIVE_IDLE_TIME
Description	tcp keepalive idle time for checking dead client session (sec)
Data type	INT
Default value/ range	1 / 1 ~ 16383

It is the (idle) duration which the TCP packet is not sent nor is received before sending a keep alive packet. In other words, if TCP packet is not exchanged during the time set in KEEPALIVE_IDLE_TIME, then it perf orms the keep alive mechanism to detect the dead connection.

KEEPALIVE_COUNT

Item	Description
Name	KEEPALIVE_COUNT
Description	tcp keepalive check count
Data type	INT
Default value/ range	5 / 1 ~ 10

It is the number of performing keep alive mechanism.

KEEPALIVE_INTERVAL

Item	Description
Name	KEEPALIVE_INTERVAL
Description	tcp keepalive packet interval (sec)
Data type	INT
Default value/ range	5

It is the time interval to send the keep alive packet.

48.

gagent

48.1 Overview of gagent

Definition

gagent is a utility communicating with **glocator** by being executed with each member node in GOLDILO CKS cluster system.

When **CLUSTER_SPLIT_BRAIN_RESOLUTION_POLICY** server property value of the node to which gagent b elongs is set to 1 or 2, and the connection between nodes is disconnected so the cluster failover is procee ding, then gagent enquires to glocator the validity of the node to which the gagent belongs.
For more information, refer to **Cluster Failover**.

Usage

```
gagent [options]
```

Options

help

Description

It outputs the help message.

Example

```
$ gagent --help
Usage:
 gagent [options]
Options:
-s  --start                     Start gagent
-t  --stop                      Stop gagent
-u  --status                    Get gagent status
-f  --conf                      Set configure file
-o  --home                      gmaster home path
-l  --silent                    Suppress display message
```

```
-r  --no-copyright              Suppress display copy right and version
-h  --help                      Print help message
```

start

Description

It starts gagent. If gagent having the same home directory already has been started, then an error occurs.

Example

```
$ gagent --start
gagent is started.
```

stop

Description

It stops gagent which is in operation. The same home directory should be set to stop gagent in operation.

Example

```
$ gagent --stop
gagent is stopped.
```

status

Description

It outputs the status message of gagent which is in operation. The same home directory should be set to check the status of gagent in operation.

Example

```
$ gagent --status
gagent(31938) is running.
```

conf

Description

It sets the configure file when starting gagent.

Example

```
$ gagent --start --conf goldilocks.gagent.conf
gagent is started.
```

home

Description

It sets the server home directory of GOLDILOCKS when starting gagent.
When a relative path is used, it searchs for the home directory based on <GOLDILOCKS_DATA>.
<GOLDILOCKS_DATA> is replaced with the environment variable value of $GOLDILOCKS_DATA.

Example

```
$ gagent --start --home g1n1_home --conf goldilocks.gagent.conf
gagent is started.
```

In the example above, gagent is started by setting <GOLDILOCKS_DATA>/g1n1_home to home.

silent

Description

It does not output the message of gagent about the execution.

Example

```
$ gagent --start --silent
```

no-copyright

Description

It does not output the message of gagent's copyright and version about the execution.

Example

```
$ gagent --start --no-copyright
gagent is started.
```

48.2 gagent Configuration

Configuration File and Environment Variable

gagent can use a file or environment variable to set the configuration.

An environment variable can be set by using the name of which 'AGENT_' is added as a prefix to a config uration property name. For example, setting HOST in the configuration file has the same effect as specifyi ng $AGENT_HOST.

The contents of a configuration file takes precedence over the environment variable setting values.

gagent has $GOLDILOCKS_DATA/conf/goldilocks.gagent.conf file as its environment file. To alter the driv ing environment of gagent, the file contents should be altered, or gagent should start after setting the en vironment variables.

Configuration Properties

HOST

Item	Description
Name	HOST
Description	It is an IP address of which gagent binds.
Data type	ip address (ip v4)
Default value/ range	0.0.0.0

It is an IP address of which gagent binds for UDP communication.
It uses an IP address in IP v4 form.

REQUEST_PORT

Item	Description
Name	REQUEST_PORT
Description	It is a port which gagent uses for request.
Data type	INT
Default value/ range	43581 / 1024~49151

It is a port of which gagent uses to request to glocator during tcp communication.
The port from 1024 to 49151 can be used.

RESPONSE_PORT

Item	Description
Name	RESPONSE_PORT
Description	It is a port which gagent uses for response.
Data type	INT
Default value/ range	43582 / 1024~49151

It is a port of which gagent uses to receive the response from glocator during tcp communication.
The port from 1024 to 49151 can be used.

LOCATOR_HOST

Item	Description
Name	LOCATOR_HOST
Description	It is an IP address of glocator.
Data type	ip address(ip v4)
Default value/ range	0.0.0.0

It is an IP address of glocator.

LOCATOR_PORT

Item	Description
Name	LOCATOR_PORT
Description	It is a port of which glocator waits for Recv.
Data type	INT
Default value/ range	42581 / 1024~49151

It is the port of glocator of when gagent sends packets to glocator.
The port from 1024 to 49151 can be used.

SYSTEM_LOGGER_DIR

Item	Description
Name	SYSTEM_LOGGER_DIR
Description	It is the directory path of gagent trace log file.
Data type	String
Default value/ range	<GOLDILOCKS_DATA>/trc

It is the directory path in which system trace log file of gagent is stored. <GOLDILOCKS_DATA> of the default value is replaced with the environment variable value of $GOLDILOCKS_DATA.

SYSTEM_UDS_DIR

Item	Description
Name	SYSTEM_UDS_DIR
Description	It is the directory path which stores Unix Domain Socket file of gagent.
Data type	String
Default value/ range	/tmp (Maximum 50 bytes)

It is the directory path which stores Unix Domain Socket file of gagent. The default value is /tmp. The directory length can be set up to 50 bytes.

ALTERNATE_LOCATORS

Item	Description
Name	ALTERNATE_LOCATORS
Description	It sets the alternate locators.
Data Type	String
Default value/ range	empty / 0 ~ 1024

If glocator set in gagent does not respond, then it proceeds to process the operation by replacing with an alternate locator.

The following is an example of a configure file which used ALTERNATE_LOCATOR.

```
[AGENT]
PORT=43581
LOCATOR_HOST=127.0.0.1
LOCATOR_PORT=42581
ALTERNATE_LOCATORS=(LOCATOR1,LOCATOR2)
[LOCATOR1]
```

```
HOST=127.0.0.1
PORT=42582
[LOCATOR2]
HOST=127.0.0.1
PORT=42583
```

KEEPALIVE_IDLE_TIME

Item	Description
Name	KEEPALIVE_IDLE_TIME
Description	tcp keepalive idle time for checking dead client session (sec)
Data type	INT
Default value/ range	1 / 1 ~ 16383

It is the (idle) duration which the TCP packet is not sent nor is received before sending a keep alive packet. In other words, if TCP packet is not exchanged during the time set in KEEPALIVE_IDLE_TIME, then it perf orms the keep alive mechanism to detect the dead connection.

KEEPALIVE_COUNT

Item	Description
Name	KEEPALIVE_COUNT
Description	tcp keepalive check count
Data type	INT
Default value/ range	5 / 1 ~ 10

It is the number of performing keep alive mechanism.

KEEPALIVE_INTERVAL

Item	Description
Name	KEEPALIVE_INTERVAL
Description	tcp keepalive packet interval (sec)
Data type	INT
Default value/ range	5

It is the time interval to send the keep alive packet.

49.

gloctl

49.1 Overview of gloctl

Definition

gloctl is an interactive utility provided by GOLDILOCKS, which provides location to **glocator** and edits it. gloctl communicates with glocator by using User Datagram Protocol (UDP) communication protocol.

Usage

```
gloctl [options]
```

Options

help

Description

It outputs the help message.

Example

```
$ gloctl --help
Usage:
 gloctl [options]
Options:
-c  --conf         User configure file
-i  --ip           Locator host ip
-p  --port         Locator port number
-o  --import       Import control FILE
-l  --silent       Suppress the display of result message and echoing commands
-r  --no-copyright Suppress display copy right and version
-h  --help         Print help message
```

conf

Description

It sets a **Configuration** file to communicate with glocator.
If it is not set, $GOLDILOCKS_DATA/conf/goldilocks.gloctl.conf is used.

Example

```
$ gloctl --conf gloctl.conf
gLoctl>
```

- The contents of gloctl.conf file above is as follows.

```
[GLOCTL]
# Port number (1024 ~ 49151)
PORT = 44581
# Locator address
LOCATOR_HOST = 127.0.0.1
# Locator port number (1024 ~ 49151)
LOCATOR_PORT = 42581
# Time out to receive message from glocator (second)
# second ( 0 ~ 2147483647 )
MESSAGE_TIMEOUT = 10
```

ip

Description

It sets IP address of glocator when starting gloctl. IP set value takes precedence when it is used together with DSN.

Example

```
$ gloctl --ip 127.0.0.1
gLoctl>
```

The port of glocator in the example above is obtained by using DSN default LOCATOR in odbc.ini.

port

Description

It sets the port of glocator when starting gloctl. The port set value takes precedence when it is used toget her with DSN.

Example

```
$ gloctl --port 42581
gLoctl>
```

IP of glocator in the example above is obtained by using DSN default LOCATOR in odbc.ini.

import

Description

It performs gloctl commands in batch within a file, not an interactive mode.

Example

```
$ gloctl --import import.txt
HELP
QUIT
IMPORT       {'FILE'}             Upload FILE to locations
EXPORT       {'FILE'}             Download locations to FILE
ADD MEMBER   {DSN¦{member_name {'host; port; db_home;'}}} Add member location
DROP MEMBER  {member_name}        Drop member location
SET TIMEOUT  {second}             Set time for session timeout
Add member succeeded.
```

- The contents of import.txt above is as follows.

```
HELP
ADD MEMBER G1N3 'HOST=127.0.0.1;PORT=24581;DB_HOME=g1n3_home;'
```

silent

Description

It does not output the message of gloctl about the execution.

Example

```
$ gloctl --silent --import import.txt
```

no-copyright

Description

It does not output the message of gloctl's copyright and version about the execution.

Example

```
$ gloctl --no-copyright
gLoctl>
```

49.2 Interactive Command References

ADD MEMBER

Syntax

```
ADD MEMBER {DSN | member_name {'host; port; db_home;'}}
```

Description

It adds a member to glocator. DSN existing in odbc.ini, a member name, and the required location inform ation should be input.

- The port is that of glsnr.
- ADD MEMBER is used to update the information of an existing member. In this case, if the member is in the process of the failover, then the update fails.

Example

ADD MEMBER by using DSN of odbc.ini.

```
gLoctl> ADD MEMBER G1N1
Add member succeeded.
gLoctl>
```

- The contents of odbc.ini is as follows.

```
[G1N1]
HOST=127.0.0.1
PORT=20101
DB_HOME= g1n1_home
LOCATOR_DSN=LOCATOR
```

- Describe the location of a member, and add it.

```
gLoctl> ADD MEMBER G1N2 'HOST=127.0.0.1; PORT=20201; DB_HOME=g1n2_home;'
Add member succeeded.
gLoctl>
```

ADD SERVICE

Syntax

```
ADD SERVICE service_name {'member_name; ... '}
```

Description

It adds a service to glocator. The node belonging to the service should be managed in **glocator**, and it is i gnored when the same node does not exist in glocator.

Example

Add the service *s1* by using ADD SERVICE.

```
gLoctl> ADD SERVICE S1 'G1N1;G1N2;G2N1'
Add service succeeded.
gLoctl>
```

- The contents of odbc.ini is as follows.

```
[GOLDILOCKS]
HOST=127.0.0.1
PORT=22581
LOCATOR_DSN=LOCATOR
LOCATOR_SERVICE=S1
[LOCATOR]
HOST=127.0.0.1
PORT=42581
```

When describing glocator properties in **odbcinst.ini file** and using **glocator**, then gsqlnet and ot her applications can access one of a server from the service list.

DROP MEMBER

Syntax

```
DROP MEMBER member_name
```

Description

It drops a member corresponding to member_name from glocator.

If a member is in the process of the failover, then the member can not be dropped.

Example

```
gLoctl> DROP MEMBER G1N2
Drop member succeeded.
gLoctl>
```

DROP SERVICE

Syntax

```
DROP MEMBER service_name
```

Description

It drops a service corresponding to service_name from glocator.

Example

```
gLoctl> DROP SERVICE S1
Drop service succeeded.
gLoctl>
```

EXPORT

Syntax

```
EXPORT {'FILE'}
```

Description

It receives the location information from glocator and stores it in a file.

Example

Download the location information in Location.txt file.

```
gLoctl> EXPORT 'Location.txt'
Export file succeeded.
gLoctl>
```

- The contents of Location.txt above is as follows.

```
[G1N1]
PORT = 20101
HOST = 127.0.0.1
DB_HOME = g1n1_home
[G1N3]
PORT = 24581
HOST = 127.0.0.1
DB_HOME = g1n3_home
```

HELP

Syntax

```
HELP
```

Description

It displays the commands list in an interactive mode of gloctl.

Example

```
gLoctl> HELP
HELP
QUIT
IMPORT {'FILE'} Upload FILE to locations
EXPORT {'FILE'} Download locations to FILE
ADD MEMBER {DSN|{member_name {'host; port; db_home;'}}} Add member location
DROP MEMBER {member_name} Drop member location
SET TIMEOUT {second} Set time for session timeout
ADD SERVICE  {service_name {'member_name; ... '} } Add service
DROP SERVICE {service_name}       Drop service
gLoctl>
```

IMPORT

Syntax

```
IMPORT {'FILE'}
```

Description

It transfers the location to glocator by using a file in ini form.

Example

It transfers the contents in Location.txt to glocator.

```
gLoctl> IMPORT 'Location.txt'
Import file succeeded.
gLoctl>
```

- The contents of Location.txt above is as follows.

```
[G1N1]
PORT = 20101
HOST = 127.0.0.1
DB_HOME = g1n1_home
[G1N3]
PORT = 24581
HOST = 127.0.0.1
DB_HOME = g1n3_home
```

QUIT

Description

It quits gloctl.

Syntax

```
QUIT
```

SET TIMEOUT

Description

It sets TIMEOUT of glocator.

Syntax

```
SET TIMEOUT {second}
```

Example

```
gLoctl> set timeout 120
Set timeout.
gLoctl>
```

49.3 Location File

Location file uploads or downloads the location information of a node from gloctl to glocator.

Description

Location file format is similar to that of **Data Source Specification** file.

```
[node_name]
HOST = host_address
PORT = port_no
DB_HOME = db_home_path
[SERVICE]
service_name = node_name {, node_name}*
```

Location keywords of Location file are as follows.

Table 49-1 Location information

Keyword	Description
node_name	It is the name of a node.
HOST	It is the IP address of a node.
PORT	It is the port number of glsnr which was executed in a node.
DB_HOME	It sets the home directory of a node.

[SERVICE] is a fixed keyword which lists the service hints. Service hints which are registered or to be regist ered in glocator are listed below SERVICE keyword.

HOST and PORT keywords should be input, and an error occurs if they are omitted.

The following is an example of configuring a location file.

```
[G1N1]
HOST = 192.168.0.101
PORT = 20101
DB_HOME = g1n1_home
[G1N2]
HOST = 192.168.0.102
PORT = 20102
DB_HOME = g1n2_home
```

```
[G2N1]
HOST = 192.168.0.201
PORT = 20201
DB_HOME = g2n1_home
[G2N2]
HOST = 192.168.0.202
PORT = 20202
DB_HOME = g2n2_home
[SERVICE]
service_1 = G1N1,G2N1
service_2 = G1N1,G1N2
```

49.4 Configuration

The environment file of gloctl is $GOLDILOCKS_DATA/conf/goldilocks.gloctl.conf. The environment file s hould be altered or the environment file should be set with **conf** option and restart gloctl, to alter the driv ing environment for gloctl.

Stop gloctl and alter the driving environment, then restart it to alter the environment of gloctl and to appl y it, while gloctl is being executed.

PORT

Item	Description
Name	PORT
Description	It is a port which is used by gloctl.
Data type	INT
Default value/ range	44581 / 1024 ~ 49151

It sets the port of a socket which is used by gloctl.

LOCATOR_HOST

Item	Description
Name	PORT
Description	It is the host address of glocator with which gloctl communicates.
Data type	STRING
Default value/ range	127.0.0.1

It sets the host address of glocator.

LOCATOR_PORT

Item	Description
Name	PORT
Description	It is the port of glocator with which gloctl communicates.

Item	Description
Data type	INT
Default value/ range	42581 / 1024 ~ 49151

It sets the port of glocator.

MESSAGE_TIMEOUT

Item	Description
Name	MESSAGE_TIMEOUT
Description	It sets the time of which gloctl waits for the response from glocator. (second)
Data type	INT
Default value/ range	10 / 0 ~ 2147483647

It sets the time of which gloctl waits for the response after transferring a packet to glocator.

Part VII.

Replication

50.

Overview

50.1 Overview of GOLDILOCKS Replication

Database replication efficiently and consistently distributes data for data recovery when an error occurs in the original database by using the remote database, so that it enables sustained service. It is also used to build multiple database with same data.

GOLDILOCKS replication supports CYCLONE, which is a tool using Change Data Capture (CDC) method.

> Change Data Capture (CDC)
> This method captures and analyzes the redo log generated during the database operation, and performs the replication.
> Only the asynchronous mode is supported because CYCLONE is available after the changes of t he database are stored in the redo log file.

The interval may occur between the original database and the remote database of the replication becaus e CDC method supports only the asynchronous mode. When the failure of original database or equipmen t occurs in the state of which the replication to the remote database is not completed, the remote databa se is in the state of which the interval is not reflected.

LOGMIRROR can be used to complement the data not reflected due to the interval of the asynchronous mode.

> LOGMIRROR is a tool which replicates the redo log files. It sends and stores the redo logs gener ated from the original database to the remote equipment without any loss of data. Therefore, u sing LOGMIRROR can prevent loss of data.

It provides CYFILE tool storing the transaction data which was processed and committed in the database i n Comma-Separated Values (CSV) format file by using CDC method. A user can directly process the data by using the stored file or converts the data according to its purpose.

> CYFILE directly captures the redo log file by using CDC method and analyzes the transaction in r eal time, then stores the contents in CSV format file.

50.2 Characteristics

CYCLONE

The followings are the characteristics of CYCLONE.

- It is a replication tool which uses Change Data Capture (CDC) method.
- It recognizes, analyzes and applies the changes in redo log files of the original database.
- It is operated being divided into master and slave.
- It supports active-active, active-standby.
- Replication is available in table unit.
- Several options can be set to improve the performance.
- It does not affect GOLDILOCKS in case of failure because it is operated as an independent process.
- Adding or changing the H/W is not required for operation.
- It allows various replication topology
- The operation between master and slave in standalone environment supports the relationship of 1 : 1 or N : N.
- The operation between master and slave in cluster environment supports the relationship of N : 1.

LOGMIRROR

The followings are the characteristics of LOGMIRROR.

- It transfers and stores the redo logs of the original database to the remote machine without any loss.
- It interworks with CYCLONE and replicates without any loss of data.
- Only one LOGMIRROR can be operated in a database.
- It is operated being divided into master and slave.

CYFILE

The followings are features of CYFILE.

- It uses Change Data Capture (CDC) method.
- It recognizes and analyzes updates in the redo log file of the original database.
- It captures in the unit of table.
- The process is independently operated, so GOLDILOCKS is not affected even when an error occurs.

- It supports standalone environment and cluster environment.

51.

CYCLONE

51.1 CYCLONE

CYCLONE is a replication tool which uses the Change Data Capture (CDC) method.

Overview

Database records the data changes which occur during the operation in the redo log file for the recovery. CDC performs the replication by analyzing the information of the recorded redo log file.

CYCLONE is driven being divided into master and slave. Master recognizes the changes of the redo log fil e in the original database and analyzes it, then transfers it to the slave. Slave analyzes the received data a nd performs the replication by using ODBC.

Operational Features

- It is divided into master and slave and is operated as master/slave in group unit.
- Master and slave of CYCLONE uses TCP/ IP communication.
- It can be executed/ terminated in group unit.
- The replication is executed in table unit, and a group may include one or more tables.
- A single table may be operated being included in several groups.
- The original database should have the redo log files, so DATA_STORE_MODE should be operated in T DS.
- The original database should have SUPPLEMENTAL LOGGING.
  - SUPPLEMENTAL LOGGING adds an additional information to the redo log files for the replication of CYCLONE.
- The original database should be operated in ARCHIVE LOG mode.
  - GOLDILOCKS recursively reuses the redo log files. When the redo log files are reused before the r eplication is completed, the replication becomes aborted and the existing replication from the cu rrent point is canceled then restarted. Therefore, the redo log files should be operated in ARCHIV E LOG mode to be archived.
- It does not affect GOLDILOCKS even when it fails to replicate because it is operated as an independe nt process.

Operational Restrictions

- The table participated in the replication should have a PRIMARY KEY.
- Only the committed transaction is allowed to be replicated. Therefore, the content is unknown to the slave before committing the transaction.
- The primary key update is not supported.
  - When the primary key is updated, the table is given up and it is not replicated any more.
- The table participated in the replication can not use the column which has Generated Always As Identity property.
- When Data Definition Language (DDL) is performed on the table in which the replication is being operated, it could be given up.
  - For more information, refer to **The occurrence of give up and whether to allow DDL statement according to DDL category** in table1.
  - If it does not comply with the processing procedure of table DDL of CYCLONE, then even the allowable DDL is given up.
  - It does not affect the replication of other tables.
  - It is same in case of the truncated table.
- The given-up table can be reset only when it was given up with --reset TABLE_NAME
- The columns configuring the table participating in the replication should have the same structure.(data type, order)
- The database performing the replication should have the same character encoding.
- The table stored in the recycle bin is not a replication target.

Table 51-1 The occurrence of give up and whether to allow DDL statement according to DDL category

DDL category	Occurrence of give up	Whether to allow DDL statement	DDL statement
Table DDL	X	-	CREATE TABLE
	O	X	DROP TABLE
	O	X	TRUNCATE TABLE
	O	X	ALTER TABLE .. RENAME
	X	-	ALTER TABLE .. STORAGE
	X	-	ALTER TABLE .. ADD SUPPLEMENTAL LOG
	O	X	ALTER TABLE .. DROP SUPPLEMENTAL LOG
	X	-	ALTER TABLE .. READ { ONLY | WRITE }
Column DDL	O	O	ALTER TABLE .. ADD COLUMN
	O	X	ALTER TABLE .. SET UNUSED COLUMN
	O	-	ALTER TABLE .. RENAME COLUMN
	X	-	ALTER TABLE .. ALTER COLUMN .. SET DEFAULT
	X	-	ALTER TABLE .. ALTER COLUMN .. DROP DEFAULT
	O	X	ALTER TABLE .. ALTER COLUMN .. SET NOT NULL

DDL category	Occurrence of give up	Whether to allow DDL statement	DDL statement
	O	X	ALTER TABLE .. ALTER COLUMN .. DROP NOT NULL
	O	X	ALTER TABLE .. ALTER COLUMN .. ALTER IDENTITY
	O	X	ALTER TABLE .. ALTER COLUMN .. DROP IDENTITY
	O	O	ALTER TABLE .. ALTER COLUMN .. SET DATATYPE
Constraint DDL	O	X	ALTER TABLE .. ADD CONSTRAINT
	O	X	ALTER TABLE .. DROP CONSTRAINT
	O	X	ALTER TABLE .. ALTER CONSTRAINT
	X	-	ALTER TABLE .. RENAME CONSTRAINT
Index DDL	O	X	CREATE UNIQUE INDEX
	X	-	CREATE INDEX
	O	X	DROP INDEX unique_index
	X	-	DROP INDEX non_unique_index
	X	-	ALTER INDEX .. STORAGE
	X	-	ALTER INDEX .. RENAME
Superordinate object of table	O	X	DROP USER
	O	X	DROP SCHEMA
	O	X	DROP TABLESPACE

The server property **DISABLE_DDL_CDC_GIVEUP** can disable the DDL statement which causes th e replication give up to avoid user created errors. Also, the server property **DISABLE_UPDATE_P K_CDC_GIVEUP** can disable primary key update.

DDL Processing during Replication

It should comply with the following procedure when the allowable DDL is performed for the table in whic h the replication is being performed.

1. Terminate all operating CYCLONE in master and slave before executing DDL in master.
   - The process in progress does not need to be terminated.
   - e.g. cyclone --master --stop / cyclone --slave --stop
2. Execute DDL in both master and slave. (It should be the allowable DDL.)
   - For more information, refer to **The occurrence of give up and whether to allow DDL statement ac cording to DDL category**
3. Restart CYCLONE of master and slave.
   - cyclone --master --start ... / cyclone --slave --start ...

- It does not require --reset option.
- It performs the recovery from the termination of the first stage and process DDL when restarting. (Refer to trace log)

Table 51-2 DDL application procedure

Item	MASTER CYCLONE	MASTER DB	SLAVE CYCLONE	SLAVE DB
1	CYCLONE STOP	-	-	-
2	-	-	CYCLONE STOP	-
3	-	Executing DDL	-	-
4	-	-	-	Executing DDL
5	CYCLONE START	-	-	-
6	-	-	CYCLONE START	-

If the table which executed DDL in CYCLONE master is given up after DDL application procedure is normally performed, then the following two should be checked.

1. Check if DDL performed in the master DB and in the slave DB is same, and check if the table structure is same after performing the DDL. (The order of performing DDL does not matter.)
2. Check if the performed DDL is allowable.

If the table is given up due to the reasons above, the given-up table should be replicated again from the current point by using reset in table unit and restarting it. (e.g. cyclone --start --master --reset TABLE_NAME).

Datatype Compatibility When Interworking with Other DBMS

- When target database of CYCLONE slave interworks with other DBMS instead of GOLDILOCKS, the column datatype of GOLDILOCKS table, source database, should be compatible with that of other DBMS table so that it can prevent the data loss or an error.
- Currently, it can interwork with Oracle, MySQL and DB2.
- When interworking with other DBMS, then SYNC and allowable DDL features are not available.

Oracle

Table 51-3 Datatype compatible with Oracle

GOLDILOCKS	Oracle	Remarks
Boolean	X	The corresponding datatype does not exist.
NATIVE_SMALLINT	NUMBER(5)	-
NATIVE_INTEGER	NUMBER(10)	-
NATIVE_BIGINT	NUMBER(19)	-
NATIVE_REAL	BINARY_FLOAT	-
NATIVE_DOUBLE	BINARY_DOUBLE	-
FLOAT	FLOAT	-
SMALLINT	NUMBER(5,0)	-
INTEGER	NUMBER(10,0)	-
BIGINT	NUMBER(19,0)	-
INT2	NUMBER(5,0)	-
INT4	NUMBER(10,0)	-
INT8	NUMBER(19,0)	-
REAL	FLOAT(24)	-
DOUBLE	FLOAT(53)	-
DOUBLE PRECISION	FLOAT(53)	-
FLOAT4	FLOAT(24)	-
FLOAT8	FLOAT(53)	-
DECIMAL	NUMERIC	-
NUMBER	NUMBER	-
NUMERIC	NUMERIC	-
CHAR	CHAR	-
VARCHAR	VARCHAR, VARCHAR2	-
BINARY	RAW	-
VARBINARY	RAW	-
DATE	DATE	-
TIME	X	The corresponding datatype does not exist.
TIMESTAMP	TIMESTAMP	-
TIMESTAMP WITH TIMEZONE	X	It does not support ODBC driver.
INTERVAL	X	It does not support ODBC driver.
LONG VARCHAR	LONG VARCHAR	-
LONG VARBINARY	LONG RAW	-

- The following four datatypes can not be replicated as described in a table above.
  - BOOLEAN
  - TIME
  - TIMESTAMP WITH TIMEZONE

- INTERVAL

MySQL

Table 51-4 Datatype compatible with MySQL

GOLDILOCKS	MySQL	Remarks
Boolean	BOOL, BOOLEAN	The corresponding type in MySQL is a synonym of TINYINT(1).
NATIVE_SMALLINT	SMALLINT	-
NATIVE_INTEGER	INT	-
NATIVE_BIGINT	BIGINT	-
NATIVE_REAL	FLOAT	-
NATIVE_DOUBLE	DOUBLE	The corresponding type in MySQL is a synonym of DOUBLE PRECISION, and it does not support the unsigned.
FLOAT	X	The precision is supported up to 53 in MySQL, and a data error occurs if the precision is bigger.
SMALLINT	SMALLINT	-
INTEGER	INTEGER	-
BIGINT	BIGINT	-
INT2	SMALLINT	-
INT4	INTEGER	-
INT8	BIGINT	-
REAL	REAL	-
DOUBLE	DOUBLE	-
DOUBLE PRECISION	DOUBLE PRECISION	-
FLOAT4	FLOAT(24)	A data error may occur.
FLOAT8	FLOAT(53)	A data error may occur.
DECIMAL	DECIMAL	-
NUMBER	X	-
NUMERIC	NUMERIC	-
CHAR	TEXT	-
VARCHAR	TEXT	-
BINARY	VARBINARY	-
VARBINARY	VARBINARY	-
DATE	DATETIME	-
TIME	TIME(6)	The data is not lost when fractional seconds precision is set.
TIMESTAMP	DATETIME(6)	The data is not lost when fractional seconds precision is set.

GOLDILOCKS	MySQL	Remarks
TIMESTAMP WITH TIMEZONE	X	It does not support ODBC driver.
INTERVAL	X	It does not support ODBC driver.
LONG VARCHAR	TEXT	-
LONG VARBINARY	BLOB	-

- The following four datatypes can not be replicated as described in a table above.
  - FLOAT
  - NUMBER
  - TIMESTAMP WITH TIMEZONE
  - INTERVAL

When 'lower_case_table_names' is set to '0' in Mysql settings, then it is case sensitive in schem a/ table name. Therefore, use a double quotation (") in schema/ table name when specifying th e configuration file.

DB2

Table 51-5 Datatype compatible with DB2

GOLDILOCKS	DB2	비고
Boolean	X	The corresponding datatype does no t exist.
NATIVE_SMALLINT	SMALLINT	-
NATIVE_INTEGER	INT	-
NATIVE_BIGINT	BIGINT	-
NATIVE_REAL	REAL	-
NATIVE_DOUBLE	DOUBLE	-
FLOAT	DOUBLE	-
SMALLINT	DECIMAL(5)	-
INTEGER	DECIMAL(10)	-
BIGINT	DECIMAL(19)	-
INT2	DECIMAL(5)	-
INT4	DECIMAL(10)	-
INT8	DECIMAL(19)	-
REAL	DOUBLE	-
DOUBLE	DOUBLE	-
DOUBLE PRECISION	DOUBLE	-
FLOAT4	DOUBLE	-
FLOAT8	DOUBLE	-

GOLDILOCKS	DB2	비고
DECIMAL	DECIMAL	The significant digit of DB2 is 31.
NUMBER	DECIMAL	The significant digit of DB2 is 31.
NUMERIC	DECIMAL	The significant digit of DB2 is 31.
CHAR	CHAR	The maximum size of DB2 is 254 bytes.
VARCHAR	VARCHAR	-
BINARY	CHAR(n) FOR BIT DATA	The maximum size of DB2 is 254 bytes.
VARBINARY	VARCHAR(n) FOR BIT DATA	-
DATE	DATE / TIMESTAMP(0)	DATE of DB2 is stored only in YYYY/MM/DD format.
TIME	X	-
TIMESTAMP	TIMESTAMP	-
TIMESTAMP WITH TIMEZONE	X	It does not support ODBC driver.
INTERVAL	X	It does not support ODBC driver.
LONG VARCHAR	CLOB	-
LONG VARBINARY	BLOB	-

Others

The replication moments are as follows.

- At the first run, the replication starts after master and slave are performed and the initialization is terminated.
- Even if it is restarted after terminated during the replication, the replication is continuously performed from the termination point. (Recovery feature)
- It should be restarted by using --reset option when giving up the existing replication and restarting from the current point.

51.2 Requirements

Original GOLDILOCKS: It is required to perform GOLDILOCKS preparations, user registration and privilege setting all.
Remote GOLDILOCKS: It is required to perform user registration and privilege setting only.

GOLDILOCKS Requirements

The followings should be set in GOLDILOCKS before starting the replication using CYCLONE.

SUPPLEMENTAL LOGGING

SUPPLEMENTAL LOGGING stores additional information in the redo log file for the replication of CYCLONE. The database restart is required to change the settings of the database in operation, but the database restart is not required to set SUPPLEMENTAL LOGGING for a particular table.

Setting SUPPLEMENTAL LOGGING in Database

- If the GOLDILOCKS property is set as SUPPLEMENTAL LOGGING, SUPPLEMENTAL LOGGING is recorded for every table.
- GOLDILOCKS restart is required.
- The appropriate information is added or updated in the property file.
  - Property file: goldilocks.properties.conf
  - Property setting: SUPPLEMENTAL LOG_DATA_PRIMARY_KEY = YES

Setting SUPPLEMENTAL LOGGING in Specific Table Participating in Replication

```
<add table supplemental log statement> ::=
    ALTER TABLE table_name
        ADD SUPPLEMENTAL LOG DATA ( PRIMARY KEY ) COLUMNS
    ;
```

ARCHIVE LOG

GOLDILOCKS reuses the redo log files recursively. When GOLDILOCKS reuses the redo log file being processed by CYCLONE, then CYCLONE does not proceed and is terminated. GOLDILOCKS should be operated in ARCHIVE LOG mode to ensure the continuous replication operation.

Changing Database in Operation to ARCHIVE LOG Mode

- Restart GOLDILOCKS.
- After database is shutdown, connect with sysdba and change it to ARCHIVE LOG mode in MOUNT p hase.

```
gSQL> \startup mount
Startup success
gSQL> alter database archivelog;
Database altered.
```

Setting ARCHIVE LOG Mode When Creating Database

- Update the property file before creating the database.
  - Property file: goldilocks.properties.conf
  - Property setting: ARCHIVELOG_MODE = 1

The path in which ARCHIVE LOG file is stored can be viewed and updated with 'ARCHIVELOG_ DIR'.

DATA_STORE_MODE

CYCLONE performs the replication by reading the redo log files of GOLDILOCKS. Therefore, GOLDILOCKS should be operated in Transactional Data Store (TDS) mode.

Changing DATA_STORE_MODE

- Restart the database.
- Add or update the corresponding information in the property file.
  - Property file: goldilocks.properties.conf
  - Property setting: DATA_STORE_MODE = 2

If the value of DATA_STORE_MODE is 1, it indicates Concurrent Data Store (CDS), and if it is 2, i t indicates Transactional Data Store (TDS).

Registering User and Setting Privileges

CYCLONE retrieves and manipulates the required information during the operation. The user operating CYCLONE and the proper privileges for the user are required.

Creating Database User

A specific user should be added to operate CYCLONE, and the corresponding user should be added for all GOLDILOCKS of which CYCLONE is operated in master, slave mode.

```
<user definition> ::=
    CREATE USER user_identifier IDENTIFIED BY password
    [ DEFAULT TABLESPACE tablespace_name ]
    [ TEMPORARY TABLESPACE tablespace_name ]
    [ INDEX TABLESPACE {tablespace_name¦NULL} ]
    [ <schema clause> ]
    ;
<schema clause> ::=
      WITH SCHEMA [schema_name]
    ¦ WITHOUT SCHEMA
```

The following is an example of creating the user cdc_user with the password cdc_password.

```
gSQL> CREATE USER cdc_user IDENTIFIED BY cdc_password;
```

Database Privileges

Granting User Access Privilege

The following is an example for granting the access privilege to cdc_user. It should be set on all GOLDILOCKS which is operated in both master/ slave mode.

```
gSQL> GRANT CREATE SESSION ON DATABASE TO cdc_user;
```

Granting Privilege for Altering Table

The following is an example of granting the table updating privilege to cdc_user. It is set on GOLDILOCKS which is operated in slave mode.

```
gSQL> GRANT INSERT ANY TABLE, DELETE ANY TABLE, UPDATE ANY TABLE ON
DATABASE TO cdc_user;
```

Tablespace Privileges

The privileges on the data tablespace and temporary tablespace should be set. The following is an exampl e of granting the privilege for using the default tablespace to cdc_user. It is set on GOLDILOCKS which is operated in slave mode.

```
gSQL> GRANT CREATE OBJECT ON TABLESPACE mem_data_tbs TO cdc_user;
gSQL> GRANT CREATE OBJECT ON TABLESPACE mem_temp_tbs TO cdc_user;
```

Schema Privileges

The schema privileges for creating and managing meta managed in CYCLONE should be set. The followin g is an example granting the schema privilege to cdc_user. It is set on GOLDILOCKS which is operated in s lave mode.

```
gSQL> GRANT CREATE TABLE, CREATE INDEX, CREATE SEQUENCE, CREATE VIEW,
ADD CONSTRAINT ON SCHEMA cdc_user TO cdc_user;
```

51.3 Configuration

Configuration File

When performing CYCLONE, the information and options required for operating are set by using the con figuration file.

- When a specific configuration file is not set by using the --conf option, a specific file in the $GOLDILO CKS_DATA/conf directory is read. cyclone.master.conf file is read when it is operated in master mode and cyclone.slave.conf file is read when it is operated in slave mode.

Table 51-6 Configuration file options

Name	Description	Coverage
COMM_CHUNK_ COUNT	It sets the size of BUFFER for communication.	Master/ slave
DSN	It sets Data Source Name.	Master/ slave
GROUP_NAME	It sets the group name.	Master/ slave
HOST_IP	It sets the host IP address of which GOLDILOCKS operates.	Master/ slave
HOST_EXTERNAL _IP	It is used when slave GOLDILOCKS IP to which the cyclone master is to be co nnected is different from HOST_IP. (When master and slave are on wan sect ion.)	Slave
HOST_PORT	It sets the host port of which GOLDILOCKS operates.	Master/ slave
PORT	It sets the port for master/ slave communication.	Master/ slave
USER_ID	It sets the user name.	Master/ slave
USER_PW	It sets the user password.	Master/ slave
USER_ENCRYPT_ PW	It sets the encrypted password for DB user.	Master/ slave
CAPTURE_TABLE	It sets the table to be replicated.	Master
LOG_PATH	It is used when interworking with LOGMIRROR and it sets the location of th e redo log file.	Master
PROTOCOL	It sets the connection type which is to be connected to GOLDILOCKS. (DA o r TCP)	Master/ slave
READ_LOG_BLO CK_COUNT	It sets the amount of data to be read at a time when operating CAPTURE.	Master
TRANS_SORT_AR EA_SIZE	It sets the size of the BUFFER to be allocated to CAPTURE.	Master
TRANS_FILE_PAT H	It sets the location in which the temporarily generated file is to be stored wh en operating CAPTURE.	Master
SYNCHER_COUN T	It is applied when using SYNC feature, and it sets the number of SYNCHER s imultaneously performing the data insertion.	Master

Name	Description	Coverage
SYNC_ARRAY_SIZE	It is applied when using SYNC feature, and it sets the array size which insert data at a time.	Master
GIVEUP_INTERVAL	If the replication performance speed is lower than GOLDILOCKS performance, it is set to stop the replication.	Master
APPLIER_COUNT	It sets the number of APPLIER simultaneously performed during the replication.	Slave
APPLY_COMMIT_SIZE	It sets the maximum size for COMMIT during the replication.	Slave
APPLY_TABLE	It sets the table to which the replicated table is to be applied.	Slave
MASTER_IP	It sets the IP address of the equipment of which CYCLONE master is operated.	Slave
PROPAGATE_MODE	It sets whether to propagate the data applied by CYCLONE.	Slave
SEPARATE_CONFLICT_LOG	It sets whether to separate the conflict log from the trace log when storing it. (The default value is 0.) • 0: It is not separated. • 1: It is separated then stored. (The filename is cyclone_conflict_GROUPNAME.log.)	Slave
UPDATE_APPLY_MODE	It distinguishes the operation when updating. (The default value is 0.) • 0: It is updated only when the primary keys are same. • 1: It is updated only when the primary key and the value before the update are same. • 2: It is updated only when the primary keys are same. It compares the value before and after the update, then leaves a log if the values are different.	Slave
TCP_NODELAY	It sets TCP_NODELAY option of a socket. (The default value is 1.) • 0: TCP_NODELAY off • 1: TCP_NODELAY on	Master
HEARTBEAT_TIMEOUT	It sets the maximum time (second) maintaining connection if the connection is not smooth due to network disconnection or system error after replication connection.	Master/ slave
SKIP_COMMENT	It enters the text for transaction skip. If the same text is entered as commit comment in master, then it skips that transaction instead of replicating it.	Master
LOG_CAPTURE_INTERVAL_1	It sets the execution cycle of capture. If the value is not changed after executing 10 times with that value, then it is converted to the value of LOG_CAPTURE_INTERVAL_2 and performs capture. (The default value is 0.2 seconds.)	Master
LOG_CAPTURE_INTERVAL_2	It sets the execution cycle of capture. If the value is not changed after executing with the value of LOG_CAPTURE_INTERVAL_1, then it sets the execution cycle of capture. (The default value is 1 second.)	Master
CLUSTER	It specifies the connection information of a master when the master is in cluster environment. A master consists of the following three information. • ID: It is a delimiter and sets to 1 or more value.	Slave

Name	Description	Coverage
	• MASTER_IP • PORT	
ORACLE_DRIVER	It specifies the file location of the Oracle ODBC driver provided by Oracle.	Slave
MYSQL_DRIVER	It specifies the file location of the MySQL ODBC driver provided by MySQL.	Slave
MYSQL_DATABASE	It specifies the database name of MySQL to replicate.	Slave
PACKET_COMPRESSION_MODE	It sets whether to compress the data of communication of master and slave. (1: Enable, 0: Disable, Default: Enable)	Master
DB2_DRIVER	It specifies the file location of DB2 ODBC driver provided by DB2	Slave
DB2_DATABASE	It specifies the DB2 database name to replicate.	Slave

Configuration Option

COMM_CHUNK_COUNT

- It sets the number of buffers(chunk) used in data communication between master and slave of CYCLONE.
- It allocates the resource with 16M * N (the set value).
- The default value is 32, and the actual size is 16M * 32 = 512 M.
- The minimum value is 10.
- It can be set in master and slave.
    - If too small value is set so the buffer is not enough, the performance becomes poor.

- Settings applied to all groups

```
COMM_CHUNK_COUNT = 10
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    COMM_CHUNK_COUNT=20
    ....
    ....
}
```

DSN

- It sets the data source name which is required when connecting to GOLDILOCKS.
- It can be set in master and slave.
- The default value is GOLDILOCKS.

- Settings applied to all groups

```
DSN=GOLDILOCKS
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    DSN=GOLDILOCKS
    ....
    ....
}
```

GROUP_NAME

- It is necessary for distinguishing the CYCLONE operation within the equipment and it is a unit of gen erating the operation process.
- It is the delimiter of when starting or terminating CYCLONE in group unit.
- After it is set, it should not be changed. If changed, it is regarded as a new group.
- It can be set in master and slave.
  - The connection between master and slave is separated not by GROUP_NAME but by PORT.
  - GROUP_NAME should be unique in the same equipment.
- The braces { } should be used.

```
GROUP_NAME = testGROUP
{
    ....
    ....
}
```

HOST_IP

- It sets the IP address of GOLDILOCKS to be connected by CYCLONE.
- It can be set in master and slave.
  - It is valid only when PROTOCOL is set to TCP.

- Settings applied to all groups

```
HOST_IP = 127.0.0.1
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    HOST_IP = 127.0.0.1
    ....
    ....
}
```

HOST_EXTERNAL_IP

- It can be set in slave.
- CYCLONE master access GOLDILOCKS of the slave's side when it is operated in sync, and it is used if s lave GOLDILOCKS IP to which the cyclone master is to be connected is different from HOST_IP.
  - It is used when IP on lan and IP on wan is different because master and slave are on wan section.
  - CYCLONE slave transfers this value to CYCLONE master, and it is used in CYCLONE master.

- Settings applied to all groups

```
HOST_EXTERNAL_IP = 192.168.0.120
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    HOST_EXTERNAL_IP = 192.168.0.120
    ....
    ....
}
```

HOST_PORT

- It sets the port of GOLDILOCKS to be connected by CYCLONE.
- It should be set together with HOST_IP.
- It can be set in master and slave.

- Settings applied to all groups

```
HOST_PORT = 22531
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    HOST_PORT = 22531
    ....
    ....
}
```

PORT

- It sets the PORT used in the communication between master and slave.
- The duplicated PORT should not be set between the groups, and the unique value should be used for each GROUP.
- It is mandatory be set and it can be set only within GROUP_NAME.

- It can be set only within a group.

```
GROUP_NAME = testGROUP
{
    PORT = 21102
    ....
    ....
}
```

USER_ID

- It sets the user ID required to access GOLDILOCKS.
- It can be set in master and slave.

- Settings applied to all groups

```
USER_ID = testID
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    USER_ID = testID
```

```
    ....
    ....
}
```

USER_PW

- It sets the user password required to access GOLDILOCKS.
- It can be set in master and slave.

- Settings applied to all groups

```
USER_PW = testPW
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    USER_PW = testPW
    ....
    ....
}
```

USER_ENCRYPT_PW

- It sets the user password which is required for the access to GOLDILOCKS by encrypting.
- It is used instead of USER_PW.
- Encrypted user password is created by [cyclone --encrypt *user password* --key *the key to be encrypted*].
- If this value is used, then --key option should be used when executing cyclone. (In this case, the key v alue as same as that created with --encrypt should be used.

- Settings applied to all groups

```
USER_ENCRYPT_PW = 't33KImiqvhqNyfN+uZmFrw=='
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    USER_ENCRYPT_PW = 't33KImiqvhqNyfN+uZmFrw=='
    ....
```

```
    ....
}
```

CAPTURE_TABLE

- It sets the table to be replicated.
  - It is set in *schema_name.table_name* format.
- It can set only in master.
- It can be set only within a group.
- When specifying several tables, the parentheses ( ) should be used.
- The table stored in the recycle bin can not be set to be replicated.

```
GROUP_NAME = testGROUP
{
    CAPTURE_TABLE =
    (
        testSchema1.testTable1,
        testSchema1.testTable2,
        testSchema2.testTable1
    )
}
```

LOG_PATH

- It is used when interworking with LOG MIRROR.
- It sets the path of the redo log files stored by LOGMIRROR.
  - The absolute path should be used.
  - The path should be specified by using single quote (').

- Settings applied to all groups

```
LOG_PATH = '/data/wal/'
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    LOG_PATH = '/data/wal/'
    ....
    ....
}
```

PROTOCOL

- It sets the type to connect to GOLDILOCKS in operation.
- It can be set to DA or TCP.
- If PROTOCOL is set to DA, neither HOST_IP nor is HOST_PORT used when accessing to GOLDILOCKS.
  - However, in a slave, HOST_IP and HOST_PORT can be used for SYNC even when PROTOCOL is D A.
- The default value is DA.

- Settings applied to all groups

```
PROTOCOL = DA
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    PROTOCOL = DA
    ....
    ....
}
```

READ_LOG_BLOCK_COUNT

- It sets the number of the log blocks to be read at a time when capturing redo log file.
- It can be set only in master.
- The size of a log block is 512 bytes.
- The default value is 40960, and the actual read size is 20 MBytes.
  - The minimum value is 100.

- Settings applied to all groups

```
READ_LOG_BLOCK_COUNT = 1024
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    READ_LOG_BLOCK_COUNT = 1024
    ....
    ....
}
```

TRANS_SORT_AREA_SIZE

- It sets the memory space required for capturing the redo log files.
- It can be set only in master.
- The unit is MB (megabytes).
- The default value is 500 MB.
  - The minimum value is 10 MB.
- If the value is set too small, the performance becomes poor.

- Settings applied to all groups

```
TRANS_SORT_AREA_SIZE = 300
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    TRANS_SORT_AREA_SIZE = 300
    ....
    ....
}
```

TRANS_FILE_PATH

- If a space bigger than TRANS_SORT_AREA_SIZE is required, the temporary file is created. It sets the p ath in which the temporary file is to be stored.
  - The absolute path should be used.
  - The path should be specified by using single quote (').
- It can be set only in master.

- Settings applied to all groups

```
TRANS_FILE_PATH = '/data/TmpTrans'
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    TRANS_FILE_PATH = '/data/TmpTrans'
    ....
    ....
}
```

SYNCHER_COUNT

- It is used for synchronization.
  - It sets the number of SYNCHERs participating in synchronization.
  - It is input in the number unit.
- It is set only in master.

- Settings applied to all groups

```
SYNCHER_COUNT = 8
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    SYNCHER_COUNT = 8
    ....
    ....
}
```

SYNC_ARRAY_SIZE

- It is used for syncronization.
  - It sets the unit of record to be inserted at a time when performing synchronization.
  - It is input in the number unit.
- It can be set only in master.

- Settings applied to all groups

```
SYNC_ARRAY_SIZE = 1000
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    SYNC_ARRAY_SIZE = 1000
    ....
    ....
}
```

GIVEUP_INTERVAL

- It gives up the replication and terminates CYCLONE if the INTERVAL between GOLDILOCKS and CYCL ONE is bigger than the set value.
  - It is input in the number of REDO LOG BLOCK unit.
- It can be set only in master.

- Settings applied to all groups

```
GIVEUP_INTERVAL = 10000
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    GIVEUP_INTERVAL = 10000
    ....
    ....
}
```

APPLIER_COUNT

- It sets the number of APPLIER executing replication.
  - It indicates a parallel factor.
- The default value is 6, and 6 sessions are created.
  - The maximum value is not limited, but too high value can cause the contention between APPLIER s.
  - The set value significantly affects on performance.
  - The session is created according to the set value and the replication is simultaneously performed.
- It can be set only in slave.

- Settings applied to all groups

```
APPLIER_COUNT = 16
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    APPLIER_COUNT = 16
    ....
    ....
```

```
}
```

APPLY_COMMIT_COUNT

- It sets the number of transaction executing COMMIT when the replication is performed.
  - It executes COMMIT after performing the transaction once when replicating transactions commit ted in the original database to the remote database.
  - The set value indicates the maximum value.
- The set value affects on performance.
- It can be set only in slave.

- Settings applied to all groups

```
APPLY_COMMIT_COUNT = 1000
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    APPLY_COMMIT_COUNT = 1000
    ....
    ....
}
```

APPLY_TABLE

- It sets the table to which the replicated table is to be applied.
  - It is set in *replicating table name TO table name to which the replicated table is applied* format.
  - The table names are not necessary to be same.
- It can be set only in slave.
- It can be set only within a group.
- The parentheses ( ) should be used when specifying several tables.

```
GROUP_NAME = testGROUP
{
    APPLY_TABLE =
    (
        testSchema1.testTable1 TO testSchema1.testTable1,
        testSchema1.testTable2 TO testSchema2.testTable3,
        testSchema2.testTable3 TO testSchema2.testTable4
    )
```

```
}
```

MASTER_IP

- It sets the IP address of the device which CYCLONE master operates.
- It can be set only in slave.

- Settings applied to all groups

```
MASTER_IP = 192.168.0.100
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    MASTER_IP = 192.168.0.100
    ....
    ....
}
```

PROPAGATE_MODE

- If CYCLONE is circularly configured, it sets whether or not another CYCLONE applies the transactions of which a CYCLONE applied.
- It can be set only in slave.
  - The default value is '0' and it does not PROPAGATE.
  - If it is set to 1, it is PROPAGATEd. If it is set to 0, it is not PROPAGATEd.

- Settings applied to all groups

```
PROPAGATE_MODE = 1
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    PROPAGATE_MODE = 1
    ....
    ....
}
```

SEPARATE_CONFLICT_LOG

- It sets whether to separate the conflict log from the trace log when storing it.
- It can be set only in slave.
  - Set it to 1 to separate it, or set it to 0 not to separate it.
    - The default value is 0.
    - If it is set to 1, then it is stored in cyclone_conflict_GROUPNAME.log file.
- It can not be set to apply to a specific group.

- Settings applied to all groups

```
SEPARATE_CONFLICT_LOG = 1
```

UPDATE_APPLY_MODE

- It is used for the update operation in slave.
  - 0: If only the primary keys are same, the update operation is performed. (The default value)
  - 1: If the primary key is same with the value of before updating, the update operation is performed.
  - 2: If only the primary keys are same, the update operation is performed. The values before and after updating are compared and if they are different, then it leaves the logs.
- If it is set to 0, the previous value is not checked and the update is performed.
- If it is set to 1 and the previous value is different, the update is failed and leaves the conflict log.
- If it is set to 2 and the previous value is different, the update is succeeded and leaves the conflict log.

- Settings applied to all groups

```
UPDATE_APPLY_MODE = 1
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    UPDATE_APPLY_MODE = 1
    ....
    ....
}
```

For a column whose data type is long varchar or long varbinary, then only the lengths are compared for the performance reason.

TCP_NODELAY

- It is used only in master.
- It sets TCP_NODELAY option for the CDC transfer socket. (This option does not affect the sync. It is fixed to TCP_NODELAY on.)
  - 0: It offs the socket TCP_NODELAY option.
  - 1: It ons the socket TCP_NODELAY option. (Default)

- Settings applied to all groups

```
TCP_NODELAY = 1
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    TCP_NODELAY = 1
    ....
    ....
}
```

HEARTBEAT_TIMEOUT

- It can be set in master and slave.
- It sets the maximum time (second) maintaining connection if the connection is not smooth due to network disconnection or system error after replication connection between master and slave.
- The default value is 30 (seconds).
  - The minimum value is 10 (seconds).

- Settings applied to all groups

```
HEARTBEAT_TIMEOUT = 40
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    HEARTBEAT_TIMEOUT = 40
    ....
    ....
}
```

SKIP_COMMENT

- It can be set in master.
- It is available when a specific transaction is not replicated.
- If the same value is input as a comment when executing transaction commit after setting the corresp onding value, then it skips that transaction instead of replicating it.

- Settings applied to all groups

```
SKIP_COMMENT = 'DO_SKIP'
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    SKIP_COMMENT = 'DO_SKIP'
    ....
    ....
}
```

LOG_CAPTURE_INTERVAL_1

- It can be set in master.
- It sets the execution cycle of capture operated in master. The execution cycle is set in millisecond.
- It is used to quickly detect the changes of the redo log file.
- If the value of redo log file is not changed after executing 10 times with that value, then it is converte d to the value of LOG_CAPTURE_INTERVAL_2 and performs capture.
- The default value is 200 (0.2 seconds), and the unit is millisecond.

- Settings applied to all groups

```
LOG_CAPTURE_INTERVAL_1 = 200
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    LOG_CAPTURE_INTERVAL_1 = 200
    ....
    ....
}
```

LOG_CAPTURE_INTERVAL_2

- It can be set in master.
- It sets the execution cycle of capture operated in master. The execution cycle is set in millisecond.
- It is used to quickly detect the changes of the redo log file.
- If the value of redo log file is not changed after executing 10 times with the value of LOG_CAPTURE_INTERVAL_1, then it is converted to the corresponding value and performs capture.
- The default value is 1000(1 second), and the unit is millisecond.

- Settings applied to all groups

```
LOG_CAPTURE_INTERVAL_2 = 1000
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    LOG_CAPTURE_INTERVAL_2 = 1000
    ....
    ....
}
```

CLUSTER

- It can be set in slave.
- It sets the connection information of N masters whih is operated in cluster environment. (It consists o f ID, MASTER_IP and PORT information.)
  - ID: It is an identifier of master in slave, and only number can be input. Once the value is set, it sho uld not be altered.
  - MASTER_IP: It sets IP in which master is being operated.
  - PORT:It sets port in which master is being operated.

- Cluster can be set only within a group.

```
GROUP_NAME = testGROUP
{
    CLUSTER = (ID=1, MASTER_IP=192.0.0.100, PORT=21102),
              (ID=2, MASTER_IP=192.0.0.101, PORT=21103)
    ....
    ....
}
```

ORACLE_DRIVER

- It can be set in slave.
- It specifies the location and name of ODBC driver file provided by Oracle.
  - Currently it supports Oracle 11g and 12c.

- It can be set in any group.

```
ORACLE_DRIVER = '/app/oracle/product/11.2.0/db_1/lib/libsqora.so.11.1'
GROUP_NAME = testGROUP
{
    ....
    ....
}
```

MYSQL_DRIVER

- It can be set in slave.
- It specifies the location and name of ODBC driver file provided by MySQL.

- It can be set in any group.

```
MYSQL_DRIVER = '/usr/lib64/libmaodbc.so'
GROUP_NAME = testGROUP
{
    ....
    ....
}
```

MYSQL_DATABASE

- It can be set in slave.
- It specifies the DATABASE name of MySQL to replicate.
- The DATABASE name of MySQL has the same meaning as the SCHEMA name.

- It can be set in any group.

```
MYSQL_DATABASE = mysql
GROUP_NAME = testGROUP
{
    ....
```

```
    ....
}
```

PACKET_COMPRESSION_MODE

- It can be set in master.
- It sets whether to compress the data which is transferred from master to slave.
  - 1: Enable (Default)
  - 0: Disable

- It can be set in a specific group.

```
GROUP_NAME = testGROUP
{
    PACKET_COMPRESSION_MODE = 1
    ....
    ....
}
```

DB2_DRIVER

- It can be set in slave.
- It specifies the location and name of ODBC driver file provided by DB2.

- It can be set in any group.

```
DB2_DRIVER = '/usr/lib64/libdb2o.so'
GROUP_NAME = testGROUP
{
    ....
    ....
}
```

DB2_DATABASE

- It can be set in slave.
- It specifies the DATABASE name of DB2 to replicate.

- It can be set in any group.

```
DB2_DATABASE = db2
GROUP_NAME = testGROUP
{
    ....
    ....
}
```

51.4 Operating

CYCLONE can be operated in D/A or C/S environment of GOLDILOCKS.

CONFIG file or PROTOCOL configuration of ODBC.INI should be used according to the environments as f ollows. CONFIG configuration takes precedence over ODBC.INI configuration.

Table 51-7 CONFIG, ODBC.INI

Configuration	Description
PROTOCOL=DA	It is used in D/A environment. (Default)
PROTOCOL=TCP	It is used in C/S environment.

The running contents during the operation can be viewed through trace log.

Item	File
Cyclone master	$GOLDILOCKS_DATA/trc/cyclone_master.trc
Cyclone slave	$GOLDILOCKS_DATA/trc/cyclone_slave.trc

For more information about error message and its handling method stored in trace log, refer to **Troubleshooting of CYCLONE**.

GOLDILOCKS Connection Policy

It is the policy for CYCLONE to connect to GOLDILOCKS.

There are two ways to connect to GOLDILOCKS. One is that CYCLONE connects to local GOLDILOCKS, a nd the other is that CYCLONE master remotely connects to slave side GOLDILOCKS on the slave's side wh en performing SYNC. Both use CYCLONE config value is preferentially used, then use odbc.ini. However, when performing SYNC, PROTOCOL value is ignored and it is set to TCP no matter what.

The config properties relating to GOLDILOCKS connection are DSN, PROTOCOL, HOST_IP, HOST_EXTERN AL_IP, HOST_PORT, USER_ID, USER_PW.
(USER_ENCRYPT_PW is listed as USER_PW because it replaces USER_PW.)

Correct Examples

- It is available just with USER_ID and USER_PW because it is connected as D/A.

Item	File
CONFIG	USER_ID=test, USER_PW=test
GOLDILOCKS configuration of odbc.ini	-

- The information should exist either in CONFIG or in odbc.ini. HOST_IP and HOST_PORT are ignored b ecause it is operated in D/A.

Item	File
CONFIG	HOST_IP=127.0.0.1, HOST_PORT=22581,USER_ID=test
GOLDILOCKS configuration of odbc.ini	USER_PW=test

- USER_ID is operated with the test account because it is connected with TCP and CONFIG takes prece dence.

Item	File
CONFIG	PROTOCOL=TCP, USER_ID=test, USER_PW=test
GOLDILOCKS configuration of odbc.ini	HOST_IP=127.0.0.1, HOST_PORT=22581, USER_ID=test2, USER_PW=test2

- HOST_EXTERANL_IP can not be set in slave even though it is connected as D/A.

Item	File
CONFIG	PROTOCOL=DA, HOST_EXTERNAL_IP=192.168.0.10, USER_ID=test
GOLDILOCKS configuration of odbc.ini	HOST_IP=127.0.0.1, HOST_PORT=22581,USER_PW=test

Wrong Examples

- It is connected as D/A, but USER_PW does not exist.

Item	File
CONFIG	HOST_EXTERNAL_IP=192.168.0.10
GOLDILOCKS configuration of odbc.ini	PROTOCOL=DA,USER_ID=test

- It is connected as TCP, but HOST_IP does not exist.

Item	File
CONFIG	HOST_EXTERNAL_IP=192.168.0.10, USER_ID=test
GOLDILOCKS configuration of odbc.ini	PROTOCOL=TCP, HOST_PORT=22581,USER_PW=test

Executing Option

CYCLONE should be used with the following options at run-time.

Table 51-8 Executing options

Option	Description	Remarks
--start | -s	It starts CYCLONE.	It should be used together with --master | --slave.
--stop | -t	It terminates CYCLONE.	It should be used together with --master | --slave.
--master | -m	It is performed in master mode.	It should be used together with --start | --stop.
--slave | -l	It is performed in slave mode.	It should be used together with --start | --stop.
--status | -u	It displays the operating status of CYCLONE.	It should be used together with --master | --slave.
--conf | -c	It sets the path of configuration file which is required when executing CYCLONE.	It is input in --conf CONFIG_FILE format. It should be used together with --start. If it is not explicitly set, master uses $GOLDILOCKS_DATA/cyclone.master.conf, and slave uses $GOLDILOCKS_DATA/cyclone.slave.conf.
--silent | -i	It sets not to output messages.	-
--reset | -r	It resets operational information of the replication.	It is input in --reset TABLE_NAME or --reset all format. It should be described within a single quote (') when resetting multiple tables.
--group | -g	It sets a specific group.	It is input in --group GROUP_NAME format.
--help | -h	It outputs the help message.	
--sync | -n	It performs data synchronization.	It should be used together with --master | --slave.
--encrypt | -e	It encrypts the user password with the given key.	-
--key | -k	It sets the encryption key when performing the --encrypt option. If USER_ENCRYPT_PW is used in the config, it sets the decryption key.	-
--info | -o	It displays the status of the table which is currently being replicated.	It should be used together with --master, --group.
--recovery | -v	It is used when passing the replication being performed in a standalone mode of cluster environment to another cluster member.	It is used in --recovery GROUP_NAME form, and GROUP_NAME describes CYCLONE GROUP_NAME of a slave which was previously performed.

Option	Description	Remarks
--stand-alone | -S	It is operated in a standalone mode of cluster environment. (It is operated in a cluster mode in cluster environment.)	It is valid only in master. (It is set in the configuration file in slave.)
--local | -a	It is used when synchronizing the table sharded in cluster environment. It synchronizes only the data in the corresponding cluster group.	It should be used together with --sync.

- It executes all groups in master mode by using the default environment file.

```
prompt> cyclone --master --start
```

- It terminates all groups in master mode.

```
prompt> cyclone --master --stop
```

- It executes all groups in slave mode by using the default environment file.

```
prompt> cyclone --slave --start
```

- It terminates all groups in slave mode.

```
prompt> cyclone --slave --stop
```

- It executes only the TEST_GROUP group in master mode.

```
prompt> cyclone --master --start --group TEST_GROUP
```

- It terminates only TEST_GROUP group among the groups operated in master mode.

```
prompt> cyclone --master --stop --group TEST_GROUP
```

- It sets the TEST_CONFIG file in slave mode and executes it.

```
prompt> cyclone --slave --start --conf TEST_CONFIG
```

- It encrypts GOLDILOCKS user password.

```
prompt> cyclone --encrypt test --key 1234
Cyclone Encrypted Passwd : 'YFH+bpBPNvk='
```

- USER_ENCRYPT_PW is set in the config.

```
prompt> cyclone --master --start --key 1234
```

- It deletes the existing replications of master and slave, and newly starts it.

```
prompt> cyclone --master --start --reset all
prompt> cyclone --slave --start --reset all
```

- It deletes the existing replications of only the tables T1, T2 in master, and starts it newly from the curr ent point.

```
prompt> cyclone --master --start --reset 'T1, T2'
prompt> cyclone --slave --start
```

- It displays the current replication status per each table on master side.

```
prompt> cyclone --master --info --group GROUP1
================================================
 GROUP NAME = GROUP1
================================================
 SCHEMA NAME         : PUBLIC
 TABLE NAME          : T1 (GIVE-UP (DDL-LSN:129541))
 PHYSICAL ID         : 5299989643264
================================================
 SCHEMA NAME         : PUBLIC
 TABLE NAME          : T2 (ACTIVE (CAPTURE-START-LSN:128922))
 PHYSICAL ID         : 35549444308992
================================================
 SCHEMA NAME         : PUBLIC
 TABLE NAME          : T3 (ACTIVE (CAPTURE-START-LSN:128922))
 PHYSICAL ID         : 35558034243584
================================================
 SCHEMA NAME         : PUBLIC
 TABLE NAME          : T4 (ACTIVE (CAPTURE-START-LSN:128922))
 PHYSICAL ID         : 35566624178176
================================================
   TOTAL   COUNT : 4
   GIVE-UP COUNT : 1
================================================
```

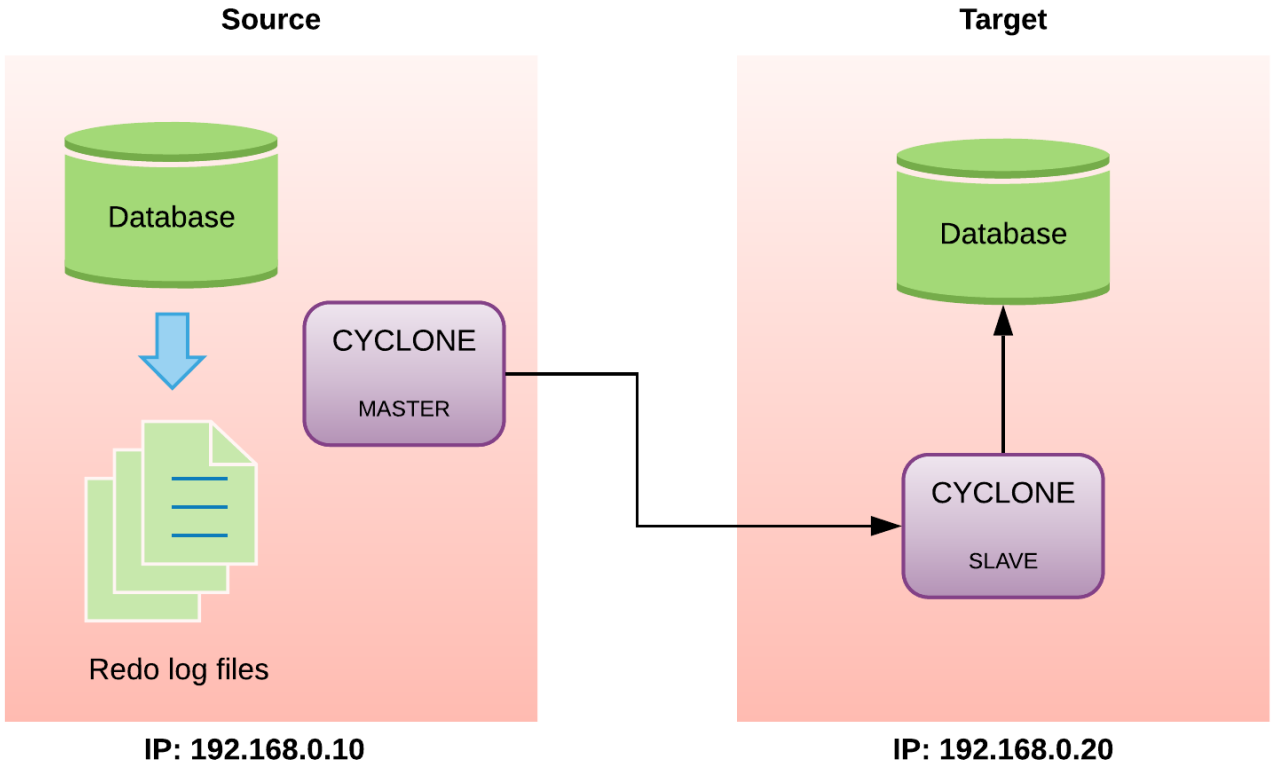
For more information about examples of adding/deleting node or group, refer to **Adding and D eleting Nodes**.

> For more information about examples of initializing replication, refer to **Initializing Replication**.

51.5 Operating Examples

The structure of operating CYCLONE is as follows.

Figure 1 The structure of operating CYCLONE



- Device structure
  - The source device which is the original data
    - GOLDILOCKS IP: 192.168.0.10
    - GOLDILOCKS port: 22581
    - The table to be replicated: T1, T2
  - The remote target device for replication
    - GOLDILOCKS IP: 192.168.0.20
    - GOLDILOCKS port: 22581

Operating Order

1. Set the original GOLDILOCKS environment.
2. Set the remote GOLDILOCKS environment.
3. Set the CYCLONE MASTER environment.
4. Set the CYCLONE SLAVE environment.

5. Execute and operate it.

Original GOLDILOCKS Configuration

Execute all described **Requirements**.
Create tables T1, T2 for testing.

```
gSQL > CREATE TABLE T1( COL1 INTEGER PRIMARY KEY, COL2 VARCHAR(20) );
gSQL > CREATE TABLE T2( COL1 INTEGER PRIMARY KEY, COL2 VARCHAR(20) );
gSQL > COMMIT;
```

Remote GOLDILOCKS Configuration

Execute **Registering User and Setting Privileges**.

CYCLONE MASTER Configuration

- CYCLONE MASTER configuration file
  - Storing path: $GOLDILOCKS_DATA/conf/cyclone.master.conf
  - It is the source device.

- It is connected with D/A of the same device, so HOST information does not exist.

```
USER_ID = cdc_user
USER_PW = cdc_password
GROUP_NAME = GROUP1
{
    PORT = 21102
    CAPTURE_TABLE =
    (
        T1,
        T2
    )
}
```

CYCLONE SLAVE Configuration

- CYCLONE SLAVE configuration file
  - Storing path: $GOLDILOCKS_DATA/conf/cyclone.slave.conf
  - It is the target device.

- It is connected with D/A of the same device, but HOST_IP and HOST_PORT are used for SYNC.

```
USER_ID = cdc_user
USER_PW = cdc_password
HOST_IP = 192.168.0.20
HOST_PORT = 22581
GROUP_NAME = GROUP1
{
    PORT = 21102
    APPLY_TABLE =
    (
        T1 TO T1,
        T2 TO T2
    )
}
```

Executing and Operating

- Execute CYCLONE master.
  - It should be executed in the source device.

```
cyclone --master --start --conf $GOLDILOCKS_DATA/conf/cyclone.master.conf
```

- Execute CYCLONE SLAVE.
  - It should be executed in the target device.

```
cyclone --slave --start --conf $GOLDILOCKS_DATA/conf/cyclone.slave.conf
```

Synchronizing Data

- Synchronizing data
  - It copies all data from the master to the slave, and it performs the replication.
  - It copies the table data in the master to be replicated to the table in the slave when performing d ata synchronization.
- It should be performed after giving --sync option to both master and slave when starting the operatio n.
  - *--sync all* is used to sync all tables, and *--sync TABLE_NAME* is used to sync a specific table.
    - The table name of master is described in TABLE_NAME. (The table name of master should be described in TABLE_NAME even when it is operated in slave.)
- HOST_IP (or HOST_EXTERNAL_IP) and HOST_PORT should be set in CONFIG or odbc.ini of slave thou gh it is D/A.

- If there is data in the slave table participating in the replication, data synchronization may n ot be properly performed due to duplicated PK. Therefore, the user should manually delete the data from the slave table before data synchronization.
- When data synchronization is performed by using the --sync option, the previous informatio n of replication is deleted. In other words, the --reset option is internally forced to be set wh en using the --sync option.
- Cyclone which is operated as master directly connects to GOLDILOCKS which is operated as slave and it performs the data synchronization.

If the target DB of the slave is Oracle or MySQL, then SYNC feature is not operable.

- Execute CYCLONE master
  - It should be executed in the source device.

```
cyclone --master --start --conf $GOLDILOCKS_DATA/conf/cyclone.master.conf --sync all
```

- Execute CYCLONE SLAVE
  - It should be executed in the target device.

```
cyclone --slave --start --conf $GOLDILOCKS_DATA/conf/cyclone.slave.conf --sync --all
```

- The following is an example of a sync error.
  - It is recommended to delete the data from tables od slave to prevent an error.

  - Execute MASTER.

```
gSQL> CREATE TABLE T1( COL1 INTEGER PRIMARY KEY, COL2 VARCHAR(20) );
gSQL> INSERT INTO T1 VALUES( 1, 'HELLO' );
```

- Execute SLAVE.

```
gSQL> CREATE TABLE T1( COL1 INTEGER PRIMARY KEY, COL2 VARCHAR(20) );
gSQL> INSERT INTO T1 VALUES( 1, 'BYE' );
```

- Execute MASTER.

```
cyclone --master --start --sync all
```

- Execute SLAVE.

```
cyclone --slave  --start --sync all
SLAVE cyclone log
[RECEIVER] start the data synchronization.
[RECEIVER] Error Occurred.
STOP Done.
```

An error occurs and the sync does not operate due to an attempt to using the same value in the primary key.

Initializing Replication Information

- Initializing replication information
  - It restarts the replication from current point for the entire table or a specific table participating in the replication.
  - It is performed by using the --reset TABLE_NAME option or the --reset ALL option.
- Differences of between master and slave when performing the --reset option
  - Master stores table information and column information participating in the replication as meta. If RESET is performed in master by using the --Reset ALL option or the Reset Table_name option, it removes the meta in the table and configures meta again by using the information at the current point.
    - The --reset option should be performed in master to replicate again the table which is given up during the replication performing DDL.
  - The slave stores the information for recovery when restating the replication. *--reset All* or *--reset TABLE_NAME* should be described when executing slave as same as when executing master.

- Add the --reset ALL option to initialize all tables participating in the replication when perfor ming master/ slave.
  prompt> cyclone --master --start --reset ALL
  prompt> cyclone --slave --start --reset ALL

- Add the --reset TABLE_NAME option when performing master to initialize the replication in formation of a specific table.
  - The table name of when performing slave should be as same as that of when performin g master.

  prompt> cyclone --master --start --reset T1
  prompt> cyclone --slave --start

- A single quote (') should be used to initialize two or more tables.
  prompt> cyclone --master --start --reset 'T1 T2'
  prompt> cyclone --master --start --reset 'T1, T2'

Recovery

- When using recovery,
  - It is used when handing over the replication to another cluster member in the same group and pr oceeding it, due to an error of master while the replication is being processed by using a standalo ne option in cluster environment.
  - It is performed with --recovery GROUP_NAME in master, and GROUP_NAME describes the group name of slave which previously performed the replication.

The process of resetting the replication information

1. Replicate with G1N1 member (standalone) in cluster environment.
prompt> cyclone --master --start --reset ALL --group g1n1_master --stand-alone
prompt> cyclone --slave --start --reset ALL --group g1n1_slave

2. Stop operating the existing cyclone to stop the cluster member or to hand over the service.
prompt> cyclone --master --stop
prompt> cyclone --slave --stop

3. Service the new replication (G1N2) by using the information of the existing replication. (G1N 1)

```
prompt> cyclone --master --start --reset all --group g1n2_master --stand-alone --recovery g1n1_
slave
prompt> cyclone --master --start --reset all --group g1n2_slave
```

51.6 Operating CYCLONE in Cluster

Requirements

- It provides a method to replicate members in the same cluster group in cluster environment.
  - Master and slave are operated in a relationship of 1 : 1 in standalone environment.
  - Master and slave are operated in a relationship of N: 1 in cluster environment.
- Available environment
  - Master should be cluster environment. (It is automatically detected.)
  - Slave can be either standalone or cluster environment.
- The status of master connected to slave is called as a node.

1. Master connecting to a slave should be a member in the same cluster group.
   - If it is a member of another cluster group, it should be operated in another slave.
2. If master is cluster environment, CYCLONE is basically operated in cluster mode, and --stand-alone option should be used to operate CYCLONE in standalone mode.
   - --stand-alone option is used only when performing master.
   - If the cluster information of master is described by adding CLUSTER option to configuration file, then slave is operated in cluster mode.

Operating

Starting and Terminating

- The time of terminating master and slave are different each in standalone environment and in cluster environment.
  - Standalone environment
    - If master is terminated, then slave is automatically terminated.
    - If slave is terminated, master is switched to an idle state but it is not terminated.
  - Cluster environment
    - Even when master is terminated, slave is not terminated. If another master is already connected, then the replication proceeds. If any master is not connected, slave is switched to an idle state.
    - If slave is terminated, all connected masters are automatically terminated.

If master is running in cluster environment, then slave should use CLUSTER_LIST option.
If standalone option is used to run master in cluster environment, then slave should not use CLU STER_LIST option.

Node Type

In cluster environment, CYCLONE classifies master connecting to slave into a trust node and a non-trust n ode.

Table 51-9 Node in cluster environment

Node	Description	Replication
Trust node	It is a node all of whose tables participating in replication are online.	O
Non-trust node	It is a node one or more of whose tables participating in replication are offli ne.	X

Operation According to Node Status

- Switching from a non-trust node to a trust node
  - It is executed when all tables are switched to online by rebalancing the cluster member while an offline table exists.
  - The node is automatically switched from a non-trust node to a trust node, and participates in repl ication.
- Switching from a trust node to a non-trust node
  - It is executed when a table being replicated in another cluster member in operation is altered whi le the cluster member is terminated and is not operated.
  - A non-trust node is automatically altered when a cluster member joins, and it is excluded from re plication.

- The replication is maintained when one or more trust nodes exist among masters participati ng in the replication.
- A trust node is not restricted to a specific node, and any node is allowed regardless of node types.
- If all masters participating in replication becomes non-trust nodes, the replication does not proceed any more, and the replication does not start again even when they become trust n odes.
- At least one or more trust nodes are required to proceeds the replication in cluster environ ment.

Executing

Configuration File

The configuration file of master is as same as that of standalone.
The following is an example of configuring G1N1 master environment, and IP of that device is assumed a s 192.168.0.10.

- There is nothing to alter for master even in cluster environment.

```
USER_ID = cdc_user
USER_PW = cdc_password
GROUP_NAME = G1N1_Master
{
    PORT = 21011
    CAPTURE_TABLE =
    (
        T1,
        T2
    )
}
```

The following is an example of configuring G1N2 master environment, and IP of that device is assumed a s 192.168.0.20.

- There is nothing to alter for master even in cluster environment.

```
USER_ID = cdc_user
USER_PW = cdc_password
GROUP_NAME = G1N2_Master
{
    PORT = 21012
    CAPTURE_TABLE =
    (
        T1,
        T2
    )
}
```

CLUSTER is added to the configuration file of slave to describe the information of master.

- CLUSTER consists of ID, MASTER_IP, and PORT. (The sequence does not matter.)
  - ID is input in numbers, and it is used as an internal delimiter to distinguish master in slave. Once i t is set, it can not be altered. (If it is altered, it is recognized as a different master.)
  - MASTER_IP is IP in which CYCLONE MASTER is operated in CLUSTER member.
  - PORT is a PORT in which CYCLONE MASTER is operated in CLUSTER member.
- The information of a CLUSTER is described in parentheses, and it is distinguished by a comma (,) bet ween parentheses.

MASTER_IP, PORT which are the master information of CLUSTER are described in CLUSTER configuration.

```
USER_ID = cdc_user
USER_PW = cdc_password
GROUP_NAME = G1_Slave
{
    CLUSTER = ( ID = 1, MASTER_IP = 192.168.0.10, PORT = 21011 ),
              ( ID = 2, MASTER_IP = 192.168.0.20, PORT = 21012 )
    APPLY_TABLE =
    (
        T1 TO T1,
        T2 TO T2
    )
}
```

Executing Method

Executing in Master

- It is executed in G1N1.

```
cyclone --master --start --conf $GOLDILOCKS_DATA/conf/cyclone.master.conf --group G1N1_Master
```

- It is executed in G1N2.

```
cyclone --master --start --conf $GOLDILOCKS_DATA/conf/cyclone.master.conf --group G1N2_Master
```

Executing in Slave

```
cyclone --slave --start --conf $GOLDILOCKS_DATA/conf/cyclone.slave.conf --group G1_Slave
```

SYNC Method

Executing Master

It is executed in G1N1. (It is master of which SYNC is to be executed, and it uses --sync option.)

```
cyclone --master --start --conf $GOLDILOCKS_DATA/conf/cyclone.master.conf --sync all
```

Executing Slave

```
cyclone --slave --start --conf $GOLDILOCKS_DATA/conf/cyclone.slave.conf --sync all
```

--local option should be used when syncing if a sharded table participates in the replication. It is because the data of a sharded table exists only in each cluster group, and data in other cluster groups are synchronized when executing sync without --local option. However, if all tables participating in the replication in cluster environment are cloned tables, then --local option does not need to be used.

Others

--reset option can be used as same as it is used in standalone environment.

51.7 Monitoring (CYMON)

CYMON (CYclone MONitor) is a tool for monitoring CYCLONE and CLUSTONE which are CDC replication tools. It periodically updates the monitoring information of CDC replication to GOLDILOCKS operated as master.

Table 51-10 Execution files

File name	Description
cymon	Monitoring cyclone and clustone

Configuration File

The configuration file of cymon uses the configuration file used when running cyclone as master. If a specific configuration file is not set by using the --conf option, the $GOLDILOCKS_DATA/conf/cyclone.master.conf file is read as the default when cyclone is run as master.

CYMON should be run on the device which is as same as the device of which CYCLONE runs as master.

Monitoring Contents

CYMON

CYMON periodically updates the operating information of CYCLONE to the CYCLONE_MONITOR_INFO table of GOLDILOCKS. The monitoring information is as follows.

Table 51-11 CYCLONE_MONITOR_INFO

Column	Description
GROUP_NAME	It is the name of a group in which CYCLONE was executed.
TIME	It is the information of time at which the information was updated. (YYYY-MM-DD HH24:MI:SS)
MASTER_STATE	It is the state of CYCLONE which is operated as MASTER. • N/A: The state is unknown. • READY: It is waiting for connection of SLAVE • RUNNING: SLAVE is connected and the replication is running.

Column	Description
SLAVE_STATE	It is the state of CYCLONE which is operated as SLAVE. • N/A: The state is unknown. • RUNNING: The replication is running.
MASTER_PORT	It is the information of PORT on which CYCLONE operated in MASTER is waiting for slave.
SLAVE_IP	It is an IP address of the device connected with CYCLONE SLAVE.
REDO_LOG_FILESEQ	It is the sequence number of the redo log file of the running GOLDILOCKS.
REDO_LOG_BLOCKSEQ	It is the block sequence number of the redo log file of the running GOLDILOCKS.
CAPTURE_FILESEQ	It is the sequence number of the redo log file being captured by CYCLONE master.
CAPTURE_BLOCKSEQ	It is the block sequence number of the redo log file being captured by CYCLONE master.
APPLY_FILESEQ	It is the sequence number of the redo log file being processed by CYCLONE slave.
APPLY_BLOCKSEQ	It is the block sequence number of the redo log file being processed by CYCLONE slave.
CAPTURE_INTERVAL	It is the number of the remaining redo log blocks to be processed by CYCLONE master • 1 block = 512 bytes (Redo log block size = 512 bytes)
CAPTURE_INTERVAL_SIZE	It is the size of the remaining redo log files to be processed by CYCLONE master.
TOTAL_TX_COUNT	It is the number of all transactions captured by CYCLONE master.
CAPTURE_TX_COUNT	It is the number of all transactions in which the replication targets are included among the transactions captured by CYCLONE master.
CAPTURE_COMMIT_LSN	It is the commit log number of the last transaction which was captured in CYCLONE master. It is not updated if there is not a transaction to be captured any more.
APPLY_COMMIT_LSN	It is the commit log number of the transaction being processed in CYCLONE slave. It is not updated if there is not a transaction to be processed any more.

INTERVAL information is the CAPTURE information for analyzing the redo log file of CYCLONE operated as master, and it is not the information reflected by APPLIER in CYCLONE SLAVE.

Executing and Monitoring

Executing Option

Table 51-12 Executing option

Option	Description	Remarks
--conf | -c	It sets the configuration file path.	It is input in --conf CONFIG FILE format. • It should set the file as same as the configuration

Option	Description	Remarks
		file used when running CYCLONE as master.
--start | -s	It executes CYMON.	-
--stop | -t	It terminates CYMON.	-
--status | -u	It displays the operating status of CY MON.	-
--cycle | -y	It sets the update cycle of monitoring i nformation.	It is input in --cycle X format. • It should be entered in seconds.
--key | -k	It sets the decryption key when USER_ ENCRYPT_PW is used in config.	-
--trace | -r	It simultaneously records the monitori ng information of capture on the trac e log.	The trace log is recorded in $GOLDILOCKS_DATA/trc/c ymon.trc.
--silent | -i	It sets not to output the message.	-
--help | -h	It outputs the help message.	-

Examples

- Execute CYMON (Update the monitoring information of cyclone_monitor_info table.)
  - Executes CYMON to update the monitoring information per second. It continuously updates the i nformation until terminating CYMON.

```
cymon --start --cycle 1
```

- View the monitoring information

```
gSQL> \set vertical on
gSQL> select * from cyclone_monitor_info;
               GROUP_NAME # GROUP1
                     TIME # 2015-01-13 17:34:53
             MASTER_STATE # READY
              SLAVE_STATE # N/A
              MASTER_PORT # 21102
                 SLAVE_IP # null
         REDO_LOG_FILESEQ # 0
        REDO_LOG_BLOCKSEQ # 52392
          CAPTURE_FILESEQ # 0
         CAPTURE_BLOCKSEQ # 0
            APPLY_FILESEQ # 0
           APPLY_BLOCKSEQ # 0
         CAPTURE_INTERVAL # 0
```

```
   CAPTURE_INTERVAL_SIZE # 0
      CAPTURE_COMMIT_LSN # 0
        APPLY_COMMIT_LSN # 0
```

    - The information above describes that only CYCLONE MASTER is being operated and SLAVE is wa iting.

```
gSQL> \set vertical on
gSQL> select * from cyclone_monitor_info;
              GROUP_NAME # GROUP1
                    TIME # 2015-01-13 17:36:17
            MASTER_STATE # RUNNING
             SLAVE_STATE # RUNNING
             MASTER_PORT # 21102
                SLAVE_IP # 127.0.0.1
        REDO_LOG_FILESEQ # 0
       REDO_LOG_BLOCKSEQ # 52811
         CAPTURE_FILESEQ # 0
        CAPTURE_BLOCKSEQ # 52811
           APPLY_FILESEQ # 0
          APPLY_BLOCKSEQ # 52811
        CAPTURE_INTERVAL # 0
   CAPTURE_INTERVAL_SIZE # 0
      CAPTURE_COMMIT_LSN # 1023
        APPLY_COMMIT_LSN # 1023
```

    - The information above describes that CYCLONE MASTER and SLAVE are being operated.

- Execute CYMON (Store the monitoring information in trace log)
    - The monitoring information is stored not only in amonitoring table but also in a trace log since w hen CYMON is normally executed with master.
    - Generally, it is used when unable to view cyclone_monitor_info table due to an error of the mast er DB. In this case, the information of master DB such as information of REDO_LOG_FILESEQ and REDO_LOG_BLOCKSEQ are not included.
    - Trace log is in $GOLDILOCKS_DATA/trc/cymon.trc.

```
cymon --start --trace
```

- View the monitoring information (When using --trace option)
    - It stores the monitoring information since when the cyclone master is normally executed.

```
GROUP_NAME         TIME                MASTER_STATE SLAVE_STATE  MASTER_PORT  SLAVE_IP
CAPTURE_FILESEQ CAPTURE_BLOCKSEQ TOTAL_TX_COUNT CAPTURE_TX_COUNT  CAPTURE_COMMIT_LSN
APPLY_FILESEQ   APPLY_BLOCKSEQ   APPLY_COMMIT_LSN
------------------ ------------------- ------------ ------------ ----------- ---------------
--------------- ----------------- -------------- ----------------- -----------------
-------------- ----------------- ------------------
GROUP1             2016-11-02 15:43:03 READY        N/A               21102 null
0               0              0               0               0                  0
0                  0
GROUP2             2016-11-02 15:43:03 READY        N/A               21103 null
0               0              0               0               0                  0
0                  0
```

- The information above describes that only CYCLONE MASTER is being operated and SLAVE is waiting.

```
GROUP_NAME         TIME                MASTER_STATE SLAVE_STATE  MASTER_PORT  SLAVE_IP
CAPTURE_FILESEQ CAPTURE_BLOCKSEQ TOTAL_TX_COUNT CAPTURE_TX_COUNT  CAPTURE_COMMIT_LSN
APPLY_FILESEQ   APPLY_BLOCKSEQ   APPLY_COMMIT_LSN
------------------ ------------------- ------------ ------------ ----------- ---------------
--------------- ----------------- -------------- ----------------- -----------------
-------------- ----------------- ------------------
GROUP1             2016-11-02 15:43:11 RUNNING      RUNNING           21102 192.168.0.206
7            52779              0               0               15346              7
52779             15346
GROUP2             2016-11-02 15:43:11 READY        N/A               21103 null
0               0              0               0                  0              0
0                  0
```

- The information above describes that CYCLONE MASTER (group1) and SLAVE are being operated.

- View the operating status of CYMON

```
$ cymon --status
=====================================
|             CYMON STATUS          |
=====================================
|       Service is running...       |
-------------------------------------
```

- Terminate CYMON

```
$ cymon --stop
stop done.
```

52.

CLUSTONE

52.1 CLUSTONE

CLUSTONE is a replication tool which uses the Change Data Capture (CDC) method.

Overview

Database records the data changes which occur during the operation in the redo log file for the recovery. CDC performs the replication by analyzing the information of the recorded redo log file.

CLUSTONE is driven being divided into master and slave. Master recognizes the changes of the redo log file in the original database and analyzes it, then transfers it to the slave. Slave analyzes the received data and performs the replication by using ODBC.

Differences between CLUSTONE and CYCLONE

Both CLUSTONE and CYCLONE are replication tools which use the Change Data Capture (CDC) method, but there are differences as follows.

Table 52-1 Differences between CLUSTONE and CYCLONE

Item	CLUSTONE	CYCLONE
Operating master : slave	It is operated 1 : N in standalone environment, and it is operated N : N in cluster environment.	It is operated 1 : 1 in standalone environment, and it is operated N : 1 in cluster environment.
Replication information	It creates TxData file by processing the redo log file, then extracts the replication data from this file.	It extracts the replication data by directly reading the redo log file.
Whether to support DDL	It does not support DDL replication.	It partially supports DDL replication based on the procedure.
Structure	Multi-process structure	Multi-thread structure
Interwork with other vendor DB	It is not supported.	It is supported.
SYNC feature	It is not supported.	It is supported.

- The biggest difference between CLUSTONE and CYCLONE is the number of targets to replicate.
  - If CYCLONE replicates two targets, then two masters and two slaves should be operated.
  - However, if CLUSTONE replicates two targets, then only one master and two slaves should be operated.

- CLUSTONE creates and stores TxData file by processing the redo log file, so it requires an extra space in HDD.
- If CYCLONE replicates multiple targets, then the performance of the original database may be degrad ed because redo log file I/O increases during the capture operation.
- Recommended environment for CYCLONE and CLUSTONE
  - If replicating in standalone environment or replicating between clusters, then it is recommended to use cyclone.
  - If replicating two or more cluster, then it is recommended to use clustone.

Figure 1 The structure of operating CYCLONE (master : slave = 1 : 2)

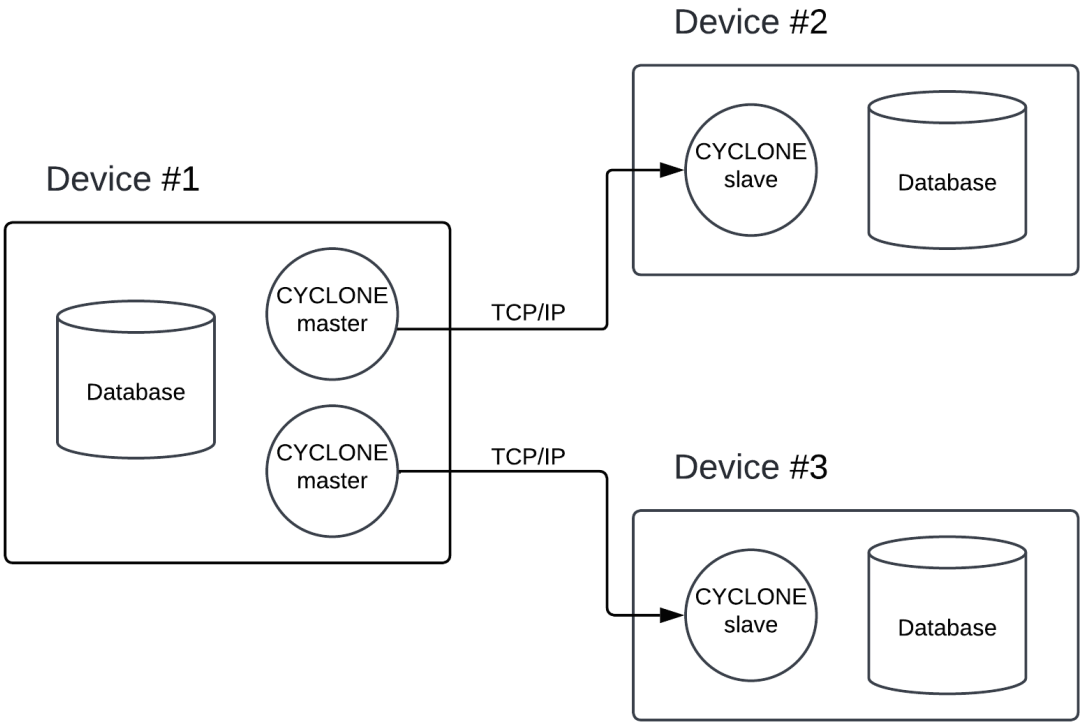
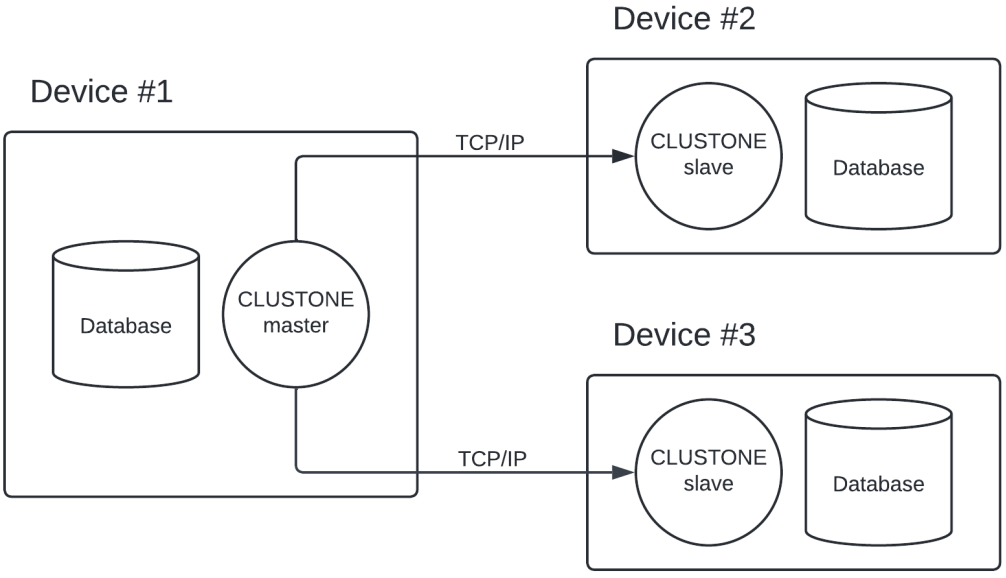


Figure 2 The structure of operating CLUSTONE (master : slave = 1 : 2)



Operational Features

- It replicates GOLDILOCKS which is operated in standalone or cluster.
- It is divided into master and slave and is operated as master/slave in group unit.
  - It can replicate one master and N slaves, so it can be used to replicate clusters. (master : slave = 1 : N)
  - The master which is operated in N members of the same cluster group can be connected to one s lave. (master : slave = N : 1)
- TxDataFile is created and stored during operating clustone.
  - This file can execute/ terminate the transaction which replicates the redo log file in group unit.
  - Replicating transaction is executed in table unit, and a group may include one or more tables.
- Master and slave of clustone uses TCP/ IP communication.
- A single table may be operated being included in several groups.
- The original database should have the redo log files, so DATA_STORE_MODE should be operated in T DS.
- The original database should have SUPPLEMENTAL LOGGING.
  - SUPPLEMENTAL LOGGING adds an additional information to the redo log files for the replication of clustone.
- The original database should be operated in ARCHIVE LOG mode.
  - GOLDILOCKS recursively reuses the redo log files. When the redo log files are reused before the r eplication is completed, the replication becomes aborted and the existing replication from the cu rrent point is canceled then restarted. Therefore, the redo log files should be operated in ARCHIV E LOG mode to be archived.
- It does not affect GOLDILOCKS even when it fails to replicate because it is operated as an independe nt process.

Operational Restrictions

- The table participated in the replication should have a PRIMARY KEY.
- Only the committed transaction is allowed to be replicated. Therefore, the content is unknown to the slave before committing the transaction.
- The primary key update is not supported.
  - When the primary key is updated, the table is given up and it is not replicated any more.
- The table participated in the replication can not use the column which has Generated Always As Iden tity property.
- When Data Definition Language (DDL) is performed on the table in which the replication is being ope rated, it could be given up.
  - It is same in case of the truncated table.
- The given-up table can be reset only when it was given up with --reset TABLE_NAME
- The columns configuring the table participating in the replication should have the same structure.(da

ta type, order)
- The database performing the replication should have the same character encoding.
- The table stored in the recycle bin is not a replication target.
- When replicating one slave and N masters, the environment settings for the masters should be same.
  - e.g. capture table list

TxData File

- If clustone is operated as master, then it reads the redo log file, filters the information related to the r eplication only, reforms it to the appropriate form for the replication, then stores it as a file. This file i s called as TxData file.
  - TxData file is stored in $GOLDILOCKS_DATA/data.
  - The file name format is clustone.GROUP_NAME_FILENO.dat, and the extension is dat.
  - The file should be periodically deleted and managed. (It is manually deleted and managed so far, but the option for automatic deletion/management will be provided soon.)
- TxData file is stored together with the control file which manages the storage information.
  - The file name format of the control file is clustone.GROUP_NAME_FILENO.ctl.
  - The control file is stored together with the backup file whose extension is ctl_0 in case for losing the original data.
- The user is not allowed to modify TxData file and the control file.

52.2 Requirements

Original GOLDILOCKS: It is required to perform GOLDILOCKS preparations, user registration and privilege setting all.
Remote GOLDILOCKS: It is required to perform user registration and privilege setting only.

GOLDILOCKS Requirements

The followings should be set in GOLDILOCKS before starting the replication using CLUSTONE.

SUPPLEMENTAL LOGGING

SUPPLEMENTAL LOGGING stores additional information in the redo log file for the replication of CLUSTONE. The database restart is required to change the settings of the database in operation, but the database restart is not required to set SUPPLEMENTAL LOGGING for a particular table.

Setting SUPPLEMENTAL LOGGING in Database

- If the GOLDILOCKS property is set as SUPPLEMENTAL LOGGING, SUPPLEMENTAL LOGGING is recorded for every table.
- GOLDILOCKS restart is required.
- The appropriate information is added or updated in the property file.
  - Property file: goldilocks.properties.conf
  - Property setting: SUPPLEMENTAL LOG_DATA_PRIMARY_KEY = YES

Setting SUPPLEMENTAL LOGGING in Specific Table Participating in Replication

```
<add table supplemental log statement> ::=
    ALTER TABLE table_name
        ADD SUPPLEMENTAL LOG DATA ( PRIMARY KEY ) COLUMNS
    ;
```

ARCHIVE LOG

GOLDILOCKS reuses the redo log files recursively. When GOLDILOCKS reuses the redo log file being processed by CLUSTONE, then CLUSTONE does not proceed and is terminated. GOLDILOCKS should be operated in ARCHIVE LOG mode to ensure the continuous replication operation.

Changing Database in Operation to ARCHIVE LOG Mode

- Restart GOLDILOCKS.
- After database is shutdown, connect with sysdba and change it to ARCHIVE LOG mode in MOUNT p hase.

```
gSQL> \startup mount
Startup success
gSQL> alter database archivelog;
Database altered.
```

Setting ARCHIVE LOG Mode When Creating Database

- Update the property file before creating the database.
  - Property file: goldilocks.properties.conf
  - Property setting: ARCHIVELOG_MODE = 1

The path in which ARCHIVE LOG file is stored can be viewed and updated with 'ARCHIVELOG_ DIR'.

DATA_STORE_MODE

CLUSTONE performs the replication by reading the redo log files of GOLDILOCKS. Therefore, GOLDILOCK S should be operated in Transactional Data Store (TDS) mode.

Changing DATA_STORE_MODE

- Restart the database.
- Add or update the corresponding information in the property file.
  - Property file: goldilocks.properties.conf
  - Property setting: DATA_STORE_MODE = 2

If the value of DATA_STORE_MODE is 1, it indicates Concurrent Data Store (CDS), and if it is 2, i t indicates Transactional Data Store (TDS).

Registering User and Setting Privileges

CLUSTONE retrieves and manipulates the required information during the operation. The user operating CLUSTONE and the proper privileges for the user are required.

Creating Database User

A specific user should be added to operate CLUSTONE, and the corresponding user should be added for all GOLDILOCKS of which CLUSTONE is operated in master, slave mode.

```
<user definition> ::=
    CREATE USER user_identifier IDENTIFIED BY password
    [ DEFAULT TABLESPACE tablespace_name ]
    [ TEMPORARY TABLESPACE tablespace_name ]
    [ INDEX TABLESPACE {tablespace_name¦NULL} ]
    [ <schema clause> ]
    ;
<schema clause> ::=
      WITH SCHEMA [schema_name]
    ¦ WITHOUT SCHEMA
```

The following is an example of creating the user cdc_user with the password cdc_password.

```
gSQL> CREATE USER cdc_user IDENTIFIED BY cdc_password;
```

Database Privileges

Granting User Access Privilege

The following is an example for granting the access privilege to cdc_user. It should be set on all GOLDILOCKS which is operated in both master/ slave mode.

```
gSQL> GRANT CREATE SESSION ON DATABASE TO cdc_user;
```

Granting Privilege for Altering Table

The following is an example of granting the table updating privilege to cdc_user. It is set on GOLDILOCKS which is operated in slave mode.

```
gSQL> GRANT INSERT ANY TABLE, DELETE ANY TABLE, UPDATE ANY TABLE ON
DATABASE TO cdc_user;
```

Tablespace Privileges

The privileges on the data tablespace and temporary tablespace should be set. The following is an exampl e of granting the privilege for using the default tablespace to cdc_user. It is set on GOLDILOCKS which is operated in slave mode.

```
gSQL> GRANT CREATE OBJECT ON TABLESPACE mem_data_tbs TO cdc_user;
gSQL> GRANT CREATE OBJECT ON TABLESPACE mem_temp_tbs TO cdc_user;
```

Schema Privileges

The schema privileges for creating and managing meta managed in CLUSTONE should be set. The followi ng is an example granting the schema privilege to cdc_user. It is set on GOLDILOCKS which is operated in slave mode.

```
gSQL> GRANT CREATE TABLE, CREATE INDEX, CREATE SEQUENCE, CREATE VIEW,
ADD CONSTRAINT ON SCHEMA cdc_user TO cdc_user;
```

52.3 Configuration

Configuration File

When performing CLUSTONE, the information and options required for operating are set by using the configuration file.

- When a specific configuration file is not set by using the --conf option, a specific file in the $GOLDILOCKS_DATA/conf directory is read. clustone.master.conf file is read when it is operated in master mode and clustone.slave.conf file is read when it is operated in slave mode.

Table 52-2 Configuration file options

Name	Description	Coverage
COMM_CHUNK_COUNT	It sets the size of BUFFER for communication.	Master/ slave
DSN	It sets Data Source Name.	Master/ slave
GROUP_NAME	It sets the group name.	Master/ slave
HOST_IP	It sets the host IP address of which GOLDILOCKS operates.	Master/ slave
HOST_PORT	It sets the host port of which GOLDILOCKS operates.	Master/ slave
PORT	It sets the port for master/ slave communication.	Master/ slave
USER_ID	It sets the user name.	Master/ slave
USER_PW	It sets the user password.	Master/ slave
USER_ENCRYPT_PW	It sets the encrypted password for DB user.	Master/ slave
CAPTURE_TABLE	It sets the table to be replicated.	Master
LOG_PATH	It is used when interworking with LOGMIRROR and it sets the location of the redo log file.	Master
PROTOCOL	It sets the connection type which is to be connected to GOLDILOCKS. (DA or TCP)	Master/ slave
READ_LOG_BLOCK_COUNT	It sets the amount of data to be read at a time when operating CAPTURE.	Master
TRANS_SORT_AREA_SIZE	It sets the size of the BUFFER to be allocated to CAPTURE.	Master
TRANS_FILE_PATH	It sets the location in which the temporarily generated file is to be stored when operating CAPTURE.	Master
APPLIER_COUNT	It sets the number of APPLIER simultaneously performed during the replication.	Slave
APPLY_COMMIT_SIZE	It sets the maximum size for COMMIT during the replication.	Slave
APPLY_TABLE	It sets the table to which the replicated table is to be applied.	Slave

Name	Description	Coverage
MASTER_IP	It sets the IP address of the equipment of which CLUSTONE master is operated.	Slave
PROPAGATE_MODE	It sets whether to propagate the data applied by CLUSTONE.	Slave
UPDATE_APPLY_MODE	It distinguishes the operation when updating. (The default value is 0.) • 0: It is updated only when the primary keys are same. • 1: It is updated only when the primary key and the value before the update are same. • 2: It is updated only when the primary keys are same. It compares the value before and after the update, then leaves a log if the values are different.	Slave
TCP_NODELAY	It sets TCP_NODELAY option of a socket. (The default value is 1.) • 0: TCP_NODELAY off • 1: TCP_NODELAY on	Master
HEARTBEAT_TIMEOUT	It sets the maximum time (second) maintaining connection if the connection is not smooth due to network disconnection or system error after replication connection.	Master/ slave
SKIP_COMMENT	It enters the text for transaction skip. If the same text is entered as commit comment in master, then it skips that transaction instead of replicating it.	Master
LOG_CAPTURE_INTERVAL_1	It sets the execution cycle of capture. If the value is not changed after executing 10 times with that value, then it is converted to the value of LOG_CAPTURE_INTERVAL_2 and performs capture. (The default value is 0.2 seconds.)	Master
LOG_CAPTURE_INTERVAL_2	It sets the execution cycle of capture. If the value is not changed after executing with the value of LOG_CAPTURE_INTERVAL_1, then it sets the execution cycle of capture. (The default value is 1 second.)	Master
CLUSTER	It specifies the connection information of a master when the master is in cluster environment. A master consists of the following three information. • ID: It is a delimiter and sets to 1 or more value. • MASTER_IP • PORT	Slave
TXDATA_FILE_SIZE	It specifies the maximum size of TXDATA file which was created by processing the redo log file.	Master
TXDATA_FILE_PATH	It sets the path in which the TxData file is to be stored.	Master
PACKET_COMPRESSION_MODE	It sets whether to compress the data of communication of master and slave. (1: Enable, 0: Disable, Default: Enable)	Master
SHARED_MEMORY_NAME	It specifies the name of the shared memory which is used during running the master. The name should be unique so that the redundant names do not exist within the clustone group in the same equipment.	Master
SHARED_MEMORY_KEY	It specifies the shared memory key which is used during running the master. It should be unique so that the redundant values do not exist within the clus	Master

Name	Description	Coverage
	tone group in the same equipment.	

Configuration Option

COMM_CHUNK_COUNT

- It sets the number of buffers(chunk) used in data communication between master and slave of CLUS TONE.
- It allocates the resource with 16M * N (the set value).
- The default value is 32, and the actual size is 16M * 32 = 512 M.
- The minimum value is 10.
- It can be set in master and slave.
  - If too small value is set so the buffer is not enough, the performance becomes poor.

- Settings applied to all groups

```
COMM_CHUNK_COUNT = 10
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    COMM_CHUNK_COUNT=20
    ....
    ....
}
```

DSN

- It sets the data source name which is required when connecting to GOLDILOCKS.
- It can be set in master and slave.
- The default value is GOLDILOCKS.

- Settings applied to all groups

```
DSN=GOLDILOCKS
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    DSN=GOLDILOCKS
    ....
    ....
}
```

GROUP_NAME

- It is necessary for distinguishing the CLUSTONE operation within the equipment and it is a unit of gen erating the operation process.
- It is the delimiter of when starting or terminating CLUSTONE in group unit.
- After it is set, it should not be changed. If changed, it is regarded as a new group.
- It can be set in master and slave.
  - The connection between master and slave is separated not by GROUP_NAME but by PORT.
  - GROUP_NAME should be unique in the same equipment.
- The braces { } should be used.

```
GROUP_NAME = testGROUP
{
    ....
    ....
}
```

HOST_IP

- It sets the IP address of GOLDILOCKS to be connected by CLUSTONE.
- It can be set in master and slave.
  - It is valid only when PROTOCOL is set to TCP.

- Settings applied to all groups

```
HOST_IP = 127.0.0.1
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    HOST_IP = 127.0.0.1
    ....
    ....
```

```
}
```

HOST_PORT

- It sets the port of GOLDILOCKS to be connected by CLUSTONE.
- It should be set together with HOST_IP.
- It can be set in master and slave.

- Settings applied to all groups

```
HOST_PORT = 22531
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    HOST_PORT = 22531
    ....
    ....
}
```

PORT

- It sets the PORT used in the communication between master and slave.
- The duplicated PORT should not be set between the groups, and the unique value should be used for each GROUP.
- It is mandatory be set and it can be set only within GROUP_NAME.

- It can be set only within a group.

```
GROUP_NAME = testGROUP
{
    PORT = 21102
    ....
    ....
}
```

USER_ID

- It sets the user ID required to access GOLDILOCKS.
- It can be set in master and slave.

- Settings applied to all groups

```
USER_ID = testID
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    USER_ID = testID
    ....
    ....
}
```

USER_PW

- It sets the user password required to access GOLDILOCKS.
- It can be set in master and slave.

- Settings applied to all groups

```
USER_PW = testPW
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    USER_PW = testPW
    ....
    ....
}
```

USER_ENCRYPT_PW

- It sets the user password which is required for the access to GOLDILOCKS by encrypting.
- It is used instead of USER_PW.
- Encrypted user password is created by [clustone --encrypt *user password* --key *the key to be encrypte*

d].

- If this value is used, then --key option should be used when executing clustone. (In this case, the key value as same as that created with --encrypt should be used.

- Settings applied to all groups

```
USER_ENCRYPT_PW = 't33KImiqvhqNyfN+uZmFrw=='
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    USER_ENCRYPT_PW = 't33KImiqvhqNyfN+uZmFrw=='
    ....
    ....
}
```

CAPTURE_TABLE

- It sets the table to be replicated.
  - It is set in *schema name.table name* format.
- It can set only in master.
- It can be set only within a group.
- When specifying several tables, the parentheses ( ) should be used.
- The table stored in the recycle bin can not be set to be replicated.

```
GROUP_NAME = testGROUP
{
    CAPTURE_TABLE =
    (
        testSchema1.testTable1,
        testSchema1.testTable2,
        testSchema2.testTable1
    )
}
```

LOG_PATH

- It is used when interworking with LOG MIRROR.
- It sets the path of the redo log files stored by LOGMIRROR.
  - The absolute path should be used.

    - The path should be specified by using single quote (').

- Settings applied to all groups

```
LOG_PATH = '/data/wal/'
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    LOG_PATH = '/data/wal/'
    ....
    ....
}
```

PROTOCOL

- It sets the type to connect to GOLDILOCKS in operation.
- It can be set to DA or TCP.
- If PROTOCOL is set to DA, neither HOST_IP nor is HOST_PORT used when accessing to GOLDILOCKS.
- The default value is DA.

- Settings applied to all groups

```
PROTOCOL = DA
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    PROTOCOL = DA
    ....
    ....
}
```

READ_LOG_BLOCK_COUNT

- It sets the number of the log blocks to be read at a time when capturing redo log file.
- It can be set only in master.
- The size of a log block is 512 bytes.
- The default value is 40960, and the actual read size is 20 MBytes.

    - The minimum value is 100.

- Settings applied to all groups

```
READ_LOG_BLOCK_COUNT = 1024
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    READ_LOG_BLOCK_COUNT = 1024
    ....
    ....
}
```

TRANS_SORT_AREA_SIZE

- It sets the memory space required for capturing the redo log files.
- It can be set only in master.
- The unit is MB (megabytes).
- The default value is 500 MB.
    - The minimum value is 10 MB.
- If the value is set too small, the performance becomes poor.

- Settings applied to all groups

```
TRANS_SORT_AREA_SIZE = 300
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    TRANS_SORT_AREA_SIZE = 300
    ....
    ....
}
```

TRANS_FILE_PATH

- If a space bigger than TRANS_SORT_AREA_SIZE is required, the temporary file is created. It sets the p ath in which the temporary file is to be stored.

    - The absolute path should be used.
    - The path should be specified by using single quote (').
- It can be set only in master.

- Settings applied to all groups

```
TRANS_FILE_PATH = '/data/TmpTrans'
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    TRANS_FILE_PATH = '/data/TmpTrans'
    ....
    ....
}
```

APPLIER_COUNT

- It sets the number of APPLIER executing replication.
    - It indicates a parallel factor.
- The default value is 6, and 6 sessions are created.
    - The maximum value is not limited, but too high value can cause the contention between APPLIER s.
    - The set value significantly affects on performance.
    - The session is created according to the set value and the replication is simultaneously performed.
- It can be set only in slave.

- Settings applied to all groups

```
APPLIER_COUNT = 16
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    APPLIER_COUNT = 16
    ....
    ....
}
```

APPLY_COMMIT_COUNT

- It sets the number of transaction executing COMMIT when the replication is performed.
  - It executes COMMIT after performing the transaction once when replicating transactions commit ted in the original database to the remote database.
  - The set value indicates the maximum value.
- The set value affects on performance.
- It can be set only in slave.

- Settings applied to all groups

```
APPLY_COMMIT_COUNT = 1000
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    APPLY_COMMIT_COUNT = 1000
    ....
    ....
}
```

APPLY_TABLE

- It sets the table to which the replicated table is to be applied.
  - It is set in *replicating table name TO table name to which the replicated table is applied* format.
  - The table names are not necessary to be same.
- It can be set only in slave.
- It can be set only within a group.
- The parentheses ( ) should be used when specifying several tables.

```
GROUP_NAME = testGROUP
{
    APPLY_TABLE =
    (
        testSchema1.testTable1 TO testSchema1.testTable1,
        testSchema1.testTable2 TO testSchema2.testTable3,
        testSchema2.testTable3 TO testSchema2.testTable4
    )
}
```

MASTER_IP

- It sets the IP address of the device which CLUSTONE master operates.
- It can be set only in slave.

- Settings applied to all groups

```
MASTER_IP = 192.168.0.100
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    MASTER_IP = 192.168.0.100
    ....
    ....
}
```

PROPAGATE_MODE

- If CLUSTONE is circularly configured, it sets whether or not another CLUSTONE applies the transactio ns of which a CLUSTONE applied.
- It can be set only in slave.
  - The default value is '0' and it does not PROPAGATE.
  - If it is set to 1, it is PROPAGATEd. If it is set to 0, it is not PROPAGATEd.

- Settings applied to all groups

```
PROPAGATE_MODE = 1
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    PROPAGATE_MODE = 1
    ....
    ....
}
```

UPDATE_APPLY_MODE

- It is used for the update operation in slave.
  - 0: If only the primary keys are same, the update operation is performed. (The default value)
  - 1: If the primary key is same with the value of before updating, the update operation is performe d.
  - 2: If only the primary keys are same, the update operation is performed. The values before and af ter updating are compared and if they are different, then it leaves the logs.
- If it is set to 0, the previous value is not checked and the update is performed.
- If it is set to 1 and the previous value is different, the update is failed and leaves the conflict log.
- If it is set to 2 and the previous value is different, the update is succeeded and leaves the conflict log.

- Settings applied to all groups

```
UPDATE_APPLY_MODE = 1
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    UPDATE_APPLY_MODE = 1
    ....
    ....
}
```

For a column whose data type is long varchar or long varbinary, then only the lengths are comp ared for the performance reason.

TCP_NODELAY

- It is used only in master.
- It sets TCP_NODELAY option for the CDC transfer socket. (This option does not affect the sync. It is fi xed to TCP_NODELAY on.)
  - 0: It offs the socket TCP_NODELAY option.
  - 1: It ons the socket TCP_NODELAY option. (Default)

- Settings applied to all groups

```
TCP_NODELAY = 1
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    TCP_NODELAY = 1
    ....
    ....
}
```

HEARTBEAT_TIMEOUT

- It can be set in master and slave.
- It sets the maximum time (second) maintaining connection if the connection is not smooth due to ne twork disconnection or system error after replication connection between master and slave.
- The default value is 30 (seconds).
  - The minimum value is 10 (seconds).

- Settings applied to all groups

```
HEARTBEAT_TIMEOUT = 40
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    HEARTBEAT_TIMEOUT = 40
    ....
    ....
}
```

SKIP_COMMENT

- It can be set in master.
- It is available when a specific transaction is not replicated.
- If the same value is input as a comment when executing transaction commit after setting the corresp onding value, then it skips that transaction instead of replicating it.

- Settings applied to all groups

```
SKIP_COMMENT = 'DO_SKIP'
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    SKIP_COMMENT = 'DO_SKIP'
    ....
    ....
}
```

LOG_CAPTURE_INTERVAL_1

- It can be set in master.
- It sets the execution cycle of capture operated in master. The execution cycle is set in millisecond.
- It is used to quickly detect the changes of the redo log file.
- If the value of redo log file is not changed after executing 10 times with that value, then it is converte d to the value of LOG_CAPTURE_INTERVAL_2 and performs capture.
- The default value is 200 (0.2 seconds), and the unit is millisecond.

- Settings applied to all groups

```
LOG_CAPTURE_INTERVAL_1 = 200
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    LOG_CAPTURE_INTERVAL_1 = 200
    ....
    ....
}
```

LOG_CAPTURE_INTERVAL_2

- It can be set in master.
- It sets the execution cycle of capture operated in master. The execution cycle is set in millisecond.
- It is used to quickly detect the changes of the redo log file.
- If the value of redo log file is not changed after executing 10 times with the value of LOG_CAPTURE_ INTERVAL_1, then it is converted to the corresponding value and performs capture.
- The default value is 1000(1 second), and the unit is millisecond.

- Settings applied to all groups

```
LOG_CAPTURE_INTERVAL_2 = 1000
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    LOG_CAPTURE_INTERVAL_2 = 1000
    ....
    ....
}
```

CLUSTER

- It can be set in slave.
- It sets the connection information of N masters whih is operated in cluster environment. (It consists o f ID, MASTER_IP and PORT information.)
  - ID: It is an identifier of master in slave, and only number can be input. Once the value is set, it sho uld not be altered.
  - MASTER_IP: It sets IP in which master is being operated.
  - PORT:It sets port in which master is being operated.

- Cluster can be set only within a group.

```
GROUP_NAME = testGROUP
{
    CLUSTER = (ID=1, MASTER_IP=192.0.0.100, PORT=21102),
              (ID=2, MASTER_IP=192.0.0.101, PORT=21103)
    ....
    ....
}
```

TXDATA_FILE_SIZE

- It sets the maximum size of TxData file.
- It can be set in master.
- The unit is MB (megabytes).
- The default value is 500 MB.
  - The minimum value is 100 MB.
  - The maximum value is 4 GB (4096).

- Settings applied to all groups

```
TXDATA_FILE_SIZE = 1024
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    TXDATA_FILE_SIZE = 1024
    ....
    ....
}
```

TXDATA_FILE_PATH

- It sets the path in which the TxData file is to be stored.
  - The absolute path should be used.
  - The path should be specified by using single quote (').
- It can be set only in master.

- Settings applied to all groups

```
TXDATA_FILE_PATH = '/data/TxData'
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    TXDATA_FILE_PATH = '/data/TxData'
    ....
    ....
}
```

PACKET_COMPRESSION_MODE

- It can be set in master.
- It sets whether to compress the data which is transferred from master to slave.
  - 1: Enable (Default)
  - 0: Disable

- It can be set in a specific group.

```
GROUP_NAME = testGROUP
{
    PACKET_COMPRESSION_MODE = 1
    ....
    ....
}
```

SHARED_MEMORY_NAME

- It specifies the name of the shared memory which is used during running CLUSTONE.
  - The name should be unique so that the redundant names do not exist within the clustone group in the same equipment.
- Maximum seven characters are allowed to input.
- It can be set only in master.

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    SHARED_MEMORY_NAME = '_CLUON'
    ....
    ....
}
```

SHARED_MEMORY_KEY

- It specifies the shared memory key which is used during running CLUSTONE.
  - It should be unique so that the redundant values do not exist within the clustone group in the same equipment.
- It can be set only in master.

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    SHARED_MEMORY_KEY = 4321102
    ....
    ....
}
```

52.4 Operating

CLUSTONE can be operated in D/A or C/S environment of GOLDILOCKS.

CONFIG file or PROTOCOL configuration of ODBC.INI should be used according to the environments as f ollows. CONFIG configuration takes precedence over ODBC.INI configuration.

Table 52-3 CONFIG, ODBC.INI

Configuration	Description
PROTOCOL=DA	It is used in D/A environment. (Default)
PROTOCOL=TCP	It is used in C/S environment.

The running contents during the operation can be viewed through trace log.

Item	File
Clustone master	$GOLDILOCKS_DATA/trc/clustone_master.trc
Clustone slave	$GOLDILOCKS_DATA/trc/clustone_slave.trc

GOLDILOCKS Connection Policy

It is the policy for CLUSTONE to connect to GOLDILOCKS.

The config properties relating to GOLDILOCKS connection are DSN, PROTOCOL, HOST_IP, HOST_EXTERN AL_IP, HOST_PORT, USER_ID, USER_PW.
(USER_ENCRYPT_PW is listed as USER_PW because it replaces USER_PW.)

Correct Examples

- It is available just with USER_ID and USER_PW because it is connected as D/A.

Item	File
CONFIG	USER_ID=test, USER_PW=test
GOLDILOCKS configuration of odbc.ini	-

- The information should exist either in CONFIG or in odbc.ini. HOST_IP and HOST_PORT are ignored b ecause it is operated in D/A.

Item	File
CONFIG	HOST_IP=127.0.0.1, HOST_PORT=22581,USER_ID=test
GOLDILOCKS configuration of odbc.ini	USER_PW=test

- USER_ID is operated with the test account because it is connected with TCP and CONFIG takes prece dence.

Item	File
CONFIG	PROTOCOL=TCP, USER_ID=test, USER_PW=test
GOLDILOCKS configuration of odbc.ini	HOST_IP=127.0.0.1, HOST_PORT=22581, USER_ID=test2, USER_PW=test2

- HOST_EXTERANL_IP can not be set in slave even though it is connected as D/A.

Item	File
CONFIG	PROTOCOL=DA, HOST_EXTERNAL_IP=192.168.0.10, USER_ID=test
GOLDILOCKS configuration of odbc.ini	HOST_IP=127.0.0.1, HOST_PORT=22581,USER_PW=test

Wrong Examples

- It is connected as TCP, but HOST_IP does not exist.

Item	File
CONFIG	HOST_EXTERNAL_IP=192.168.0.10, USER_ID=test
GOLDILOCKS configuration of odbc.ini	PROTOCOL=TCP, HOST_PORT=22581,USER_PW=test

Executing Option

CLUSTONE should be used with the following options at run-time.

Table 52-4 Executing options

Option	Description	Remarks
--start | -s	It starts CLUSTONE.	It should be used together with --master | --slave.
--stop | -t	It terminates CLUSTONE.	It should be used together with --master | --slave.
--master | -m	It is performed in master mode.	It should be used together with --start | --stop.

Option	Description	Remarks
--slave | -l	It is performed in slave mode.	It should be used together with --start | --stop.
--status | -u	It displays the operating status of CLUSTONE.	It should be used together with --master | --slave .
--conf | -c	It sets the path of configuration file which is required when executing CLUSTONE.	It is input in --conf CONFIG_FILE format. It should be used together with --start. If it is not explicitly set, master uses $GOLDILOCKS_DATA/clustone.master.conf, and slave uses $GOLDILOCKS_DATA/clustone.slave.conf.
--silent | -i	It sets not to output messages.	-
--reset | -r	It resets operational information of the replication.	It is input in --reset TABLE_NAME or --reset all format. It should be described within a single quote (') when resetting multiple tables.
--group | -g	It sets a specific group.	It is input in --group GROUP_NAME format.
--help | -h	It outputs the help message.	
--encrypt | -e	It encrypts the user password with the given key.	-
--key | -k	It sets the encryption key when performing the --encrypt option. If USER_ENCRYPT_PW is used in the config, it sets the decryption key.	-
--info | -o	It displays the status of the table which is currently being replicated.	It should be used together with --master, --group.
--stand-alone | -S	It is operated in a standalone mode of cluster environment. (It is operated in a cluster mode in cluster environment.)	It is valid only in master. (It is set in the configuration file in slave.)
--dump | -d	It dumps txData file.	It should be input in --dump txdatafile format.

- It executes all groups in master mode by using the default environment file.

```
prompt> clustone --master --start
```

- It terminates all groups in master mode.

```
prompt> clustone --master --stop
```

- It executes all groups in slave mode by using the default environment file.

```
prompt> clustone --slave --start
```

- It terminates all groups in slave mode.

```
prompt> clustone --slave --stop
```

- It executes only the TEST_GROUP group in master mode.

```
prompt> clustone --master --start --group TEST_GROUP
```

- It terminates only TEST_GROUP group among the groups operated in master mode.

```
prompt> clustone --master --stop --group TEST_GROUP
```

- It sets the TEST_CONFIG file in slave mode and executes it.

```
prompt> clustone --slave --start --conf TEST_CONFIG
```

- It encrypts GOLDILOCKS user password.

```
prompt> clustone --encrypt test --key 1234
clustone Encrypted Passwd : 'YFH+bpBPNvk='
```

- USER_ENCRYPT_PW is set in the config.

```
prompt> clustone --master --start --key 1234
```

- It deletes the existing replications of master and slave, and newly starts it.

```
prompt> clustone --master --start --reset all
prompt> clustone --slave --start --reset all
```

- It deletes the existing replications of only the tables T1, T2 in master, and starts it newly from the curr ent point.

```
prompt> clustone --master --start --reset 'T1, T2'
prompt> clustone --slave --start
```

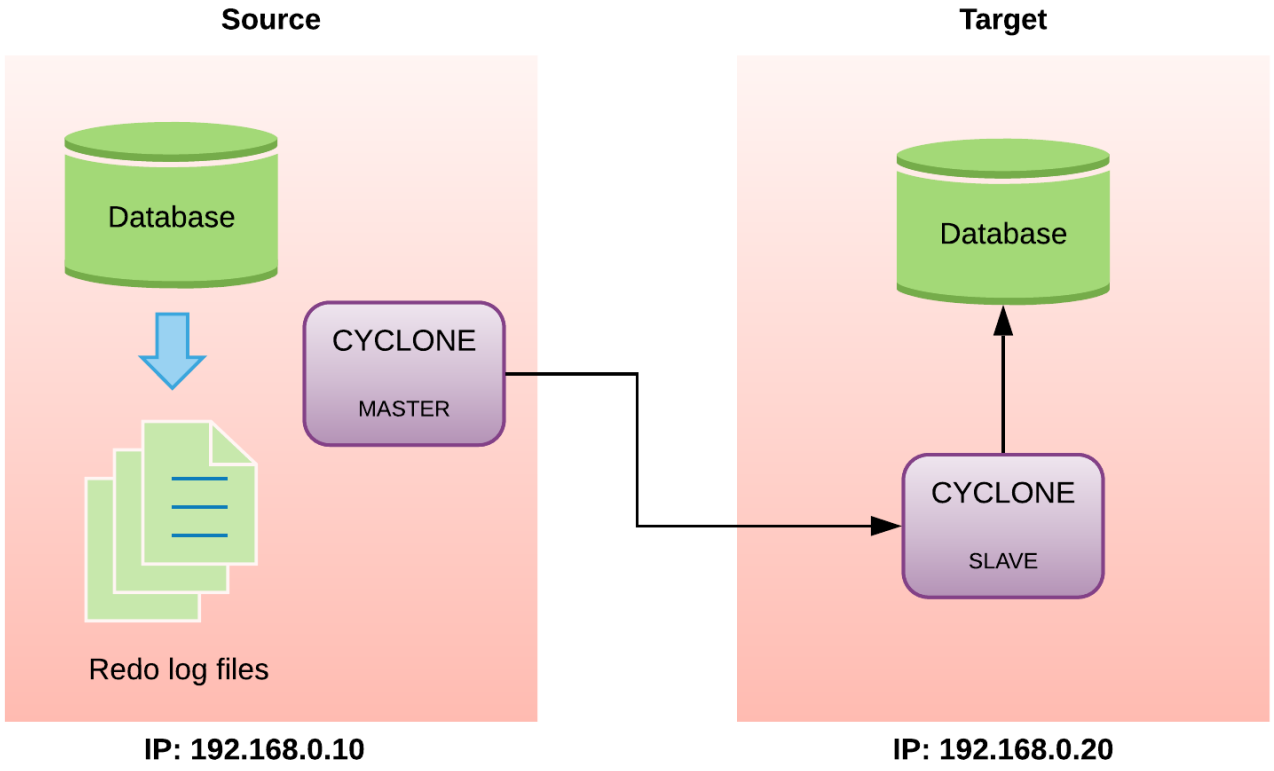
- It displays the current replication status per each table on master side.

```
prompt> clustone --master --info --group GROUP1
================================================
 GROUP NAME = GROUP1
================================================
 SCHEMA NAME         : PUBLIC
 TABLE NAME          : T1 (GIVE-UP (DDL-LSN:129541))
 PHYSICAL ID         : 5299989643264
================================================
 SCHEMA NAME         : PUBLIC
 TABLE NAME          : T2 (ACTIVE (CAPTURE-START-LSN:128922))
 PHYSICAL ID         : 35549444308992
================================================
 SCHEMA NAME         : PUBLIC
 TABLE NAME          : T3 (ACTIVE (CAPTURE-START-LSN:128922))
 PHYSICAL ID         : 35558034243584
================================================
 SCHEMA NAME         : PUBLIC
 TABLE NAME          : T4 (ACTIVE (CAPTURE-START-LSN:128922))
 PHYSICAL ID         : 35566624178176
================================================
   TOTAL   COUNT : 4
   GIVE-UP COUNT : 1
================================================
```

52.5 Operating Examples

The structure of operating CLUSTONE is as follows.

Figure 3 The structure of operating CLUSTONE



- Device structure
  - The source device which is the original data
    - GOLDILOCKS IP: 192.168.0.10
    - GOLDILOCKS port: 22581
    - The table to be replicated: T1, T2
  - The remote target device 1 for replication
    - GOLDILOCKS IP: 192.168.0.20
    - GOLDILOCKS port: 22581
  - The remote target device 2 for replication
    - GOLDILOCKS IP: 192.168.0.21
    - GOLDILOCKS port: 22581

Requirements

- It provides a method to replicate members in the same cluster group in cluster environment.
- Available environment
  - Master should be cluster environment. (It is automatically detected.)
  - Slave can be either standalone or cluster environment.
- The status of master connected to slave is called as a node.

Node Type

In cluster environment, CLUSTONE classifies master connecting to slave into a trust node and a non-trust node.

Table 52-5 Node in cluster environment

Node	Description	Replication
Trust node	It is a node all of whose tables participating in replication are online.	O
Non-trust node	It is a node one or more of whose tables participating in replication are offline.	X

Operation According to Node Status

- Switching from a non-trust node to a trust node
  - It is executed when all tables are switched to online by rebalancing the cluster member while an offline table exists.
  - The node is automatically switched from a non-trust node to a trust node, and participates in replication.
- Switching from a trust node to a non-trust node
  - It is executed when a table being replicated in another cluster member in operation is altered while the cluster member is terminated and is not operated.
  - A non-trust node is automatically altered when a cluster member joins, and it is excluded from replication.

- The replication is maintained when one or more trust nodes exist among masters participating in the replication.
- A trust node is not restricted to a specific node, and any node is allowed regardless of node types.
- If all masters participating in replication becomes non-trust nodes, the replication does not proceed any more, and the replication does not start again even when they become trust nodes.
- At least one or more trust nodes are required to proceeds the replication in cluster environ

> ment.

Operating Order

1. Set the original GOLDILOCKS environment.
2. Set the remote GOLDILOCKS environment.
3. Set the CLUSTONE MASTER environment.
4. Set the CLUSTONE SLAVE environment.
5. Execute and operate it.

Original GOLDILOCKS Configuration

Execute all described **Requirements**.

Create tables T1, T2 for testing.

```
gSQL > CREATE TABLE T1( COL1 INTEGER PRIMARY KEY, COL2 VARCHAR(20) );
gSQL > CREATE TABLE T2( COL1 INTEGER PRIMARY KEY, COL2 VARCHAR(20) );
gSQL > COMMIT;
```

Remote GOLDILOCKS Configuration

Execute **Registering User and Setting Privileges**.

CLUSTONE MASTER Configuration

- CLUSTONE MASTER configuration file
  - Storing path: $GOLDILOCKS_DATA/conf/clustone.master.conf
  - It is the source device.

- It is connected with D/A of the same device, so HOST information does not exist.

```
USER_ID = cdc_user
USER_PW = cdc_password
GROUP_NAME = GROUP1
```

```
{
    PORT = 21102
    CAPTURE_TABLE =
    (
        T1,
        T2
    )
}
```

CLUSTONE SLAVE Configuration

- CLUSTONE SLAVE configuration file
  - Storing path: $GOLDILOCKS_DATA/conf/clustone.slave.conf
  - It is the target device.
- The following is an example of operating two slaves, so it describes two information.

- The connection information of the first slave and the information about the original cluster's member.
  - The following example describes the connection information of two masters in cluster, but actual ly it is connected by using ID 1 information only.

```
USER_ID = cdc_user
USER_PW = cdc_password
HOST_IP = 192.168.0.20
HOST_PORT = 22581
GROUP_NAME = GROUP1
{
    PORT = 21102
    CLUSTER = ( ID=1, MASTER_IP = 192.168.0.10, PORT=21102 ),
              ( ID=2, MASTER_IP = 192.168.0.10, PORT=21103 )
    APPLY_TABLE =
    (
        T1 TO T1,
        T2 TO T2
    )
}
```

- The connection information of the second slave and the information about the original cluster's mem ber.
  - The following example describes the connection information of two masters in cluster, but actual ly it is connected by using ID 1 information only.

```
USER_ID = cdc_user
USER_PW = cdc_password
HOST_IP = 192.168.0.21
HOST_PORT = 22581
GROUP_NAME = GROUP2
{
    PORT = 21102
    CLUSTER = ( ID=1, MASTER_IP = 192.168.0.10, PORT=21102 ),
              ( ID=2, MASTER_IP = 192.168.0.10, PORT=21103 )
    APPLY_TABLE =
    (
        T1 TO T1,
        T2 TO T2
    )
}
```

Executing and Operating

- Execute CLUSTONE master.
  - It should be executed in the source device.

```
clustone --master --start --conf $GOLDILOCKS_DATA/conf/clustone.master.conf
```

- Execute CLUSTONE SLAVE.
  - It should be executed in the target device.
  - It is executed in both devices.

```
clustone --slave --start --conf $GOLDILOCKS_DATA/conf/clustone.slave.conf
```

Initializing Replication Information

- Initializing replication information
  - It restarts the replication from current point for the entire table or a specific table participating in the replication.
  - It is performed by using the --reset TABLE_NAME option or the --reset ALL option.
- Differences of between master and slave when performing the --reset option
  - Master stores table information and column information participating in the replication as meta. I f RESET is performed in master by using the --Reset ALL option or the Reset Table_name option, i

t removes the meta in the table and configures meta again by using the information at the curre nt point.

- The --reset option should be performed in master to replicate again the table which is given up during the replication performing DDL.

◦ The slave stores the information for recovery when restating the replication. *--reset All* or *--reset TABLE_NAME* should be described when executing slave as same as when executing master.

- Add the --reset ALL option to initialize all tables participating in the replication when perfor ming master/ slave.

  prompt> clustone --master --start --reset ALL

  prompt> clustone --slave --start --reset ALL

- Add the --reset TABLE_NAME option when performing master to initialize the replication in formation of a specific table.
  - The table name of when performing slave should be as same as that of when performin g master.

  prompt> clustone --master --start --reset T1

  prompt> clustone --slave --start

- A single quote (') should be used to initialize two or more tables.

  prompt> clustone --master --start --reset 'T1 T2'

  prompt> clustone --master --start --reset 'T1, T2'

52.6 Monitoring (CYMON)

CYMON (CYclone MONitor) is a tool for monitoring CYCLONE and CLUSTONE which are CDC replication tools. It periodically updates the monitoring information of CDC replication to GOLDILOCKS operated in master.

Table 52-6 Execution files

File name	Description
cymon	Monitoring cyclone and clustone

Configuration File

The configuration file of cymon uses the configuration file used when running clustone as master. If a specific configuration file is not set by using the --conf option, the $GOLDILOCKS_DATA/conf/clustone.master.conf file is read as the default when clustone is run as master.

CYMON should be run on the device which is as same as the device of which CLUSTONE runs as master.

Monitoring Contents

CYMON

CYMON periodically updates the operating information of CLUSTONE to the CLUSTONE_MONITOR_INFO table of GOLDILOCKS. The monitoring information is as follows.

Table 52-7 CLUSTONE_MONITOR_INFO

Column	Description
ID	It is the identifier of the connected SLAVE.
MASTER_GROUP_NAME	It is the CLUSTONE group name which is operated as master.
SLAVE_GROUP_NAME	It is the slave group name of CLUSTONE which is connected to the master.
TIME	It is the information of time at which the information was updated. (YYYY-MM-DD HH24:MI:SS)
MASTER_STATE	It is the state of CLUSTONE which is operated as MASTER. • N/A: The state is unknown.

Column	Description
	• READY: It is waiting for connection of SLAVE • RUNNING: SLAVE is connected and the replication is running.
SLAVE_STATE	It is the state of CLUSTONE which is operated as SLAVE. • N/A: The state is unknown. • RUNNING: The replication is running.
MASTER_PORT	It is the information of PORT on which CLUSTONE operated in MASTER is waiting for slave.
SLAVE_IP	It is an IP address of the device connected with CLUSTONE SLAVE.
REDO_LOG_FILESEQ	It is the sequence number of the redo log file of the running GOLDILOCKS.
REDO_LOG_BLOCKSEQ	It is the block sequence number of the redo log file of the running GOLDILOCKS.
CAPTURE_FILESEQ	It is the sequence number of the redo log file captured by CLUSTONE master.
CAPTURE_BLOCKSEQ	It is the block sequence number of the redo log file captured by CLUSTONE master.
CAPTURE_COMMIT_LSN	It is the commit log number of the last transaction which was captured in CLUSTONE master. It is not updated if there is not a transaction to be captured any more.
CAPTURE_COMMIT_SCN	It is the commit scn number of the last transaction which was captured in CLUSTONE master. It is not updated if there is not a transaction to be captured any more.
CAPTURE_INTERVAL	It is the number of the remaining redo log blocks to be processed by CLUSTONE master • 1 block = 512 bytes (Redo log block size = 512 bytes)
CAPTURE_INTERVAL_SIZE	It is the size of the remaining redo log files to be processed by CLUSTONE master.
TOTAL_TX_COUNT	It is the number of all transactions captured by CLUSTONE master.
CAPTURE_TX_COUNT	It is the number of all transactions in which the replication targets are included among the transactions captured by CLUSTONE master.

INTERVAL information is the CAPTURE information for analyzing the redo log file of CLUSTONE operated as master, and it is not the information reflected by APPLIER in CLUSTONE SLAVE.

Executing and Monitoring

Executing Option

Table 52-8 Executing option

Option	Description	Remarks
--conf | -c	It sets the configuration file path.	It is input in --conf CONFIG FILE format. • It should set the file as same as the configuration file used when running CLUSTONE as master.
--start | -s	It executes CYMON.	-

Option	Description	Remarks
--stop | -t	It terminates CYMON.	-
--status | -u	It displays the operating status of CY MON.	-
--cycle | -y	It sets the update cycle of monitoring information.	It is input in --cycle X format. • It should be entered in seconds.
--key | -k	It sets the decryption key when USER_ ENCRYPT_PW is used in config.	-
--trace | -r	It simultaneously records the monitori ng information of capture on the trac e log.	The trace log is recorded in $GOLDILOCKS_DATA/trc/cy mon.trc.
--silent | -i	It sets not to output the message.	-
--help | -h	It outputs the help message.	-

53.

LOGMIRROR

53.1 LOGMIRROR

LOGMIRROR is a replication tool which copies the redo logs generated by GOLDILOCKS to the remote loc ation for the configuration of the same redo log file.

Overview

CYCLONE, using the CDC method, analyzes and replicates the redo log file generated during the operatio n of GOLDILOCKS. Therefore, if the operating server is failed when analysis of the redo log file is not com pleted, then unanalyzed redo log file is not replicated and the data is lost.

However, if the redo logs are sent to a remote location without data loss and the redo log file is configur ed by using LOGMIRROR, then this problem of CYCLONE can be solved.

LOGMIRROR is a tool for eliminating the data loss which is caused because CYCLONE uses the ASYNC m ethod, and it should work together with CYCLONE.

Operational Features

- It is performed being divided into master and slave. Only one LOGMIRROR per GOLDILOCKS is allowe d to be operated.
- Master and slave can use the TCP/ IP and infiniband communication.
  - Network speed significantly affects the GOLDILOCKS operation speed.
- The replication target is the redo log file of GOLDILOCKS.
- The original database should be operated in Transactional Data Store (TDS) mode.
  - LOGMIRROR has the same operational characteristics of CYCLONE because it should be operated together with CYCLONE.

Performance Degradation Factors of GOLDILOCKS When Ope rating LOGMIRROR

LOGMIRROR sends the redo logs generated by GOLDILOCKS to the remote location before storing them i n the file, then processes the next after it is normally processed. Therefore, the network speed between t he replication devices is an important factor of the performance, and it can cause performance degradati on than GOLDILOCKS operated without LOGMIRROR. However, the high speed infiniband is supported t

o minimize the performance degradation.

53.2 Requirements

GOLDILOCKS Requirement

The followings should be set in GOLDILOCKS before performing LOG MIRROR.

LOG_MIRROR_MODE

The LOG_MIRROR_MODE property is set to use LOGMIRROR in GOLDILOCKS. When it is activated, the re sources which temporarily store the redo logs generated by GOLDILOCKS before LOGMIRROR sends the m to a remote location are allocated.

- If the LOG_MIRROR_MODE property is changed, it is applied after restarting GOLDILOCKS.
- The changes are added or reflected in the property file.
  - Property file: goldilocks.properties.conf
  - Property setting: LOG_MIRROR_MODE = 1

Or, the following statement is executed in gSQL.

```
gSQL> ALTER SYSTEM SET LOG_MIRROR_MODE=1 SCOPE=FILE;
System altered.
```

LOG_MIRROR_SHARED_MEMORY_STATIC_SIZE

LOG_MIRROR_SHARED_MEMORY_STATIC_SIZE sets the temporary storage space used by LOGMIRROR.

- It is a temporary storage space used before storing redo log file from redo log buffer.
  - The set value is related to MAXIMUM_FLUSH_LOG_BLOCK_COUNT used when storing the redo l og file.
- The default value is 100 M.
  - The minimum value is 10 M, and the maximum value is 1 G.
  - Too small set value can affect the performance.

- If LOG_MIRROR_SHARED_MEMORY_STATIC_SIZE property is changed, then it is applied after the G OLDILOCKS is restarted.
- The changes are added or reflected in the property file.
  - Property file: goldilocks.properties.conf
  - Property setting: LOG_MIRROR_SHARED_MEMORY_STATIC_SIZE = the set value

Or, the following statement is executed in gSQL.

```
gSQL> ALTER SYSTEM SET LOG_MIRROR_SHARED_MEMORY_STATIC_SIZE = 200M SCOPE=FILE;
System altered.
```

LOG_MIRROR_TIMEOUT

When interworking with LOGMIRROR, GOLDILOCKS includes the step of waiting for the response from LOG MIRROR. Too slow response causes the degradation of GOLDILOCKS performance. LOG_MIRROR_TIMEOUT sets the response time so when there is not any response after the time is over, LOGMIRROR stops the service, but only the GOLDILOCKS continues the service.

When restarting LOGMIRROR to perform LOGMIRROR again, it is normally operated after the recovery process.

- The default value is '0'.
  - 0 refers to an infinite waiting.
  - It can be set in seconds.
- If LOG_MIRROR_TIMEOUT property is changed, it is immediately reflected.
- The changes are added or reflected in the property file.
  - Property file: goldilocks.properties.conf
  - Property setting: LOG_MIRROR_TIMEOUT = the set value

Or, the following statement is executed in gSQL.

```
gSQL> ALTER SYSTEM SET LOG_MIRROR_TIMEOUT = 20;
System altered.
```

The GOLDILOCKS requirements for operating CYCLONE should also be applied.

53.3 Configuration

Configuration File

The information and options required for the operation can be set by using the configuration file when executing LOGMIRROR.

- If a specific configuration file is not set by using the --conf option, a specific file is read at $ GOLDILOCKS_DATA / conf directory. The logmirror.master.conf file is read when operating as master and the logmirror.slave.conf file is read when operating as slave.

Table 53-1 Configuration contents

Name	Description	Coverage
PORT	It sets the port to be used for the communication between master and slave.	Master/ slave
DSN	It sets Data Source Name.	Master
HOST_IP	It sets the host IP address which is operated by GOLDILOCKS.	Master
HOST_PORT	It sets the host port which is operated by GOLDILOCKS.	Master
USER_ID	It sets the user name.	Master
USER_PW	It sets the user password.	Master
USER_ENCRYPT_PW	It sets encrypted user password.	Master/ slave
LOG_PATH	It sets the path in which the replicated redo log file is to be stored.	Slave
MASTER_IP	It sets the IP address of the device which is being operated by the log mirror master.	Slave
HEARTBEAT_TIMEOUT	It sets the maximum time (second) maintaining connection if the connection is not smooth due to network disconnection or system error after replication connection.	Master/ slave
TCP_NODELAY	It sets TCP_NODELAY option of a socket. (The default value is 1.) • 0: TCP_NODELAY off • 1: TCP_NODELAY on	Master

Configuration Options

PORT

- It sets the port used for the communication between master and slave.
- It can be set in master and slave.

```
PORT=21106
```

DSN

- It sets the data source name required for the access to GOLDILOCKS.
- It can be set in master.

```
DSN = GOLDILOCKS
```

HOST_IP

- It sets the IP address of GOLDILOCKS in which LOGMIRROR is to be operated.
- It should be set together with HOST_PORT.
- It can be set in master.

```
HOST_IP = 127.0.0.1
```

HOST_PORT

- It sets the port of GOLDILOCKS on which LOGMIRROR is to be operated.
- It should be set together with HOST_IP.
- It can be set in master.

```
HOST_PORT = 22531
```

USER_ID

- It sets the user ID required for the access to GOLDILOCKS.
- It can be set in master.

```
USER_ID = testID
```

USER_PW

- It sets the user password required for the access to GOLDILOCKS.
- It can be set in master.

```
USER_PW = testPW
```

USER_ENCRYPT_PW

- It sets the user password which is required for the access to GOLDILOCKS by encrypting.
- It is used instead of USER_PW.
- Encrypted user password is created by [logmirror --encrypt *user password* --key *the key to be encrypted*].
- If this value is used, then --key option should be used when executing logmirror. (In this case, the key value as same as that created with --encrypt should be used.

```
USER_ENCRYPT_PW = 't33KImiqvhqNyfN+uZmFrw=='
```

LOG_PATH

- It sets the path in which the replicated redo log file is to be stored.
  - The set path should be an absolute path.
  - The path is set by using a single quote (') .
- It can be set in slave.

```
LOG_PATH = '/data/wal'
```

MASTER_IP

- It sets the IP address of the device of which LOGMIRROR master is operating.
- It can be set in slave.

```
MASTER_IP = 192.168.0.100
```

HEARTBEAT_TIMEOUT

- It can be set in master and slave.
- It sets the maximum time (second) maintaining connection if the connection is not smooth due to network disconnection or system error after replication connection between master and slave.
- The default value is 30 (seconds).
  - The minimum value is 10 (seconds).

```
HEARTBEAT_TIMEOUT = 40
```

TCP_NODELAY

- It is used only in master.
- It sets TCP_NODELAY option for the LogMirror transfer socket.
  - 0: It offs the socket TCP_NODELAY option.
  - 1: It ons the socket TCP_NODELAY option. (Default)

```
TCP_NODELAY = 1
```

53.4 Operating

The executing environment of master/ slave of LOGMIRROR is as follows.

Table 53-2 Executing environment

Item	Whether to operate GOLDILOCKS	Description
Master	O	When operating as master, LOGMIRROR should be operated at the device in which GOLDILOCKS is operated.
Slave	X	When operating as slave, GOLDILOCKS is not required to be operated and the storage space in the disk is required to store the redo log file.

The operating contents during the execution can be viewed through a trace log.

Item	File
Master	$GOLDILOCKS_DATA/trc/LogMirror_master.trc
Slave	$GOLDILOCKS_DATA/trc/LogMirror_slave.trc

For more information about the error messages and handlings stored in the trace log, refer to **Troubleshooting of LOGMIRROR**.

Operating LOGMIRROR

LOGMIRROR can normally replicates the redo log file only when the following procedures should be completed even after the GOLDILOCKS configuration, initializing and operating master/ slave.

SWITCH of the redo log file should occur to start the replication. LOGMIRROR starts the normal operation after generating the new log file. Therefore, the following step should be performed.

```
gSQL> ALTER SYSTEM SWITCH LOGFILE;
System altered.
```

The redo log file replicated by LOGMIRROR is continuously stored, and it is not automatically deleted. Therefore, the file management such as deleting or moving is regularly required according to the environment of the operating device.

Executing Option

Table 53-3 Execution options

Option	Description	Remarks
--start | -s	It executes LOGMIRROR.	It should be used together with --master | --slave.
--stop | -t	It terminates LOGMIRROR.	It should be used together with --master | --slave.
--master | -m	It is performed in master mode.	It should be used together with --start | --stop.
--slave | -l	It is performed in slave mode.	It should be used together with --start | --stop.
--conf | -c	It sets the configuration file path.	It is input in --conf CONFIG_FILE format.
--infiniband | -f	It uses infiniband network environment.	It is not input when using TCP/IP environment.
--silent | -i	It sets not to output the message.	-
--help | -h	It sets to output the help message.	-

- Execute it in master mode by using the default configuration file.

```
prompt> logmirror --master --start
```

- Terminate the LOGMIRROR which is being operated in master mode.

```
prompt> logmirror --master --stop
```

- Execute it in slave mode by using the default configuration file.

```
prompt> logmirror --slave --start
```

- Terminate the LOGMIRROR which is being operated in slave mode.

```
prompt> logmirror --slave --stop
```

- Execute it in master mode by using TEST_CONFIG file and infiniband network.

```
prompt> logmirror --master --start --conf TEST_CONFIG --infiniband
```

- Execute it in slave mode by using TEST_CONFIG file and infiniband network.

```
prompt> logmirror --slave --start --conf TEST_CONFIG --infiniband
```

For examples of initializing LOGMIRROR, refer to **LOGMIRROR**.

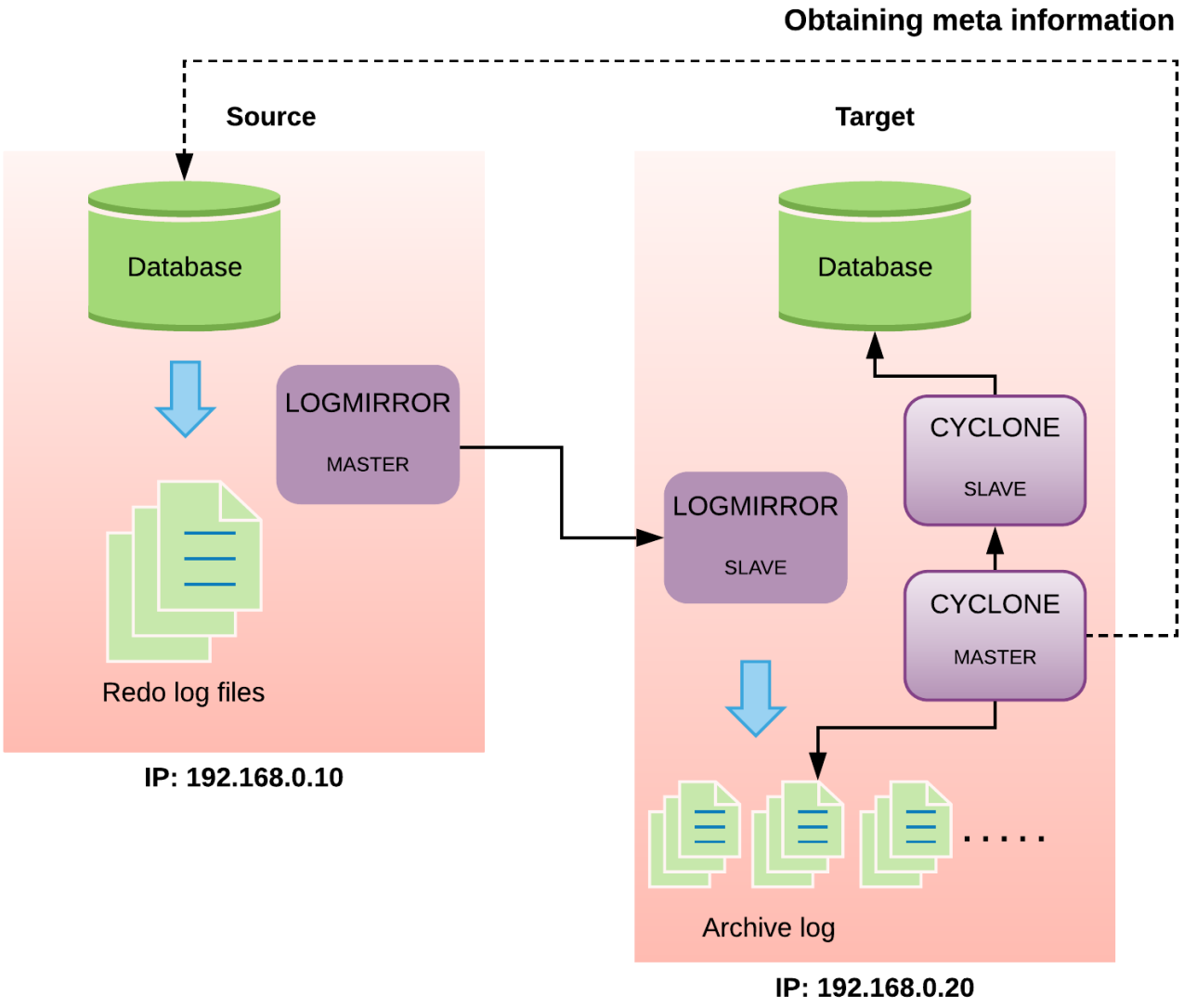
53.5 Examples of Interworking with CYCLONE

Interworking of CYCLONE, the CDC replication tool, with LOGMIRROR, the replication tool of the redolo g file, eliminates the risk of data loss.

Tool	Function	Interworking
CYCLONE	Replication of CDC	If it is independently operated it may cause the data loss.
LOGMIRROR	Replication of REDO LOG FILE	If it is operated together it does not cause the data loss.

The following describes the examples of interworked operation, and its structure.

Figure 1 Example of operating structure



- Device structure
  - The source device which is the original data
    - IP: 192.168.0.10

        - GOLDILOCKS listen port: 22581
        - The table to be replicated: T1, T2
- The remote target device for replication
    - IP: 192.168.0.20
- Two devices executes interworking between LOGMIRROR and CYCLONE for the replication without data loss.
    - Source device: LOGMIRROR (master)
    - Target device: LOGMIRROR (slave) + CYCLONE (master, slave)

Operating Order

1. Set the configuration of CYCLONE and LOGMIRROR in the original GOLDILOCKS.
2. Set the configuration of LOGMIRROR MASTER/SLAVE.
3. Execute LOGMIRROR MASTER/SLAVE.
4. Perform LOG FILE SWITCH of the original GOLDILOCKS for the normal operation of LOGMIRROR.
5. Set the remote GOLDILOCKS configuration.
6. Set the CYCLONE MASTER/SLAVE configuration.
7. Execute CYCLONE MASTER/SLAVE.

Configuring CYCLONE and LOGMIRROR in Original GOLDILOCKS

Refer to the followings.

- CYCLONE configuration: **Requirements**
- LOGMIRROR configuration: **Requirements**

Create the tables T1, T2 for a test after the configuration.

```
gSQL > CREATE TABLE T1( COL1 INTEGER PRIMARY KEY, COL2 VARCHAR(20) );
gSQL > CREATE TABLE T2( COL1 INTEGER PRIMARY KEY, COL2 VARCHAR(20) );
gSQL > COMMIT;
```

Configuring LOGMIRROR MASTER/SLAVE

- LOGMIRROR MASTER configuration file
    - Storage location: $GOLDILOCKS_DATA/conf/logmirror.master.conf
    - It is the source device.

LOG MIRROR MASTER should be operated on the device in which the original GOLDILOCKS is operated.

- Enter DSN or HOST information. HOST information is entered in the following example.

```
HOST_IP   = 192.168.0.10
HOST_PORT = 22581
```

- Use CYCLONE USER created in the previous step.

```
USER_ID = cdc_user
USER_PW = cdc_password
```

- Set the PORT to be used for LOGMIRROR communication.

```
PORT = 21106
```

- LOGMIRROR SLAVE configuration file
  - Storage path: $GOLDILOCKS_DATA/conf/logmirror.slave.conf
  - It is the target device.

- Enter the IP address of the device in which LOG MIRROR MASTER is being operated.

```
MASTER_IP = 192.168.0.10
```

- The PORT which is as same as LOGMIRROR MASTER should be entered.

```
PORT = 21106
```

- Enter the absolute path in which the replicated redo log file is to be stored.

```
LOG_PATH = '/data/LogMirrorWAL'
```

Executing LOGMIRROR MASTER/ SLAVE

- Execute LOGMIRROR MASTER.
  - It should be executed in the source device.

```
logmirror --master --start --conf $GOLDILOCKS_DATA/conf/logmirror.master.conf
```

- Execute LOGMIRROR SLAVE.

- It should be executed in the target device.

```
logmirror --slave --start --conf $GOLDILOCKS_DATA/conf/logmirror.slave.conf
```

Executing LOGFILE SWITCH of Original GOLDILOCKS for Normal Operation of LOGMIRROR

- LOGFILE SWITCH for starting to record LOGMIRROR file.
  - It should be executed in the source device.

```
gSQL> ALTER SYSTEM SWITCH LOGFILE;
```

When it is executed as above, the redo log file is generated in the path set in the SLAVE device. If the redo log file is not generated, a user should check the directory privilege or check whether the directory is created.

Configuring Remote GOLDILOCKS

It is required to check the table configuration for the normal operation and replication of GOLDILOCKS operated on the target device, and check the schema information.

The tables T1, T2 are created for test as follows.

```
gSQL > CREATE TABLE T1( COL1 INTEGER PRIMARY KEY, COL2 VARCHAR(20) );
gSQL > CREATE TABLE T2( COL1 INTEGER PRIMARY KEY, COL2 VARCHAR(20) );
gSQL > COMMIT;
```

Configuring CYCLONE MASTER/ SLAVE

- CYCLONE MASTER configuration file
  - Storage path: $GOLDILOCKS_DATA/conf/cyclone.master.conf
  - It is the target device.

- Enter DSN or HOST information of the source device. HOST information is entered in the following example.

```
HOST_IP   = 192.168.0.10
HOST_PORT = 22581
```

- Use CYCLONE USER

```
USER_ID = cdc_user
USER_PW = cdc_password
GROUP_NAME = GROUP1
{
    PORT = 21102
    CAPTURE_TABLE =
    (
        T1,
        T2
    )
}
```

- CYCLONE SLAVE configuration file
  - Storage path: $GOLDILOCKS_DATA/conf/cyclone.slave.conf
  - It is the target device.

- Connect to GOLDILOCKS which is operated in the target device with D/A. (Therefore, DSN and HOST information does not exist.)

```
USER_ID = cdc_user
USER_PW = cdc_password
```

- CYCLONE master is being operated on the same device.

```
MASTER_IP = 192.168.0.20
GROUP_NAME = GROUP1
{
```

- Enter the path set by LogMirror slave.

```
LOG_PATH = '/data/LogMirrorWAL'
    PORT = 21102
    APPLY_TABLE =
    (
        T1 TO T1,
        T2 TO T2
    }
}
```

Executing CYCLONE MASTER/ SLAVE

- Execute CYCLONE MASTER.
  - It should be executed in the target device when it is connected with D/A.

```
cyclonet --master --start --conf $GOLDILOCKS_DATA/conf/cyclone.master.conf
```

- Execute CYCLONE SLAVE.
  - It should be executed in the target device when it is connected with D/A.

```
cyclone --slave --start --conf $GOLDILOCKS_DATA/conf/cyclone.slave.conf
```

54.

CYFILE

54.1 CYFILE

CYFILE is a tool which uses Change Data Capture (CDC) method to store the altered data in Comma-Separated Values (CSV) format file.

Overview

It analyzes the redo log file in real time and stores the transaction executed in the database in CSV format file. It can replicate the transaction file which is recorded in an async way and in near real time to database by using the 3rd party tool, or converts it to another format.

Operational Features

- It should be operated in the device in which the database is being operated.
- A single file is stored per a group.
- It can be started or terminated in groups.
- Storing CSV file can be set in a unit of table, and one or more tables can be included in a group.
- A table can be included in multiple groups.
- The redo log file should exist in the original database. In other words, DATA_STORE_MODE should be operated in TDS.
- SUPPLEMENTAL LOGGING should exist in the original database.
  - SUPPLEMENTAL LOGGING adds an additional information to the redo log file.
- The database should be operated in ARCHIVE LOG mode.
- Its process is independent from that of GOLDILOCKS, so terminating the tool does not affect GOLDILOCKS.

Operational Restrictions

- The table to capture should have PRIMARY KEY.
- It captures the committed transaction only. Therefore, it is unable to see the contents in CSV file before commitment
- It does not support the primary key update.
  - When the primary key value is updated, then the table is given up and is not captured any more.
- The table to capture can not use a column which has Generated Always As Identity property.
- When Data Definition Language (DDL) was performed on the table to capture, then it is given up.

    - It does not affect the replication of another table.
    - It is same for the truncate table.
- Only the table which was given up with --reset TABLE_NAME among give up tables can be reset.
- The table stored in the trash bin can not be a target to capture.
- It does not support long varchar, long varbinary.

The server property **DISABLE_DDL_CDC_GIVEUP** can disable the DDL statement which causes th e replication give up to avoid user created errors. Also, the server property **DISABLE_UPDATE_P K_CDC_GIVEUP** can disable primary key update.

Others

It stores the file at the following moment.

- It stores the file from the moment when TX occurs at the first running.
- It stores the file in succession from the terminated moment even when run it again after it is terminat ed during the operation.
- Use --reset option to give up the old file and restart from the current time.

54.2 Requirements

It is required to perform GOLDILOCKS preparations, user registration and privilege settings.

GOLDILOCKS Requirements

The followings should be set in GOLDILOCKS before starting CYFILE.

SUPPLEMENTAL LOGGING

SUPPLEMENTAL LOGGING stores an additional information together in the redo log file. The database sh ould be restarted to change the settings of operating database, but restarting is not needed when setting SUPPLEMENTAL LOGGING only in the specific table.

Setting SUPPLEMENTAL LOGGING in Database

- When setting SUPPLEMENTAL LOGGING as GOLDILOCKS property, SUPPLEMENTAL LOGGING is rec orded for all tables.
- Restart GOLDILOCKS.
- Add or update the contents in the property file.
  - Property file: goldilocks.properties.conf
  - Property setting: SUPPLEMENTAL LOG_DATA_PRIMARY_KEY = YES

Setting SUPPLEMENTAL LOGGING in Specific Table Participating in the Replicatio n

```
<add table supplemental log statement> ::=
    ALTER TABLE table_name
        ADD SUPPLEMENTAL LOG DATA ( PRIMARY KEY ) COLUMNS
    ;
```

ARCHIVE LOG

GOLDILOCKS reuses the redo log files recursively. When GOLDILOCKS reuses the redo log file being proc essed by CYFILE, then CYFILE does not proceed and is terminated. GOLDILOCKS should be operated in A RCHIVE LOG mode to ensure the continuous replication operation.

Changing Database in Operation to ARCHIVE LOG Mode

- Restart GOLDILOCKS.
- After database is shutdown, connect with sysdba and change it to ARCHIVE LOG mode in MOUNT p hase.

```
gSQL> \startup mount
Startup success
gSQL> alter database archivelog;
Database altered.
```

Setting ARCHIVE LOG Mode When Creating Database

- Update the property file before creating the database.
  - Property file: goldilocks.properties.conf
  - Property setting: ARCHIVELOG_MODE = 1

The path in which ARCHIVE LOG file is stored can be viewed and updated with 'ARCHIVELOG_ DIR'.

DATA_STORE_MODE

CYFILE performs the replication by reading the redo log files of GOLDILOCKS. Therefore, GOLDILOCKS sh ould be operated in Transactional Data Store (TDS) mode.

Changing DATA_STORE_MODE

- Restart the database.
- Add or update the corresponding information in the property file.
  - Property file: goldilocks.properies.conf
  - Property setting: DATA_STORE_MODE = 2

If the value of DATA_STORE_MODE is 1, it indicates Concurrent Data Store (CDS), and if it is 2, i t indicates Transactional Data Store (TDS).

Registering User and Setting Privileges

CYFILE retrieves and manipulates the required information during the operation. The user operating CYFILE and the proper privileges for the user are required.

Creating Database User

A specific user should be added to operate CYFILE.

```
<user definition> ::=
    CREATE USER user_identifier IDENTIFIED BY password
    [ DEFAULT TABLESPACE tablespace_name ]
    [ TEMPORARY TABLESPACE tablespace_name ]
    [ INDEX TABLESPACE {tablespace_name¦NULL} ]
    [ <schema clause> ]
    ;
<schema clause> ::=
      WITH SCHEMA [schema_name]
    ¦ WITHOUT SCHEMA
```

The following is an example of creating the user cyfile_user and the password cyfile_password.

```
gSQL> CREATE USER cyfile_user IDENTIFIED BY cyfile_password;
```

Database Privileges

Granting User Access Privilege

The following is an example for granting the access privilege to cyfile_user.

```
gSQL> GRANT CREATE SESSION ON DATABASE TO cyfile_user;
```

54.3 Configuration

Configuration File

When performing CYFILE, the information and options required for operating are set by using the configuration file.

- When a specific configuration file is not set by using the --conf option, *$GOLDILOCKS_DATA/conf/ cyfile.conf* file is read.

Table 54-1 Configuration file options

Name	Description
DSN	It sets Data Source Name.
GROUP_NAME	It sets the group name.
HOST_IP	It sets the host IP address of which GOLDILOCKS operates.
HOST_PORT	It sets the host port of which GOLDILOCKS operates.
USER_ID	It sets the user name.
USER_PW	It sets the user password.
USER_ENCRYPT_PW	It sets the encrypted password for a user.
CAPTURE_TABLE	It sets the table to be replicated.
PROTOCOL	It sets the connection type which is to be connected to GOLDILOCKS. (DA or TCP)
READ_LOG_BLOCK_COUNT	It sets the amount of data to be read at a time when operating CAPTURE.
TRANS_SORT_AREA_SIZE	It sets the size of the BUFFER to be allocated to CAPTURE.
TRANS_FILE_PATH	It sets the location in which the temporarily generated file is to be stored when operating CAPTURE.
LOG_CAPTURE_INTERVAL_1	It sets the execution cycle of capture. If the value is not changed after executing 10 times with that value, then it is converted to the value of LOG_CAPTURE_INTERVAL_2 and performs capture. (The default value is 0.2 seconds.)
LOG_CAPTURE_INTERVAL_2	It sets the execution cycle of capture. If the value is not changed after executing with the value of LOG_CAPTURE_INTERVAL_1, then it sets the execution cycle of capture. (The default value is 1 second.)
DATA_FILE_PATH	It sets the path to store CSV file. (It should be an absolute path.)
DATA_FILE_PREFIX	It sets the prefix of CSV file name.
DATA_FILE_SIZE	It sets the maximum size of CSV file. (It is an approximate size and it is not always set to the specified value.)
UPDATE_BEFORE_VALUE	It sets whether to store the value before the update in CSV when processing update SQL. (The default value is 0.)

Configuration Options

DSN

- It sets the data source name required for the access to GOLDILOCKS.
- It can be set in master and slave.
- The default value is GOLDILOCKS.

- Settings applied to all groups

```
DSN=GOLDILOCKS
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    DSN=GOLDILOCKS
    ....
    ....
}
```

GROUP_NAME

- It is necessary for distinguishing the CYFILE operation within the equipment and it is a unit of generating the operation process.
- It is the delimiter of when starting or terminating CYFILE in group unit.
- After it is set, it should not be changed. If changed, it is regarded as a new group.
- GROUP_NAME should be unique in the same equipment.
- The braces { } should be used.

```
GROUP_NAME = testGROUP
{
    ....
    ....
}
```

HOST_IP

- It sets the IP address of GOLDILOCKS in which CYFILE accesses to.
- It is valid only when PROTOCOL is set to TCP.

- Settings applied to all groups

```
HOST_IP = 127.0.0.1
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    HOST_IP = 127.0.0.1
    ....
    ....
}
```

HOST_PORT

- It sets port of GOLDILOCKS in which CYFILE accesses to.
- It should be set together with HOST_IP.

- Settings applied to all groups

```
HOST_PORT = 22531
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    HOST_PORT = 22531
    ....
    ....
}
```

USER_ID

It sets the user ID required for the access to GOLDILOCKS.

- Settings applied to all groups

```
USER_ID = testID
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    USER_ID = testID
    ....
    ....
}
```

USER_PW

It sets the user password required for the access to GOLDILOCKS.

- Settings applied to all groups

```
USER_PW = testPW
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    USER_PW = testPW
    ....
    ....
}
```

USER_ENCRYPT_PW

- It sets the user password which is required for the access to GOLDILOCKS by encrypting.
- It is used instead of USER_PW.
- Encrypted user password is created by [cyfile --encrypt *user password* --key *the key to be encrypted*].
- If this value is used, then --key option should be used when executing cyfile. (In this case, the key valu e as same as that created with --encrypt should be used.

- Settings applied to all groups

```
USER_ENCRYPT_PW = 't33KImiqvhqNyfN+uZmFrw=='
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    USER_ENCRYPT_PW = 't33KImiqvhqNyfN+uZmFrw=='
    ....
```

```
    ....
}
```

CAPTURE_TABLE

- It sets the table to capture.
  - It is set in *schema name.table name* format.
- It can be set only within a group.
- When specifying several tables, the parentheses ( ) should be used.
- The table stored in the recycle bin can not be set to be replicated.

```
GROUP_NAME = testGROUP
{
    CAPTURE_TABLE =
    (
        testSchema1.testTable1,
        testSchema1.testTable2,
        testSchema2.testTable1
    )
}
```

PROTOCOL

- It sets the type to connect to GOLDILOCKS in operation.
- It can be set to DA or TCP.
- If PROTOCOL is set to DA, neither HOST_IP nor is HOST_PORT used when accessing to GOLDILOCKS.
- The default value is DA.

• Settings applied to all groups

```
PROTOCOL = DA
```

• Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    PROTOCOL = DA
    ....
    ....
}
```

READ_LOG_BLOCK_COUNT

- It sets the number of the log blocks to be read at a time when capturing redo log file.
- The size of a log block is 512 bytes.
- The default value is 40960, and the actual read size is 20 MBytes.
  - The minimum value is 100.

- Settings applied to all groups

```
READ_LOG_BLOCK_COUNT = 1024
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    READ_LOG_BLOCK_COUNT = 1024
    ....
    ....
}
```

TRANS_SORT_AREA_SIZE

- It sets the memory space required for capturing the redo log files.
- The unit is MB (megabytes).
- The default value is 500 MB.
  - The minimum value is 10 MB.
- If the value is set too small, the performance becomes poor.

- Settings applied to all groups

```
TRANS_SORT_AREA_SIZE = 300
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    TRANS_SORT_AREA_SIZE = 300
    ....
    ....
}
```

TRANS_FILE_PATH

- If a space bigger than TRANS_SORT_AREA_SIZE is required, the temporary file is created. It sets the path in which the temporary file is to be stored.
  - The absolute path should be used.
  - The path should be specified by using single quote (').

- Settings applied to all groups

```
TRANS_FILE_PATH = '/data/TmpTrans'
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    TRANS_FILE_PATH = '/data/TmpTrans'
    ....
    ....
}
```

LOG_CAPTURE_INTERVAL_1

- It sets the execution cycle of capture in millisecond.
- It is used to quickly detect the changes of the redo log file.
- If the value of redo log file is not changed after executing 10 times with that value, then it is converted to the value of LOG_CAPTURE_INTERVAL_2 and performs capture.
- The default value is 200 (0.2 seconds), and the unit is millisecond.

- Settings applied to all groups

```
LOG_CAPTURE_INTERVAL_1 = 200
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    LOG_CAPTURE_INTERVAL_1 = 200
    ....
    ....
}
```

LOG_CAPTURE_INTERVAL_2

- It sets the execution cycle of capture in millisecond.
- It is used to quickly detect the changes of the redo log file.
- If the value of redo log file is not changed after executing 10 times with the value of LOG_CAPTURE_INTERVAL_1, then it is converted to the corresponding value and performs capture.
- The default value is 1000(1 second), and the unit is millisecond.

- Settings applied to all groups

```
LOG_CAPTURE_INTERVAL_2 = 1000
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    LOG_CAPTURE_INTERVAL_2 = 1000
    ....
    ....
}
```

DATA_FILE_PATH

- It sets the path to store CSV file.
  - The absolute path should be used.
  - The path should be specified by using single quote (').
- If it is not set, then the location in which cyfile is run becomes the default path.
- The following files are stored.
  - Data file (.dat)
  - Control file (.ctl)
  - Control mirror file (.ctl_0)

- Settings applied to all groups

```
DATA_FILE_PATH = '/home/goldilocks/dat'
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    DATA_FILE_PATH = '/home/goldilocks/dat'
    ....
```

```
    ....
}
```

DATA_FILE_PREFIX

- It sets the prefix of the name to be used when storing CSV file.
- If it is not set, then the prefix is 'cyfile'.

- Settings applied to all groups

```
DATA_FILE_PREFIX = 'SET_PREFIX'
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    DATA_FILE_PREFIX = 'SET_PREFIX'
    ....
    ....
}
```

DATA_FILE_SIZE

- It sets the approximate maximum size of CSV file.
  - The size can be maximum 16 M bigger than the input size.
- The input value is in megabytes.
  - Input 1024 for 1 giga.
- The default value is 100 M.
  - The minimum value is 30 M and the maximum value is 4096 M (4G).

- Settings applied to all groups

```
DATA_FILE_SIZE = 200
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    DATA_FILE_SIZE = 200
    ....
    ....
```

```
}
```

UPDATE_BEFORE_VALUE

- It sets whether to store the value before the update in CSV when processing UPDATE SQL.
  - Set it to 1 to preserve the previous value.
  - Set it to 0 not to preserve the previous value.
- The default value is 0 which does not preserve the previous value.

- Settings applied to all groups

```
UPDATE_BEFORE_VALUE = 1
```

- Settings applied to a specific group

```
GROUP_NAME = testGROUP
{
    UPDATE_BEFORE_VALUE = 1
    ....
    ....
}
```

54.4 Operating

CYFILE can be operated in D/A or C/S environment of GOLDILOCKS.

CONFIG file or PROTOCOL configuration of ODBC.INI should be used according to the environments as f ollows. CONFIG configuration takes precedence over ODBC.INI configuration.

Table 54-2 CONFIG, ODBC.INI

Configuration	Description
PROTOCOL=DA	It is used in D/A environment. (Default)
PROTOCOL=TCP	It is used in C/S environment.

The running contents during the operation can be viewed through trace log.

Item	File
Cyfile	$GOLDILOCKS_DATA/trc/cyfile_(groupName).trc

Executing Option

CYFILE should be used with the following options at run-time.

Table 54-3 Executing options

Option	Description	Remarks
--start | -s	It starts CYFILE.	-
--stop | -t	It stops CYFILE.	-
--status | -u	It displays the status of CYFILE.	-
--conf | -c	It sets the path of configuration file which is req uired when executing CYFILE.	It is input in --conf CONFIG_FILE format. It should be used together with --start. If it is not explicitly set, $GOLDILOCKS_DATA/conf/cyfile.conf is used.
--silent | -i	It sets not to output messages.	-
--reset | -r	It resets the information about capture.	It is input in --reset TABLE_NAME or --reset all f ormat. It should be described within a single quote (') when resetting multiple tables.
--group | -g	It sets a specific group.	It is input in --group GROUP_NAME format.
--help | -h	It outputs the help message.	-
--encrypt | -e	It encrypts the user password with the given ke y.	-

Option	Description	Remarks
--key | -k	It sets the encryption key when performing the --encrypt option. If USER_ENCRYPT_PW is used in the config, it sets the decryption key.	-
--info | -o	It displays the status of currently running table.	It should be used together with --group.

- It executes all groups by using the default environment file.

```
prompt> cyfile --start
```

- It stops all groups.

```
prompt> cyfile --stop
```

- It executes TEST_GROUP group only.

```
prompt> cyfile --start --group TEST_GROUP
```

- It stops only TEST_GROUP group among operating groups.

```
prompt> cyfile --stop --group TEST_GROUP
```

- The encrypts GOLDILOCKS user password.

```
prompt> cyfile --encrypt test --key 1234
Cyfile Encrypted Passwd : '73LsLxss6lk='
```

- USER_ENCRYPT_PW is set in the config.

```
prompt> cyfile --start --key 1234
```

- It deletes the existing operational information and newly starts it.

```
prompt> cyfile --start --reset all
```

- It deletes the existing operational information of the tables T1, T2, and starts it newly from the current point.

```
prompt> cyfile --start --reset 'T1, T2'
```

- It displays the current operational information per each table.

```
prompt> cyfile --info --group GROUP1
================================================
 GROUP NAME = GROUP1
================================================
 SCHEMA NAME         : PUBLIC
 TABLE NAME          : TEST_TABLE_02 (ACTIVE (CAPTURE-START-LSN:217368))
 PHYSICAL ID         : 36313948487680
 STATUS              : ONLINE
================================================
 SCHEMA NAME         : PUBLIC
 TABLE NAME          : TEST_TABLE_01 (ACTIVE (CAPTURE-START-LSN:217602))
 PHYSICAL ID         : 36322538422272
 STATUS              : ONLINE
================================================
   TOTAL   COUNT : 2
   GIVE-UP COUNT : 0
   NODE          : TRUST
================================================
```

54.5 Files

CYFILE stores the information about insert/ update/ delete and capture table in CSV format. It uses async method but it is operated in near real-time.

There are two types of file for storage when operating CYFILE, which are the data file and the control file. The data file stores the transaction in CSV format and the control file stores the storage information about the data file. The control file creates and operates the mirror file in preparation for data loss or damage.

Table 54-4 CYFILE

File	Name information
Data file	It is stored in CSV format, and the extension is dat.
Control file	The extension is ctl. The extension of mirror file is ctl_0.

Data File

It records I/D/U information which is performed in transaction unit and in CSV format, the information about the column of the table participating in capture and give up table information. Those files are not automatically deleted unless a user deletes them.

File Name

The location and name of the stored file is determined by DATA_FILE_PREFIX, DATA_FILE_PATH. The extension is .dat and it is created according to the following rules.

- Data file format
  - (DATA_FILE_PREFIX).(GROUP_NAME)_(FILE_SEQUENCE).dat
    - e.g. CYFILE.GROUP1_1.dat, CYFILE.GROUP1_2.dat

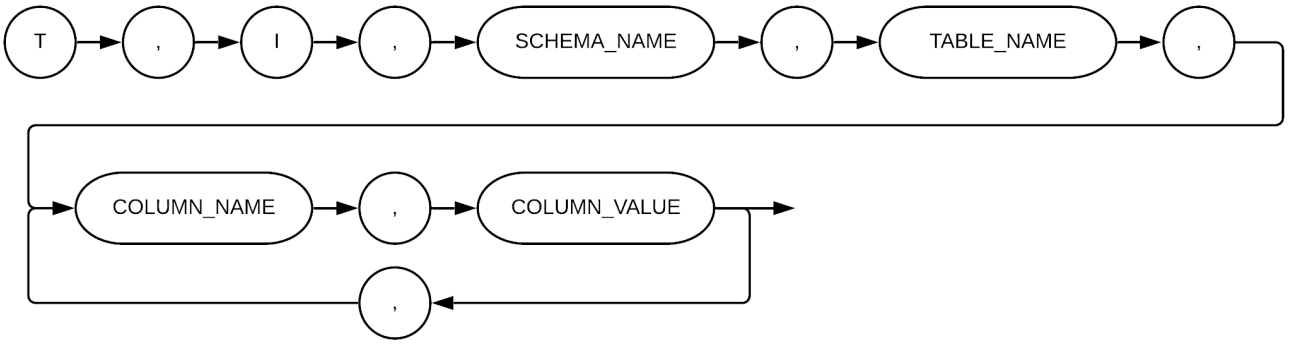
When a new file is created because the data file size becomes bigger than DATA_FILE_SIZE, then the value is determined by adding 1 to FILE_SEQUENCE.

DATA_FILE_PATH, DATA_FILE_PREFIX should not be changed after it starts the operation.

INSERT Storage Format

- In case of INSERT, T standing for table and I standing for insert are described as delimiters, and the sc hema name and the table name of the table in which the insert is performed are described.
- The actual data is a pair of the column name and the column value, and the information about the e ntire column is described in it.

Figure 1 INSERT expression



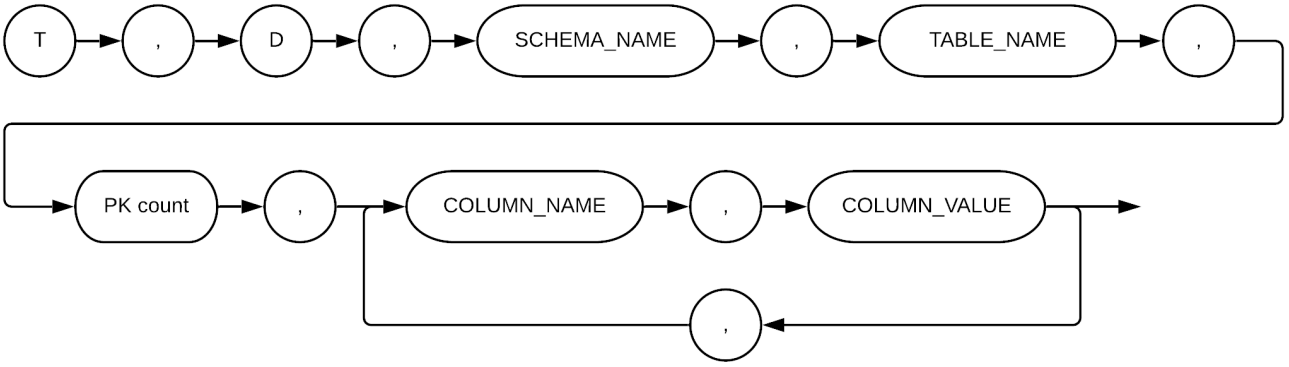

```
Query: INSERT INTO PUBLIC.TEST(C1, C2, C3) VALUES( 1, 2, 'ABC' );
CSV storage: T, I, "PUBLIC", "TEST", "C1", "1", "C2", "2", "C3", "ABC"
```

```
Query: INSERT INTO PUBLIC.TEST(C1, C3) VALUES( 1, ABC );
CSV storage: T, I, "PUBLIC", "TEST", "C1", "1", "C2", NULL, "C3", "ABC"
```

DELETE Storage Format

- In case of DELETE, T standing for table and D standing for delete are described as delimiters, and the schema name and the table name of the table in which the delete is performed are described.
- The primary key information of the deleted data is described, and if it is a composite key, then the pri mary key count becomes 2 or bigger and repeatedly described as many times as the count.

Figure 2 DELETE expression



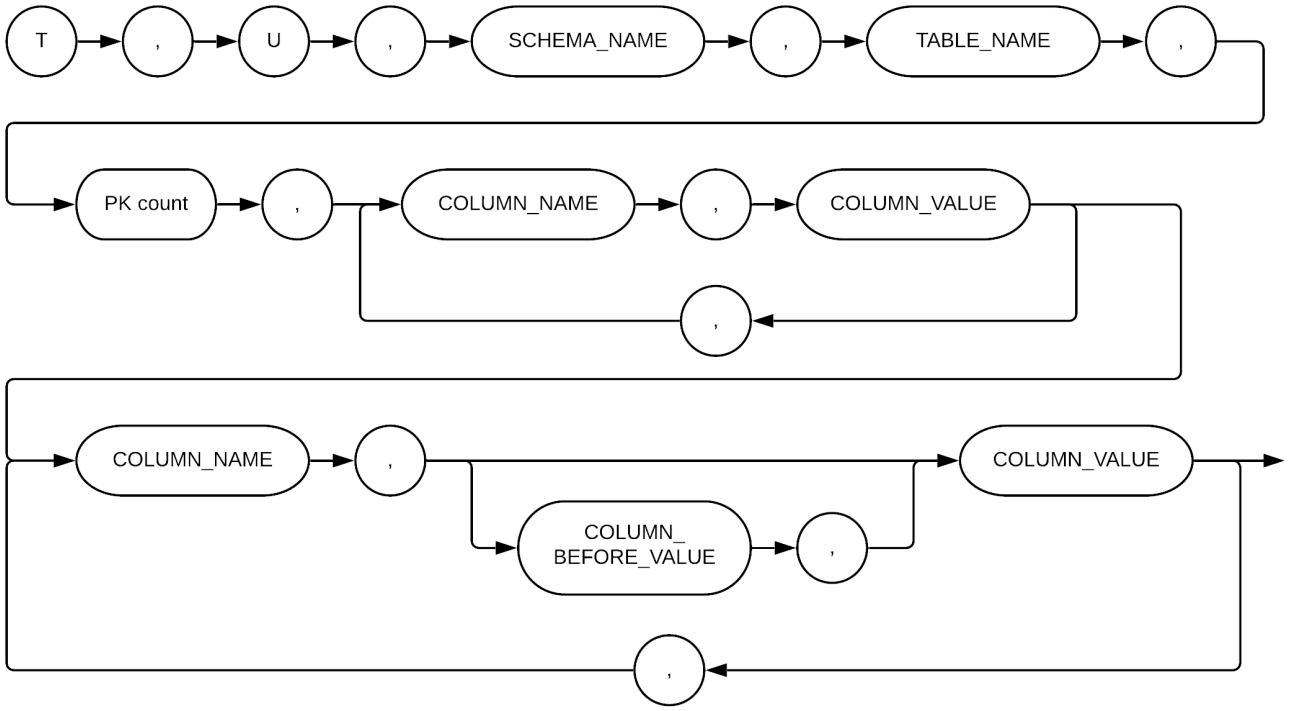
```
Sample: When there is one primary key
Query: DELETE FROM PUBLIC.TEST WHERE C1=1;
CSV storage: T, D, "PUBLIC", "TEST", 1, "C1", "1"
```

```
Sample: When there are two or more primary keys
Query: DELETE FROM PUBLIC.TEST WHERE C1=1 AND C2=2;
CSV storage: T, D, "PUBLIC", "TEST", 2, "C1", "1", "C2", "2"
```

UPDATE Storage Format

- In case of UPDATE, T standing for table and U standing for update are described as delimiters, and th e schema name and the table name of the table in which the update is performed are described.
- The primary key information of the updated data is described, and if it is a composite key, then the pr imary key count becomes 2 or bigger and repeatedly described as many times as the count.
- The information of actually updated column is repeatedly described.
  - When UPDATE_BEFORE_VALUE is set to 1, the value before the update is described together.

Figure 3 UPDATE expression




```
Sample: C2 value in the record whose primary key is only c1 is changed from 1 to 2.
Query: UPDATE PUBLIC.TEST SET C2=2 WHERE C1=1;
* When UPDATE_BEFORE_VALUE is not set
CSV storage: T, U, "PUBLIC", "TEST", 1, "C1", "1", "C2", "2"
* When UPDATE_BEFORE_VALUE is set
CSV storage: T, U, "PUBLIC", "TEST", 1, "C1", "1", "C2", "1", "2"
```

```
Sample: Change C3 value from 'ABC' to 'BCD', and C4 value from 3 to 4 in the record whose primary keys are C1, C2.
Query: UPDATE PUBLIC.TEST SET C3='BCD', C4=4 WHERE C1=1 AND C2=2;
* When UPDATE_BEFORE_VALUE is not set
CSV storage: T, U, "PUBLIC", "TEST", 2, "C1", "1", "C2", "2", "C3", "BCD", "C4", "4"
* When UPDATE_BEFORE_VALUE is set
CSV storage: T, U, "PUBLIC", "TEST", 2, "C1", "1", "C2", "2", "C3", "ABC", "BCD", "C4", "3", "4"
```

Transaction Information

- begin stands for the beginning of the transaction.
  - Additional information: Transaction ID
    - For cluster environment (Global transaction ID)
- Transaction commit
  - Additional information: Commit SCN (System Change Number)
    - For cluster environment (Global SCN)

Figure 4 Transaction begin expression

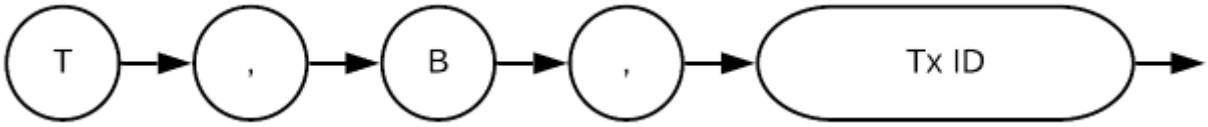


Figure 5 Transaction commit expression

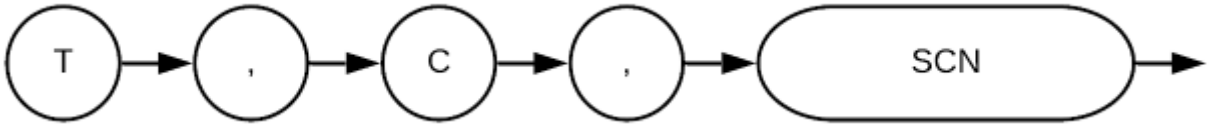


Table Information

- It records the information about the table and column to replicate.
  - The information about N columns per one table information are consecutively recorded.
  - It is repeatedly recorded as may times as the number of tables to replicate.
- It is recorded only once.
  - It is recorded for the first time when starting the program.
  - It is not recorded when restarting the program.
- When resetting or the table is added, then the information is recorded again.
- It records the information about the give up table.
  - The give up table is not captured any more.

- It can make the table to participate in the replication again by using reset option.

Figure 6 Table expression

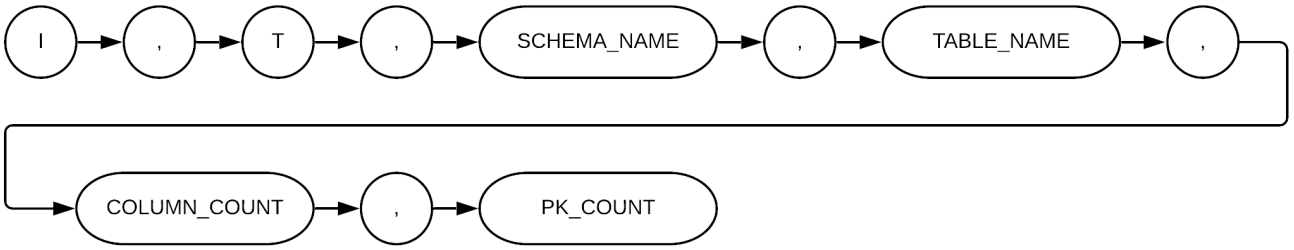
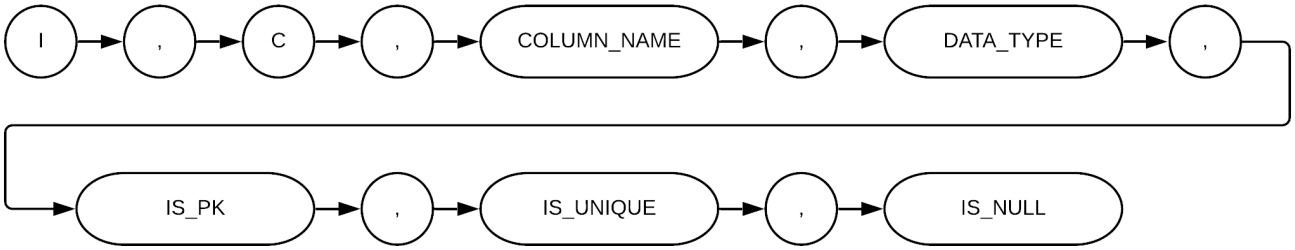


Figure 7 COLUMN expression



- Information about table TEST, TEST2

```
I,T,"PUBLIC","TEST",3,1
I,C,"C1","NUMBER(10,0)",1,0,0
I,C,"C2","VARCHAR(20)",0,0,1
I,C,"C3","VARCHAR(10)",0,0,1
I,T,"PUBLIC","TEST2",2,1
I,C,"C1","NUMBER(10,0)",1,0,0
I,C,"C2","VARCHAR(20)",0,0,1
```

Meta Information

- The following information is recorded at the front line when creating the data file.
  - Version information
    - 8 byte version information
    - It is used to check the compatibility of the data file.
  - Date information
    - The point of when the file is created is recorded.
  - Control file path
    - It records the path of the control file in which the information of the data file is recorded.

Figure 8 Version expression

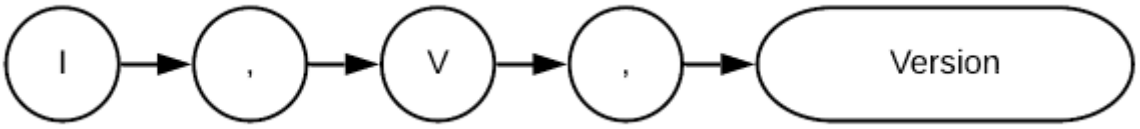


Figure 9 Date information expression

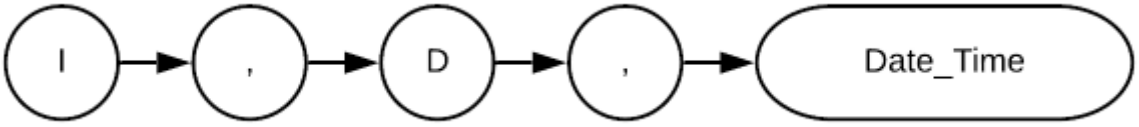
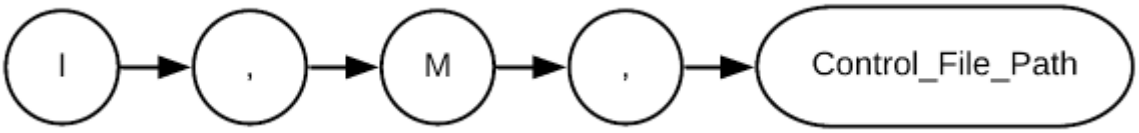


Figure 10 Control file information expression

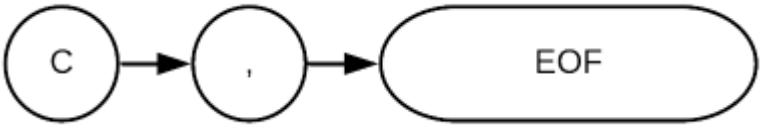



```
I,V, 00000000
I,D,"2022-12-24 00:01:00.000000"
I,M,"/home/cyfile/ctrl_file/cyfile.GROUP1.ctl"
```

Command Information

- EOF is recorded at the end of the existing file when the data file is stored in a new file because its size exceeds the maximum size.
  - Process it by reading the following file when that information is recorded.

Figure 11 EOF expression



Control File

- It records the valid information about the data file.
  - The data file is restarted and restored, so the stored value is not always valid.
  - If the information in the control file is used, then the value is always valid even when it is restarte d and restored.

- The control file size is 512 Bytes.
  - It should be used by offsetting the data or defining the structure.

```
typedef struct ctrlFileStr
{
    char           mVersion[8];
    unsigned int   mFileInfoCrc;   //mFileSeq + mFileOffset CRC
    unsigned int   mDummy;         //Not Used.

    signed long    mFileSeq;
    signed long    mFileOffset;    //Valid Offset
} ctrlFileStr;
```

- mVersion: Checking the compatibility
- mFileInfoCRC: CRC information of mFileSeq and mFileOffset
- mDummy: It is not used.
- mFileSeq: The current data file number
- mFileSeq: The valid location in the data file number including mFileSeq (Offset)

Appendix A.

Error Codes

GOLDILOCKS provides user error codes as follows. The error messages can be viewed with **V$ERROR_CODE**.

A.1 OS Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-1 OS related error

Error code	SQLSTATE	Message
11000	HY000	Invalid argument
11001	HY000	The operation was incomplete although some processing was performed and the results are partially valid
11002	HY000	Permission denied '%s'
11003	HY000	Standard Layer has encountered the end of the file
11004	HY000	Unable to duplicate a file
11005	HY000	File Lock Error
11006	HY000	File Unlock Error
11007	HY000	Unable to open the file '%s'
11008	HY000	Unable to close the file
11009	HY000	Unable to remove the file
11010	HY000	Unable to link the file
11011	HY000	Unable to read the file
11012	HY000	Unable to write the file
11013	HY000	Unable to sync the file
11014	HY000	Unable to seek the file
11015	HY000	Unable to truncate the file
11016	HY000	Unable to get the status of the file
11017	HY000	Unable to create a pipe
11018	HY000	Unable to wait for the given process
11019	HY000	Insufficient resource
11020	HY000	Unable to send signal to given process
11021	HY000	Unable to set a signal handler
11022	HY000	Unable to block a signal
11023	HY000	Unable to set thread attribute
11024	HY000	Unable to create a thread
11025	HY000	Unable to join with a terminated thread
11026	HY000	Unable to set thread once code
11027	HY000	Truncation occurred on I/O operation
11028	HY000	The given name(%s) is too long
11029	HY000	shared memory segment exists
11030	HY000	Unable to allocate a shared memory segment

Error code	SQLSTATE	Message
11031	HY000	Unable to attach the shared memory segment
11032	HY000	Unable to detach the shared memory segment
11033	HY000	Unable to destroy the shared memory segment
11034	HY000	Unable to create a semaphore
11035	HY000	Unable to acquire the given semaphore
11036	HY000	Unable to release the given semaphore
11037	HY000	Unable to destory the given semaphore
11038	HY000	The given object was busy
11039	HY000	Operation timed out
11040	HY000	No such object (%s)
11041	HY000	Two objects do not match
11042	HY000	Not enough memory
11043	HY000	File system is full
11044	HY000	Unable to execute the program
11045	HY000	not supported function
11046	HY000	given buffer does not have enough space
11047	HY000	invalid network address
11048	HY000	sendto() system call returned failure
11049	HY000	recvfrom() system call returned failure
11050	HY000	fail to get address from socket with getsockname()
11051	HY000	fail to get peer address from socket with getpeername()
11052	HY000	fail to manipulate file descriptor with fcntl()
11053	HY000	fail to manipulate socket descriptor with setsockopt()
11054	HY000	not implemented feature
11055	HY000	fail to manipulate device parameters of special file with ioctl()
11056	HY000	fail to get hostname
11057	HY000	fail to close a socket
11058	HY000	fail to shutdown a socket
11059	HY000	fail to create a socket
11060	HY000	fail to get property of a file descriptor
11061	HY000	fail to set property of a file descriptor
11062	HY000	fail to bind a socket to address
11063	HY000	fail to listen on a socket
11064	HY000	fail to accept a connection request with a socket
11065	HY000	system call was interrupted by a signal
11066	HY000	fail to get options of a socket
11067	HY000	fail to connect to an host with a socket
11068	HY000	fail to wait for some file descriptors with poll()
11069	HY000	internal error
11070	HY000	sendmsg() system call returned failure

Error code	SQLSTATE	Message
11071	HY000	recvmsg() system call returned failure
11072	HY000	socketpair() system call returned failure
11073	HY000	converted value is out of range (overflow or underflow)
11074	HY000	given string is not number
11075	HY000	not a valid file desciptor
11076	HY000	resource temporarily unavailable
11077	HY000	given address is already in use
11078	HY000	fail to initialize communication context
11079	HY000	fail to finalize communication context
11080	HY000	fail to connect communication context
11081	HY000	fail to poll communication context
11082	HY000	fail to write packet through communication context
11083	HY000	fail to read packet through communication context
11084	HY000	fail to send descriptor through communication context
11085	HY000	fail to receive descriptor through communication context
11086	HY000	fail to get current file path
11087	HY000	fail to open library (%s)
11088	HY000	fail to close library
11089	HY000	fail to get symbol address (%s)
11090	HY000	not supported query
11091	HY000	fail to check file (%s)
11092	HY000	input is too long
11093	HY000	system call error (function : %s, error no : %d)
11094	HY000	divide zero
11095	HY000	Unable to open a semaphore
11096	HY000	Unable to close the given semaphore
11097	HY000	Unable to unlink the given semaphore
11098	HY000	permission denied - '%s'
11099	HY000	not supported os
11100	HY000	Environment Variable "%s" is not defined.
11101	HY000	License file(%s) does not exist
11102	HY000	License is out of date. License is valid after %4d-%02d-%02d
11103	HY000	License date is expired. License is expired in %4d-%02d-%02d
11104	HY000	License core count mismatch.
11105	HY000	License key is corrupted.
11106	HY000	License host name mismatch.
11107	HY000	failed to set thread affinity.
11108	HY000	failed to create timer.
11109	HY000	failed to set timer.
11110	HY000	failed to destroy timer.

Error code	SQLSTATE	Message
11111	HY000	address is not given.
11112	HY000	Unable to access the message queue.
11113	HY000	A message queue exists.
11114	HY000	Unable to create a message queue.
11115	HY000	The message queue was removed.
11116	HY000	Unable to destroy the message queue.
11117	HY000	Unable to send message.
11118	HY000	Unable to receive message.
11119	HY000	No message was available in the message queue.
11120	HY000	fail to get property of a file status
11121	HY000	fail to set property of a file status
11122	HY000	infiniband ibv/rdma interface returned error
11123	HY000	no infiniband device was found
11124	HY000	cannot find specified infiniband device
11125	HY000	cannot find suitable Memory Region for specified ptr and size
11126	HY000	fail to get semaphore value
11127	HY000	fail to control pollset
11128	HY000	fail to create pollset
11129	HY000	overflow fd
11130	HY000	invalid INI file format
11131	HY000	path '%s' does not exist
11132	HY000	Unable to get maximum undo semaphores per array
11133	HY000	Unable to create a undo semaphore
11134	HY000	Unable to bind the given undo semaphore
11135	HY000	Unable to unbind the given undo semaphore
11136	HY000	Unable to get the given undo semaphore state
11137	HY000	Unable to destroy the given undo semaphore
11138	HY000	Unable to suspend asyncronous io
11139	HY000	Unable to control the shared memory segment
11140	HY000	Unable to detach a thread
11141	08S01	Communication link failure
11142	HY000	No such process
11143	HY000	getaddrinfo() system call returned failure
11144	HY000	address is unavailable
11145	HY000	semaphore set exists
11146	HY000	unable to create a semaphore set
11147	HY000	unable to return a semaphore set
11148	HY000	unable to set value a semnum-th semaphore
11149	HY000	unable to performs operations on selected semaphores
11150	HY000	unable to remove a semaphore set

A.2 Datatype and Operation Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-2 Datatype and operation related error

Error code	SQLSTATE	Message
12000	HY000	failed to initialize datatype layer
12001	HYC00	not implemented feature, in a function %s
12002	22001	byte length of data greater than column length
12003	22001	character length of data greater than column length
12004	22001	data converted with truncation of fractional digits
12005	22001	conversion of data would result in loss of whole digits
12006	22018	data value is not a numeric literal
12007	22003	data is outside the range of the data type to which the number is being converted
12008	22001	data is greater than 0, less than 2, and not equal to 1
12009	22003	data is less than 0 or greater than or equal to 2
12010	22001	(byte length of data)/2 is greater than column byte length
12011	22003	data value is not a hexadecimal value
12012	22008	data value is a valid timestamp literal; time portion is nonzero
12013	22018	data value is not a valid date literal or timestamp literal
12014	22008	data value is a valid timestamp literal; fractional seconds portion is nonzero
12015	22018	data value is not a valid time literal or timestamp literal
12016	22008	data value is a valid timestamp literal; fractional seconds portion truncated
12017	22018	data value is not a valid date literal or time literal or timestamp literal
12018	22015	data value is a valid interval value; the value in one of the fields is truncated
12019	22018	data value is not a valid interval literal
12020	22001	number of digits greater than column byte length
12021	22001	number of characters greater than column byte length
12022	22003	data converted with truncation of whole digits
12023	22003	data is outside the range of the data type to which the number is being converted
12024	22001	data is greater than 0, less than 2, and not equal to 1
12025	22003	data is less than 0 or greater than or equal to 2
12026	22015	interval data truncated
12027	22001	byte length of data greater than column length
12028	22001	character length of data greater than column length
12029	22003	byte length of data not equal to SQL data length

Error code	SQLSTATE	Message
12030	22001	column byte length is less than %d
12031	22008	data value is not a valid date
12032	22007	data value is not a valid date format
12033	22001	column byte length is less than %d
12034	22018	data value is not a valid GUID
12035	22001	column byte length is less than %d
12036	22001	data value is not a valid time
12037	22007	data value is not a valid time format
12038	22001	column byte length is between %d and character byte length
12039	22001	column byte length is less than %d
12040	22008	data value is not a valid timestamp
12041	22008	time fields are nonzero
12042	22007	data value does not contain a valid date
12043	22008	fractional seconds fields are nonzero
12044	22007	data value does not contain a valid time
12045	22008	fractional seconds fields are truncated
12046	22007	data value does not contain a valid timestamp
12047	22001	column byte length is less than character byte length
12048	22015	data value is not a valid interval literal
12049	22003	conversion resulted in truncation of whole digits
12050	22015	one or more fields of data value were truncated during conversion
12051	22001	column byte length is less than character byte length
12052	22015	data value is not a valid interval literal
12053	22003	conversion resulted in truncation of whole digits
12054	22015	one or more fields of data value were truncated during conversion
12055	01004	byte length of data greater equal than buffer length
12056	01004	character length of data greater equal than buffer length
12057	01S07	data converted with truncation of fractional digits
12058	22003	conversion of data would result in loss of whole digits
12059	22018	data is not numeric literal
12060	22003	data is outside the range of the data type to which the number is being convert ed
12061	01S07	data is greater than 0, less than 2, and not equal to 1
12062	22003	data is less than 0 or greater than or equal to 2
12063	01S07	data value is a valid timestamp-value; time portion is nonzero
12064	22018	data value is not a valid date-value or timestamp-value
12065	01S07	data value is a valid timestamp-value; fractional seconds portion is nonzero
12066	22018	data value is not a valid time-value or timestamp-value
12067	22018	data value is not a valid date-value or time-value or timestamp-value
12068	01S07	data value is a valid interval value;truncation of one or more trailing fields

Error code	SQLSTATE	Message
12069	22015	data value is a valid interval; leading field significant precision is lost
12070	22018	data value is not a valid interval value
12071	01004	number of whole digits is less than buffer length
12072	22003	number of whole digits is greater than buffer length
12073	01S07	data converted with truncation of fractional digits
12074	22003	conversion of data would result in loss of whole digits
12075	22003	data is outside the range of the data type to which the number is being converted
12076	01S07	data is greater than 0, less than 2, and not equal to 1
12077	22003	data is less than 0 or greater than or equal to 2
12078	22003	byte length of data is greater than buffer length
12079	01S07	fractional seconds portion truncated
12080	22015	whole part of number truncated
12081	22003	buffer length is less equal than 1
12082	22003	buffer length is less than 1
12083	01004	(byte length of data)/2 is greater equal than buffer length
12084	01004	(character length of data)/2 is greater equal than buffer length
12085	01004	byte length of data is greater than buffer length
12086	01004	buffer length is between %d and character byte length
12087	22003	buffer length is less than %d
12088	22003	byte length of data is greater than buffer length
12089	22003	buffer length is less than %d
12090	22003	byte length of data is greater than buffer length
12091	01004	buffer length is between %d and character byte length
12092	22003	buffer length is less than %d
12093	22003	byte length of data is greater than buffer length
12094	01004	buffer length is between %d and character byte length
12095	22003	buffer length is less than %d
12096	01004	buffer length is between %d and character length
12097	22003	byte length of data is greater than buffer length
12098	01S07	time portion of timestamp is nonzero
12099	01S07	fractional seconds portion of timestamp is nonzero
12100	01S07	fractional seconds portion of timestamp is truncated
12101	01S07	trailing fields portion truncated
12102	22015	leading precision of target is not big enough to hold data from source
12103	22003	interval precision was a single field and truncated whole
12104	22015	interval precision was not a single field
12105	22003	byte length of data is greater than buffer length
12106	01004	number of whole digits is less than buffer length
12107	22003	number of whole digits is greater equal than buffer length

Error code	SQLSTATE	Message
12108	01S07	trailing fields portion truncated
12109	22015	leading precision of target is not big enough to hold data from source
12110	01S07	interval precision was a single field and truncated fractional
12111	22003	interval precision was a single field and truncated whole
12112	07006	interval precision was not a single field
12113	22003	byte length of data is greater than buffer length
12114	01004	number of whole digits is less than buffer length
12115	22003	number of whole digits is greater equal than buffer length
12116	22009	invalid time zone displacement value
12117	42000	invalid number of arguments
12118	42000	inconsistent datatype of function parameters: %s
12119	42000	comparison is not applicable: %s and %s
12120	42000	conversion is not applicable: from %s to %s
12121	22021	invalid character value in characterset repertoire
12122	22012	divisor is equal to zero
12123	22018	data is not boolean literal
12124	HY000	invalid argument for factorial function
12125	22027	trim set should have only one character
12126	22008	datetime field overflow
12127	HY000	invalid ROWID
12128	22019	invalid escape character
12129	HY000	invalid character set identifier
12130	HY000	invalid extract field for extract source
12131	42000	inconsistent datatypes: argument[%d] expected %s
12132	HY000	date format value exceeds maximum length (%d)
12133	HY000	date format not recognized
12134	HY000	format code appears twice
12135	HY000	%s conflicts with %s
12136	HY000	literal does not match format string
12137	HY000	input value not long enough for date format
12138	HY000	date format picture ends before converting entire input string
12139	HY000	year must be between %s and %s, and not be 0
12140	2200C	missing or illegal character following the escape character
12141	HY000	not a valid month
12142	HY000	a non-numeric character was found where a numeric was expected
12143	HY000	day of month must be between 1 and last day of month
12144	HY000	not a valid day of the week
12145	HY000	day of year must be between 1 and 365 (366 for leap year)
12146	HY000	format code cannot appear in date input format
12147	HY000	BC/B.C. or AD/A.D. required

Error code	SQLSTATE	Message
12148	HY000	julian date must be between 0 and 5373484
12149	HY000	%s may only be specified once
12150	HY000	Julian date precludes use of day of year
12151	HY000	Not enough format info buffer
12152	HY000	hour must be between %d and %d
12153	HY000	minute must be between 0 and 59
12154	HY000	second must be between 0 and 59
12155	HY000	the fractional seconds must be between 0 and 999999
12156	HY000	time zone hour must be between -14 and 14
12157	HY000	time zone minute must be between 0 and 59
12158	HY000	seconds in day must be between 0 and 86399
12159	HY000	AM/A.M. or PM/P.M. required
12160	HY000	HH24 precludes use of meridian indicator
12161	HY000	%s conflicts with use of %s
12162	HY000	invalid XID string
12163	HY000	date not valid for month specified
12164	HY000	signed year precludes use of BC/AD
12165	HY000	number format value exceeds maximum length (%d)
12166	HY000	invalid number format model
12167	HY000	cannot use %s twice
12168	HY000	cannot use %s and %s together
12169	22001	string data, right truncation
12170	HY000	argument '%d' is out of range
12171	HY000	invalid datepart for data type %s
12172	42000	maximum number of arguments exceeded
12173	42000	invalid char length units(%s): use OCTETS or CHARACTERS
12174	22023	The argument of %s function is invalid
12175	22023	The argument [%d] of %s function is invalid
12176	HY000	internal error, in a function %s
12177	HY000	not supported data type (%s)
12178	HY000	data type specifier is out of range (type : %d, range : %d to %d)
12179	HY000	precision specifier is out of range(type : %s, precision : %d, range : %d to %d)
12180	HY000	leading precision specifier is out of range(type : %s, precision : %d, range : %d to %d)
12181	HY000	scale specifier is out of range(type : %s, scale : %d, range : %d to %d)
12182	HY000	fractional second precision specifier is out of range(type : %s, precision : %d, range : %d to %d)
12183	HY000	character length unit specifier is out of range(type : %s, character length unit : %d, range : %d to %d)

Error code	SQLSTATE	Message
12184	HY000	interval indicator specifier is out of range(type : %s, indicator : %d, range : %d to %d)
12185	42000	%s and %s cannot be matched
12186	0A000	not supported function %s
12187	39000	failed to open customer built-in function library (%s)
12188	42000	window frame offset must not be null
12189	42000	window frame offset must not be negative

A.3 Resource Management Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-3 Resource management related error

Error code	SQLSTATE	Message
13000	HY000	Failed to initialize kernel layer
13001	HY000	kernel layer is not initialized yet
13002	HY000	Invalid argument
13003	HY000	Invalid shared memory segment type
13004	HY000	static shared memory segment is corrupted
13005	HY000	maximum number of shared memory segment num exceeded
13006	HY000	kernel layer internal error - %s
13007	42000	identifier is too long
13008	HY000	comment is too long
13009	42S02	fixed table does not exist
13010	HY000	Insufficient static area
13011	HY000	Unable to extend memory: [MAX: %ld, TOTAL: %ld, ALLOC: %ld] DESC: %s
13012	HY000	too many files are opened
13013	HY000	Property(%s) does not exist
13014	HY000	Unable to change property at this startup phase.(%s)
13015	0U000	Unable to change read-only property.(%s)
13016	22000	Invalid property value.(%s)
13017	0U000	Specified property cannot be modified.(%s)
13018	22000	Specified property cannot be modified with this SCOPE option.(%s)
13019	22000	Specified property cannot be modified.(%s)
13020	22000	Invalid GOLDILOCKS_DATA Directory
13021	22000	Invalid SID
13022	HY004	Invalid property data type
13023	22000	Invalid Property ID
13024	HY000	exceeded maximum number of processes
13025	HY000	File does not exist (%s)
13026	HY000	Property file already exists (%s)
13027	HY000	File Process Error
13028	HY000	Property is not ReadOnly
13029	HY000	Read-only property file already exists
13030	HY000	Property File Parsing Error Line (%d)
13031	0A000	Unsupported return type is used

Error code	SQLSTATE	Message
13032	0A000	Function does not exist
13033	HY000	It can not make range
13034	08S01	Service is not available
13035	HY000	Specified Breakpoint name is not registered yet
13036	HY000	General abort error of Breakpoint
13037	HY000	Invalid Session Name
13038	HYT00	Exceeded maximum idle time
13039	HYT00	Exceeded maximum query time
13040	08S01	Communication link failure
13041	RD000	User session ID does not exist
13042	08004	Server rejected the connection
13043	HY008	operation canceled
13044	RD000	invalid process identifier
13045	08004	mismatched binary version - server(%s.%d.%d.%d.%d), client(%s.%d.%d.%d.%d)
13046	HY000	session cannot be killed now
13047	22003	data is outside the range of the data type to which the number is being converted
13048	22015	one or more fields of data value were truncated during conversion
13049	HYC00	not implemented feature, in a function
13050	HY000	insufficient plan cache memory
13051	HY000	Unable to attache the shared memory segment. File does not exist(%s)
13052	42000	invalid dump option string: %s
13053	42000	dump object modified by concurrent statement
13054	42000	invalid numa map (%s)
13055	HY000	AS ADMIN must be executed in dedicated session
13056	HY000	unable to create file '%s' - already in use
13057	HY000	Property(%s) is desupported
13058	HY000	memory alloc size is too large: [MAX: %ld, ALLOC: %ld] DESC: (%d)%s
13059	HYT00	Exceeded maximum packet allocation time
13060	HY000	Failed to validate the license key

A.4 Storage Management Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-4 Storage management related error

Error code	SQLSTATE	Message
14000	HY000	Failed to initialize storage manager layer
14001	HY000	storage manager layer is not initialized yet
14002	HY000	Tablespace(%d) exceed datafile limit
14003	HY000	Tablespace(%d) is full
14004	HY000	Tablespace(%d) is already dropped
14005	HY000	System Tablespace exists
14006	42000	Table is mutating, trigger/function may not see it
14007	RD000	deadlock detected
14008	24000	fetch out of sequence
14009	2200H	sequence generator limit exceeded
14010	22023	MAXVALUE cannot be made to be less than the current value
14011	22023	MINVALUE cannot be made to exceed the current value
14012	22023	cannot be made to be less than the current value
14013	22023	cannot be made to exceed the current value
14014	HY000	Fail to extend datafile(%u) of tablespace(%u) in Transaction(%ld)
14015	HY000	there is no extendible datafile in tablespace '%s'
14016	23001	some rows of base table violate uniqueness of index
14017	42000	invalid operation on fixed table or dump table
14018	42000	the operation is disallowed before %s startup phase
14019	HY000	log stream number string is invalid
14020	HY000	log stream %d already exists
14021	HY000	log group number string is invalid
14022	HY000	log group state must be INACTIVE or UNUSED
14023	HY000	log group %d already exists
14024	HY000	log member %s already exists
14025	42000	invalid dump option string: %s
14026	HYT00	resource busy or timeout expired
14027	HY000	datafile(%d,%d) is already dropped
14028	HY000	file '%s' already exists
14029	HY000	sequence is not yet defined
14030	RD000	maximum index depth(%d) exceeded
14031	HY000	maximum key length(%d) exceeds

Error code	SQLSTATE	Message
14032	42000	can't serialize access for this transaction
14033	42000	unable to checkpoint in cds mode
14034	42000	unable to recover database in cds mode
14035	42000	some rows of base table violate not null constraint
14036	42000	invalid ROWID
14037	42000	Database is READ ONLY - deprecated
14038	42000	unable to recover database in READ ONLY mode - deprecated
14039	HY000	exceed segment MAXSIZE(%ld)
14040	HY000	Target object of dump table is mutating
14041	42000	cannot access the OFFLINE tablespace '%s'
14042	42000	cannot find the row (%ld)
14043	42000	file is already used in database '%s'
14044	42000	datafile not empty
14045	42000	cannot drop datafile of OFFLINE tablespace '%s'
14046	42000	file does not exist - '%s'
14047	42000	file is READ ONLY - '%s'
14048	42000	file is invalid or old version - '%s'
14049	42000	tablespace is not online - '%s'
14050	42000	tablespace is not offline - '%s'
14051	42000	media recovery required - '%s'
14052	42000	segment is unusable (physical id : %ld)
14053	RD000	insufficient resource for sort
14054	HY000	internal error (%s, %d)
14055	HY000	segment is corrupted (physical id : %ld)
14056	HY000	failed to apply redo log (lsn : %ld, lpsn : %d, physical id : %ld)
14057	HY000	dropping log group makes total log group count less than minimum log group count(%d)
14058	42000	log member does not exist - '%s'
14059	42000	log member is already used - '%s'
14060	HY000	CURRENT logfile is empty - '%s'
14061	HY000	next logfile to switch should be INACTIVE or UNUSED - '%s'
14062	42000	INITRANS must be less equal than MAXTRANS
14063	42000	cannot be added log member to exceed maximum log member count(%d)
14064	42000	the operation is disallowed at %s phase
14065	RD000	the instant table row length is too large
14066	RD000	the sort instant table key row length is too large
14067	HY000	cannot media recover; archivelog mode not enabled
14068	HY000	logfile does not exist - '%s'
14069	HY000	cannot support ARCHIVELOG in cds mode
		control file is not recent; do media recovery with USING BACKUP CONTROLFILE

Error code	SQLSTATE	Message
14070	HY000	option
14071	HY000	cannot BACKUP; tablespace(%s) is offline
14072	HY000	cannot BACKUP; tablespace(%s) is temporary
14073	HY000	cannot BACKUP; tablespace is dropped
14074	HY000	cannot BACKUP; tablespace(%s) is already in backup
14075	HY000	cannot execute; tablespace(%s) is proceeding backup
14076	HY000	cannot END BACKUP; tablespace(%s) is not in backup
14077	HY000	cannot END BACKUP; none of the tablespaces are in backup
14078	HY000	cannot BACKUP; tablespace(%s) is proceeding other operation '%s'
14079	HY000	cannot BACKUP; database is already in backup
14080	HY000	cannot END BACKUP; database is not in backup
14081	HY000	cannot BACKUP; database is going shutdown
14082	HY000	(%s) is not an appropriate logfile to recover
14083	HY000	must use RESETLOGS option for database open
14084	HY000	must use NORESETLOGS option for database open
14085	HY000	(%s) was not a sufficiently old backup
14086	HY000	media recovery canceled
14087	HY000	cannot SHUTDOWN; backup in progress
14088	HY000	control file is corrupted
14089	HY000	LEVEL 0 INCREMENTAL BACKUP does not exist
14090	HY000	cannot flush logs; disabled log flushing
14091	HY000	(%s) is not valid logfile
14092	HY000	not exist incremental backup
14093	HY000	not exist valid incremental backup for tablespace(%s)
14094	HY000	datafile recovery required - datafile(%s) of tablespace(%s) corrupted
14095	HY000	backup file(%s) is corrupted
14096	23001	fail to build index (%ld)
14097	HY000	control file is corrupted - '%s'
14098	HY000	maximum record length(%d) exceeds
14099	HY000	log member state must be UNUSED or INACTIVE
14100	HY000	dropping log member makes log member count of group(%d) less than minimum log member count(%d)
14101	HY000	log group id does not exist - '%d'
14102	HY000	cannot execute; recovery is in progress
14103	01000	Warning: media recovery needs a logfile including log (Lsn %ld)
14104	01000	Warning: suggestion '%s'
14105	HY000	control file '%s' is inconsistent with primary
14106	HY000	tablespace (%s) is taken offline as the result of a write error
14107	HY000	given LOG_BLOCK_SIZE(%d) is not a suitable value for a size of log block; LOG_BLOCK_SIZE should be one of 512, 1024, 2048 or 4096.

Error code	SQLSTATE	Message
14108	HY000	cannot disable archivelog - exist tablespace needed media recovery.
14109	HY000	OFFLINE NORMAL disallowed; cannot be taken offline consistently.
14110	HY000	OFFLINE IMMEDIATE disallowed; noarchivelog mode
14111	HY000	cannot create COLUMNAR TABLE; given MIN ROW COUNT is too big to put int o one page
14112	HY000	record size is too large for columnar table
14113	HY000	datafile '%s' is corrupted
14114	HY000	datafile '%s' is more recent than redo logfile
14115	HY000	transaction undo limit exceeded
14116	HY000	failed to journal log
14117	HY000	cannot end incomplete recovery; incomplete recovery never been begun
14118	HY000	property TRANSACTION_TABLE_SIZE value must be equal to or greater than '% d'
14119	HY000	property UNDO_RELATION_COUNT value must be equal to or greater than '%ld '
14120	HY000	sequence is dropped
14121	HY000	file '%s' does not match with database - file signature '%s', database signature '%s'
14122	HY000	there is a possibility that a deadlock occurs
14123	22023	INCREMENTBY sign cannot be changed in the cluster system
14124	HY000	RESTORE in progress
14125	HY000	exceed maximum journal file size
14126	HY000	there must be at least one recoverable member in the group
14127	HY000	object no longer exists
14128	HY000	invalid remote view scn
14129	HYT00	transaction allocation time exceeded
14130	HYT00	undo relation allocation time exceeded
14131	42000	unable to enable change tracking in cds mode
14132	42000	unable to enable change tracking in noarchivelog mode
14133	HY000	change tracking is already enabled
14134	HY000	change tracking is not enabled
14135	HY000	change tracking file is corrupted - '%s'
14136	HY000	failed to backup incremental because change tracking is being altered
14137	HY000	failed to disable change tracking because incremental backup is in progress
14138	HY000	exceed maximum insertable records (%lu) by transaction
14139	HY000	transaction(%ld) is timed out for waiting to resolve cluster deadlock
14140	HY000	the value of MAXTRANS to be changed must be equal to or greater than the cu rrent MAXTRANS '%d'
14141	HY000	shard '%d' exceeds maximum shard sequence
14142	HY000	cannot recover; database is going shutdown

Error code	SQLSTATE	Message
14143	HY000	cannot backup; database is proceeding other operation '%s'
14144	HY000	file is not in use for global transaction log - '%s'
14145	HY000	file is already in use for global transaction log - '%s'
14146	01000	shard(%d) of index(%ld) used all shard sequences
14147	HY000	incomplete recovery is possible up to '%s'

A.5 Dictionary Cash Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-5 Dictionary cash related error

Error code	SQLSTATE	Message
15000	HY000	failed to initialize execution library layer
15001	HY000	execution library layer is not initialized yet
15002	42000	%s: schema does not exist
15003	42000	%s: table or view does not exist
15004	42000	%s: index does not exist
15005	42000	%s: column does not exist
15006	23000	%s: dictionary integrity constraint violation by concurrent DDL execution
15007	HY090	not enough stack space to add entries
15008	HY000	table related objects are modified while validation
15009	HY000	Invalid argument
15010	42000	object modified by concurrent DDL execution
15011	HYC00	not implemented feature, in a function %s
15012	42000	user dropped by other session
15013	42000	user default schema dropped
15014	42000	user default data tablespace dropped
15015	42000	user default temporary tablespace dropped
15016	0Y002	limit(%d) on the number of named cursor exceeded
15017	42000	DDL not allowed for supplemental log table
15018	42000	cluster member '%s' has already joined in an other cluster system
15019	42000	the %s of cluster member '%s' is not compatible with the cluster system
15020	42000	tablespace '%s' of the cluster system does not exist at cluster member '%s'
15021	42000	tablespace '%s' of the cluster member '%s' does not exist at the cluster system
15022	42000	the %s of the tablespace '%s' at cluster member '%s' is not compatible with th e cluster system
15023	HY000	limit(%d) on the number of named instant table exceeded
15024	42000	the cluster member '%s' does not join in a cluster system
15025	42000	startup phase of new cluster member should be OPEN
15026	HY000	internal error, in a function %s
15027	42000	attempt to create, alter or drop an index on temporary table already in use
15028	42000	user default index tablespace dropped
15029	2F000	limit(%d) on the number of package instance exceeded
15030	42000	role dropped by other session

A.6 SQL Handling Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-6 SQL handling related error (16000~16099)

Error code	SQLSTATE	Message
16000	HY000	failed to initialize sql processor layer
16001	HYC00	not implemented feature, in a function %s
16002	HY000	invalid function sequence of sql processor layer
16003	42000	identifier is too long
16004	28000	invalid username/password; logon denied
16005	42000	name '%s.%s' is already used by an existing object
16006	42000	schema '%s' does not exist
16007	42000	schema '%s' is not writable
16008	42000	tablespace '%s' does not exist
16009	42000	tablespace '%s' is not writable
16010	42000	invalid value for PCTFREE or PCTUSED
16011	42000	invalid INITRANS option value
16012	42000	duplicate column name
16013	HY000	invalid value is prevented in validation step
16014	22023	INCREMENT must be a non-zero integer
16015	22023	START WITH cannot be less than MINVALUE
16016	22023	START WITH cannot be more than MAXVALUE
16017	22023	the number of values to CACHE must be greater than 1
16018	22023	number to CACHE must be less than one cycle
16019	22023	MINVALUE must be less than MAXVALUE
16020	22023	INCREMENT must be less than MAXVALUE minus MINVALUE
16021	22023	descending sequences that CYCLE must specify MINVALUE
16022	22023	ascending sequences that CYCLE must specify MAXVALUE
16023	22023	duplicate or conflicting CACHE/NO CACHE specifications
16024	42000	numeric precision specifier is out of range(%d to %d)
16025	42000	numeric scale specifier is out of range(%d to %d)
16026	42000	character length specifier is out of range(1 to %d)
16027	42000	character varying length specifier is out of range(1 to %d)
16028	42000	binary length specifier is out of range(1 to %d)
16029	42000	binary varying length specifier is out of range(1 to %d)
16030	42000	name already used by an existing constraint
		auto-generated name already used by an existing constraint, specify unique con

Error code	SQLSTATE	Message
16031	42000	straint name
16032	42000	table can have only one primary key
16033	42000	such unique or primary key already exists in the table
16034	42000	name already used by an existing index
16035	42000	auto-generated name already used by an existing index, specify non-duplicate index name
16036	42000	'%s': invalid identifier
16037	42000	maximum number of key columns exceeded
16038	23000	integrity constraint violation
16039	3B000	savepoint '%s' never established in this session or is invalid
16040	42000	table or view does not exist
16041	42000	table '%s' is not writable
16042	42000	unique/primary keys in table referenced by foreign keys
16043	42000	table referenced by views
16044	42000	sequence does not exist
16045	42000	duplicate options
16046	42000	such column list already indexed
16047	22003	numeric value out of range
16048	42000	specified index does not exist
16049	42000	index '%s' is not writable
16050	42000	cannot modify index used for enforcement of unique/primary/foreign key
16051	HY000	not applicable
16052	42000	column not allowed here
16053	42000	not enough values
16054	42000	too many values
16055	22004	NULL value not allowed
16056	22004	cannot insert NULL into "%s"."%s"."%s"
16057	23000	unique constraint (%s.%s) violated
16058	42000	not applicable hint
16059	42000	not applicable datatype
16060	07002	bind variable does not exist
16061	22004	cannot update ("%s"."%s"."%s") to NULL
16062	42000	syntax error
16063	42000	fractional seconds precision specifier is out of range(%d to %d)
16064	42000	invalid interval indicator
16065	42000	interval leading precision specifier is out of range(%d to %d)
16066	42000	interval fractional seconds precision specifier is out of range(%d to %d)
16067	42000	invalid INITIAL storage option value
16068	42000	invalid NEXT storage option value
16069	42000	invalid MINSIZE storage option value

Error code	SQLSTATE	Message
16070	42000	invalid MAXSIZE storage option value
16071	42000	storage option MAXSIZE is less than MINSIZE
16072	42000	duplicate or conflicting NULL and/or NOT NULL specifications
16073	42000	duplicate UNIQUE/PRIMARY KEY specifications
16074	42000	sequence number not allowed here
16075	42000	inconsistent datatypes : expected %s
16076	42000	invalid DATAFILE clause for alter of %s %s TABLESPACE
16077	42000	invalid DATAFILE clause for alter of %s %s TABLESPACE
16078	42000	invalid TEMPFILE clause for alter of %s %s TABLESPACE
16079	42000	file path is too long (< %d)
16080	RD000	file size is out of range(%ld to %ld)
16081	42000	failed to open database : system threads are already exist.
16082	42000	invalid MAXTRANS option value
16083	42000	INITRANS must be less equal than MAXTRANS
16084	42000	result OFFSET must be 0 or positive numeric value
16085	42000	result LIMIT/FETCH must be positive numeric value
16086	42000	related objects are modified by concurrent DDL execution
16087	42000	boolean expression expected on where clause
16088	HY000	session cannot be killed now
16089	28000	built-in authorization(user or role) '%s' is not modifiable
16090	42000	built-in tablespace '%s' is not modifiable
16091	42000	built-in schema '%s' is not modifiable
16092	42000	built-in table or view '%s' is not modifiable
16093	42000	built-in index '%s' is not modifiable
16094	42000	built-in constraint '%s' is not modifiable
16095	42000	built-in sequence '%s' is not modifiable
16096	28000	authorization(user or role) '%s' does not exist
16097	42000	constraint '%s' does not exist
16098	07006	bind type mismatch of parameter number (%d)
16099	42000	the number of elements in the INTO list is mismatch

Table A-7 SQL handling related error (16100~16199)

Error code	SQLSTATE	Message
16100	HY000	single-row subquery returns more than one row
16101	42000	parallel factor is out of range(0 to %d)
16102	42000	invalid MEMORY clause for alter of %s %s TABLESPACE
16103	3C000	cursor '%s' is already declared
16104	42000	invalid cursor query: '%s'
16105	42000	invalid cursor property

Error code	SQLSTATE	Message
16106	34000	cursor '%s' does not exist
16107	34000	cursor '%s' is already open
16108	34000	cursor '%s' is not open
16109	42000	invalid use: cursor '%s' is not a declared cursor
16110	42000	invalid use: cursor '%s' is not an ODBC/JDBC cursor
16111	42000	cursor '%s' is not scrollable
16112	42000	query expression is not updatable
16113	42000	invalid phase(%s): executable phase of '%s' is between %s and %s
16114	42000	database is already mounted
16115	42000	database is already local opened
16116	42000	transaction is READ ONLY
16117	42000	database not mounted
16118	42000	supplemental logging attribute primary key exists
16119	42000	cannot drop nonexistent primary key supplemental logging
16120	42000	column referenced by views
16121	42000	column has a check constraint defined
16122	42000	pseudo column not allowed here
16123	42000	cannot select ROWID from not writable table
16124	42000	table can have only one identity column
16125	42000	column being added already exists in table
16126	42000	table must be empty to add mandatory (%s) column ('%s')
16127	42000	cannot modify INITIAL storage option
16128	42000	name '%s' is already used by an existing tablespace
16129	42000	duplicate datafile name
16130	42000	file already exists - '%s'
16131	42000	cannot rename the built-in tablespace
16132	42000	cannot rename the OFFLINE tablespace
16133	42000	cannot drop tablespace: %s
16134	42000	cannot drop constraint - nonexistent constraint
16135	42000	cannot drop constraint - nonexistent primary key
16136	42000	cannot drop constraint - nonexistent unique key
16137	42000	the CDS database cannot be closed with ABORT option
16138	42000	the DS database can be closed with NORMAL option - deprecated
16139	42000	multiple ORDER BY clauses not allowed
16140	42000	multiple OFFSET clauses not allowed
16141	42000	multiple LIMIT clauses not allowed
16142	42000	column ambiguously defined
16143	42000	column part of USING clause cannot have qualifier
16144	42000	invalid OFFSET option value
16145	42000	invalid LIMIT option value

Error code	SQLSTATE	Message
16146	42000	common column name '%s' appears more than once in %s table
16147	42000	file is already used in database
16148	42000	tablespace not empty, use INCLUDING CONTENTS option
16149	42000	datafile does not exist in tablespace
16150	42000	cannot drop datafile of OFFLINE tablespace
16151	42000	cannot drop first datafile of tablespace
16152	42000	memory is already used in database
16153	42000	memory does not exist in tablespace
16154	42000	cannot drop first memory of tablespace
16155	42000	tablespace '%s' is not writable for logging index
16156	42000	tablespace '%s' is not writable for nologging index
16157	42000	ORDER BY item must be the number of a SELECT-list expression
16158	42000	group function is not allowed here
16159	42000	invalid number of arguments
16160	42000	group function is nested too deeply
16161	42000	not a single-group group function
16162	42000	mismatched datafile pairs
16163	42000	cannot rename datafile of ONLINE tablespace
16164	42000	file does not exist
16165	42000	tablespace is not online
16166	42000	tablespace is not offline
16167	42000	cannot offline temporary tablespace
16168	42000	cannot offline dictionary tablespace
16169	42000	cannot offline undo tablespace
16170	42000	invalid WAIT interval
16171	42000	column is not updatable
16172	22003	data is outside the range of the data type to which the number is being converted
16173	22015	one or more fields of data value were truncated during conversion
16174	42000	system session cannot be killed
16175	42000	hint not allowed in positioned DELETE
16176	42000	hint not allowed in positioned UPDATE
16177	34000	cursor '%s' is not updatable
16178	34000	cursor cannot identify the underlying table
16179	HY109	cursor was not positioned on a certain row
16180	HY109	cursor was positioned on a row that had been deleted
16181	42000	the CDS database cannot be closed with OPERATIONAL option
16182	42000	invalid segment identifier
16183	42000	FETCH position must be numeric value
16184	42000	cannot drop all columns in a table

Error code	SQLSTATE	Message
16185	42000	column already has a NOT NULL constraint
16186	42000	column does not have a NOT NULL constraint
16187	42000	null values found
16188	HY000	result set does not exist
16189	42000	column is not an identity column
16190	42000	column cannot be cast to incompatible type
16191	42000	column cannot be cast to decrease precision or scale
16192	42000	column cannot be cast to decrease string length
16193	42000	column cannot be cast to incompatible char length unit
16194	42000	column cannot be cast to incompatible interval indicator
16195	42000	float precision specifier is out of range(%d to %d)
16196	42000	file is already used in log stream - '%s'
16197	42000	duplicate logfile name
16198	42000	size of log file is smaller than minimum size of log file(%ld bytes).
16199	42000	can not find specified log member.

Table A-8 SQL handling related error (16200~16299)

Error code	SQLSTATE	Message
16200	42000	the number of source logfiles is different from the number of targets
16201	42000	can not find specified log file.
16202	42000	name '%s' is already used by an existing schema
16203	42000	schema not empty, use CASCADE option
16204	RD000	another checkpoint is in progress
16205	42000	boolean expression expected
16206	RD000	maximum number of arguments exceeded
16207	42000	%s and %s cannot be matched
16208	42000	insufficient privileges
16209	42000	user %s lacks CREATE SESSION privilege; logon denied
16210	42000	lacks privilege (%s ON DATABASE)
16211	42000	lacks privilege (%s ON TABLESPACE "%s")
16212	42000	lacks privilege (%s ON SCHEMA "%s")
16213	42000	user name '%s' conflicts with another user or role name
16214	42000	tablespace '%s' is not data tablespace
16215	42000	tablespace '%s' is not temporary tablespace
16216	28000	user '%s' does not exist
16217	42000	user has own schema: DROP SCHEMA first
16218	42000	CASCADE must be specified to drop user '%s'
16219	42000	you may not GRANT/REVOKE privileges to/from yourself
16220	42000	duplicate authorization name in grantee/revokee list

Error code	SQLSTATE	Message
16221	42000	user default schema dropped
16222	42000	user default data tablespace dropped
16223	42000	user default temporary tablespace dropped
16224	42000	no privileges to GRANT ALL PRIVILEGES
16225	42000	insufficient privilege to GRANT
16226	42000	duplicate privilege listed
16227	42000	cannot drop a user that is currently connected
16228	42000	fixed table not grantable/revokable
16229	42000	invalid old password
16230	42000	schema element does not match schema identifier
16231	28000	cannot GRANT/REVOKE privileges to/from built-in authorization
16232	28000	cannot REVOKE privileges you did not grant to '%s'
16233	HY000	OFFLINE IMMEDIATE disallowed; noarchivelog mode
16234	HY000	OFFLINE IMMEDIATE disallowed; no logging tablespace
16235	2B000	dependent privilege descriptors still exist
16236	0A000	not supported query%s
16237	42000	a subquery block has incorrect number of result columns
16238	42000	ORDER BY item must be the number of a SELECT-list expression
16239	42000	expression must have same datatype as corresponding expression
16240	42000	subquery expressions not allowed here
16241	42000	invalid number of column names specified
16242	42000	must name this expression with a column alias
16243	01000	Warning: View definition has compilation errors
16244	42000	bind variables not allowed for data definition operations
16245	42000	view "%s.%s" has errors
16246	42000	illegal use of %s data type
16247	HY000	cannot BACKUP; noarchivelog mode
16248	HY000	cannot RECOVER TABLESPACE; tablespace is in use
16249	HY000	cannot SHUTDOWN; backup in progress
16250	HY000	RECOVER UNTIL CANCEL can be issued by directly attached session only - deprecated
16251	42000	single-row subquery returns more than one row
16252	42000	circular view definition encountered
16253	42000	only base table columns are allowed to GRANT/REVOKE
16254	42000	lacks privilege (%s ON TABLE "%s"."%s")
16255	42000	lacks privilege (USAGE ON SEQUENCE "%s"."%s")
16256	07001	wrong number of parameters
16257	HY000	argument of CASE/WHEN must be type boolean
16258	HY000	the XID does not exist
16259	HY000	the XID string is not valid

Error code	SQLSTATE	Message
16260	HY000	the global transaction is in progress
16261	42000	invalid INCREMENTAL BACKUP LEVEL specified : %d
16262	42000	INCREMENTAL BACKUP LEVEL 0 cannot have any BACKUP options
16263	HY000	cannot RESTORE TABLESPACE; tablespace is in use
16264	HY000	transaction is in progress
16265	42000	outer join operator (+) cannot be used with ANSI joins
16266	42000	outer join operator (+) not allowed in %s
16267	42000	an outer join cannot be specified on a correlation column
16268	42000	a predicate may reference only one outer-joined table
16269	42000	a table may be outer joined to at most one other table
16270	42000	two tables cannot be outer-joined to each other
16271	42000	RESTART value cannot be made to be less than the MINVALUE
16272	42000	RESTART value cannot be made to exceed the MAXVALUE
16273	42000	MINVALUE cannot be made to exceed the current value
16274	42000	MAXVALUE cannot be made to be less than the current value
16275	42000	MINVALUE cannot be made to exceed the START WITH value
16276	42000	MAXVALUE cannot be made to be less than START WITH value
16277	42000	log group id is already used
16278	42000	identity column cannot have a default value
16279	42000	DEFAULT expression has errors
16280	42000	the statement is disallowed
16281	42000	resource manager doing work outside transaction
16282	42000	MEMORY tablespace does not support an AUTOEXTEND option
16283	HY000	invalid ROWID
16284	42000	unable to execute in CDS mode
16285	RP000	failed to create system threads
16286	HY000	valid incremental backup does not exist
16287	42000	log group id does not exist
16288	42000	invalid string value specified
16289	42000	into clause can have only one row
16290	42000	SHUTDOWN must be executed in dedicated session
16291	40002	transaction rollback: integrity constraint violation
16292	42000	column length specifier is out of range(1 to %d) for columnar table
16293	42000	columnar table does not support LONG VARCHAR and LONG VARBINARY
16294	42000	cannot defer a constraint that is not deferrable
16295	42000	invalid NOT NULL constraint specified on an identity column
16296	42000	constraint '%s' is not deferrable
16297	42000	duplicate constraint name specified
16298	42000	cannot make primary key supplemental log data with primary key constraint violation

Error code	SQLSTATE	Message
16299	42000	datafile does not exist in database

Table A-9 SQL handling related error (16300 ~ 16399)

Error code	SQLSTATE	Message
16300	42000	too many SQL resource
16301	42000	name '%s' is already used by an existing public synonym
16302	42000	private synonym '%s' to be dropped does not exist
16303	42000	public synonym '%s' to be dropped does not exist
16304	42000	looping chain of synonyms
16305	42000	synonym translation is no longer valid
16306	42000	expression is out of iteration time scope
16307	42000	constant value expression expected
16308	42000	invalid or redundant resource
16309	42000	the account is locked
16310	42000	the password will expire in %ld days
16311	42000	the password will expire soon
16312	42000	the password has expired
16313	42000	the password cannot be reused
16314	42000	cannot lock "SYS" account
16315	42000	profile '%s' already exists
16316	42000	function '%s' not found
16317	42000	profile '%s' has users assigned, cannot drop without CASCADE
16318	42000	profile '%s' does not exist
16319	42000	cannot assign PROFILE to "SYS" account
16320	42000	cannot assign DEFAULT parameter value to "DEFAULT" profile
16321	42000	password length less than '%d'
16322	42000	password must contain at least '%d' letter(s)
16323	42000	password must contain at least uppercase '%d' character(s)
16324	42000	password must contain at least lowercase '%d' character(s)
16325	42000	password must contain at least '%d' digit(s)
16326	42000	password must contain at least '%d' special character(s)
16327	42000	password same as or similar to user name
16328	42000	password contains the user name reversed
16329	42000	password same as or similar to database name
16330	42000	password too simple
16331	42000	Password should differ from the old password by at least '%d' characters
16332	42000	constant value does not support LONG VARCHAR and LONG VARBINARY
16333	42000	value of parameter number (%d) must be consistent datatype
16334	42000	invalid value is prevented in parameter number (%d)

Error code	SQLSTATE	Message
16335	42000	UPDATE primary key not allowed for supplemental log table
16336	42000	cluster group '%s' does not exist
16337	42000	duplicate cluster object listed
16338	42000	cluster domain '%s' does not exist
16339	42000	invalid hash shard count (must between %d and %d)
16340	42000	not found available sharding key column
16341	42000	cannot specifiy sharding strategy and placement for a table on dictionary schem a '%s'
16342	HYC00	not implemented feature for cluster system, in a function %s
16343	42000	invalid syntax for a stand-alone system
16344	42000	database is not local opened
16345	42000	database is already opened
16346	42000	name '%s' is already used by an existing cluster object
16347	42000	cluster port is out of range(%d to %d)
16348	42000	duplicate cluster connection listed
16349	42000	the first cluster group does not contain local cluster member '%s'
16350	42000	maximum number of cluster members exceeded
16351	HY000	failed to connect to the cluster member '%s'
16352	42000	mismatch local cluster member with location file
16353	42000	conflict between new cluster member and location file
16354	HY000	connection of member '%s' is broken
16355	42000	non-deterministic function not allowed here
16356	42000	version conflict
16357	42000	must be accessible to at least one member of group '%s'
16358	42000	accessible member does not exist
16359	42000	no inactive members
16360	42000	cloned table "%s"."%s" must be accessible to at least one member
16361	42000	sharded table "%s"."%s" must be accessible to at least one member of group '%s'
16362	42000	NODE SHARDED table cannot be rebalanced
16363	42000	the cluster member '%s' does not join in a cluster system
16364	42000	cluster group has sharded data
16365	42000	schema name of instant table must be SESSION_SCHEMA
16366	42000	hash shard count smaller than AT CLUSTER GROUP list
16367	42000	cannot drop sharding key column
16368	42000	not a fixed table
16369	42000	not a statistics table
16370	42000	have to create one or more datafiles to each member
16371	42000	conversion is not applicable
16372	42000	duplicate shard name

Error code	SQLSTATE	Message
16373	42000	the number of elements in shard bound mismatch the sharding key columns
16374	42000	range shard bound value must be one of: valid constant or MAXVALUE
16375	42000	only MAXVALUE allowed after MAXVALUE
16376	42000	multiple MAX shard defined
16377	42000	MAX shard not defined
16378	42000	duplicate shard bound value
16379	42000	failed to rebalance some tables
16380	HYC00	UNIQUE or PRIMARY KEY must include all sharding key columns for cluster system
16381	42000	invalid shard count (must between %d and %d)
16382	42000	cluster member '%s' does not exist
16383	42000	domain '%s' is unknown
16384	42000	multiple DEFAULT shard defined
16385	42000	DEFAULT shard not defined
16386	42000	DEFAULT cannot be specified with other values
16387	42000	list shard bound value must be one of: valid constant or NULL or DEFAULT
16388	HYC00	does not support deferrable constraints in the cluster system
16389	42000	domain names of the datafile for renaming don't mismatch
16390	42000	invalid table for sharding strategy and placement specified
16391	42000	the number of sharding key in function is mismatch
16392	HYC00	does not support ROWID in the cluster system
16393	01000	Warning: accessible member does not exist
16394	01000	Warning: connection of member '%s' is broken
16395	01000	Warning: must be accessible to at least one member of group '%s'
16396	42000	exceeds the maximum number of nodes
16397	42000	exceeds the maximum number of groups
16398	42000	the domain of property does not match with domain '%s'
16399	42000	cannot insert or update a generated always identity column, value shall be a DEFAULT

Table A-10 SQL handling related error (16400 ~ 16499)

Error code	SQLSTATE	Message
16400	HY000	cannot RECOVER UNTIL CHANGE SCN; invalid SCN format
16401	42000	invalid SAMPLE size specified
16402	42000	duplicate index name
16403	42000	failed to join some of nodes to the global database
16404	42000	cannot execute the statement with OS user permissions
16405	42000	some tables in the database need to be rebalanced
16406	42000	invalid routine name
16407	42000	procedure or function does not exist

Error code	SQLSTATE	Message
16408	42000	bind variables not allowed for procedures or functions
16409	01000	Warning: Routine definition has compilation errors
16410	42000	Startup driver node must have the latest data - a suitable startup driver node is '%s' member
16411	42000	maximum number of recursive SQL levels (%d) exceeded.
16412	42000	all of shards in a table must be online.
16413	42000	LOCAL_OFFLINE domain must be specified to base table
16414	42000	domain option not allowed here
16415	42000	LOCAL_OFFLINE domain cannot be used with other domains
16416	42000	lacks privilege (EXECUTE ON PROCEDURE "%s"."%s")
16417	42000	active member '%s' cannot be offlined
16418	42000	cluster member '%s' does not exist in cluster group '%s'
16419	42000	object identifier not allowed here
16420	0A000	does not support adding identity column in the cluster system
16421	0A000	does not support deferred unqiue integrity in the cluster system
16422	0A000	does not support SERIALIZABLE in the cluster system
16423	42000	does not support non-deterministic DML in the cluster system : global secondary index expected
16424	42000	function generating different values is not allowed here
16425	42000	procedure, function, package, or type is not allowed here
16426	42000	does not support ADD COLUMN DEFAULT generating different values in the cluster system
16427	42000	invalid events (%s)
16428	42000	cluster system is modified by concurrent cluster DDL execution
16429	42000	does not support cursor in the cluster system : global secondary index expected
16430	42000	in the GLOBAL OPEN phase, the cluster database can be closed only with ABORT option
16431	42000	non-deterministic expression not allowed here : may lead to a replica synchronization failure
16432	42000	global secondary index already exists in table
16433	42000	global secondary index does not exist in table
16434	HY000	invalid grid is prevented in cluster operation, in a function %s
16435	HY000	duplicate source datafile name
16436	HY000	duplicate target datafile name
16437	HY000	cannot execute on cloned tables
16438	HY000	invalid shard name - '%s'
16439	HY000	cannot move shard to the same group
16440	HY000	cannot execute on cluster wide sharded tables
16441	42000	not analyzed table
16442	42000	invalid analyzed file format (%s)

Error code	SQLSTATE	Message
16443	42000	old plan
16444	42000	invalid cluster group name
16445	42000	REBALANCE EXCLUDE can be executed only on cluster wide sharded tables
16446	42000	REBALANCE EXCLUDE target group is empty
16447	HY000	internal error (%s, %d)
16448	42000	of the total '%d' tables, '%d' tables failed to move shard
16449	42000	cluster group does not have sharded data
16450	42000	exceeded global property lock timeout
16451	42000	rownum not allowed here
16452	42000	rownum not allowed in sensitive cursor
16453	42000	rownum not allowed in positioned UPDATE
16454	42000	function '%s' not allowed here
16455	42000	routine "%s.%s" has errors
16456	42000	the startup phase '%s' of cluster member '%s' is mismatch with the required st artup phase '%s'
16457	42000	the connection map of cluster member '%s' is mismatch with local connection map
16458	42000	reserved domain cannot be used with other domains
16459	42000	cannot execute on hash sharded tables
16460	42000	cannot execute on range sharded tables
16461	42000	invalid shard values
16462	42000	cannot split shard - source shard cannot be empty
16463	42000	first CREATE CLUSTER GROUP must include only one cluster member
16464	HY000	unsupported feature with temporary table
16465	42000	cannot drop local cluster group
16466	42000	cannot execute with SERIALIZABLE
16467	42000	table '%s' is READ ONLY
16468	HY000	cannot create TEMPORARY object in a NON-TEMPORARY tablespace
16469	0A000	cannot execute on global connection
16470	42000	datafile size specified is smaller than minimum required '%ld'
16471	42000	audit policy '%s' does not exists
16472	42000	audit policy cannot be dropped as it is currently enabled
16473	42000	audit policy '%s' already exists
16474	42000	not auditable option
16475	42000	audit policy already applied with the BY clause
16476	42000	audit policy already applied with the EXCEPT clause
16477	42000	not auditable object
16478	42000	tablespace '%s' is not temporary tablespace or data tablespace
16479	42000	failed to update system version - %s
16480	42000	cannot set irrecoverable on the local cluster member

Error code	SQLSTATE	Message
16481	42000	global tempoary table segment no longer exists
16482	42000	database is not accessible; '%s' has detached from the cluster
16483	42000	of the total '%d' sequences, '%d' sequences failed to synchronize
16484	42000	of the total '%d' identity columns, '%d' identity columns failed to synchronize
16485	42000	table does not have an identity column
16486	RD000	maximum file size is out of range(%ld to %ld)
16487	42000	only range sharded table is allowed
16488	42000	shards being merged are not adjacent
16489	42000	cluster group name must be specified
16490	42000	mismatch the group name of the source shard '%s'
16491	42000	exceed the maximum number of shards (%d)
16492	42000	can not perform DDL/DML over objects in Recycle Bin
16493	42000	object not in Recycle Bin
16494	42000	original name is used by an existing object
16495	HY000	package or package body does not exist
16496	42000	built-in package '%s' is not modifiable
16497	42000	lacks privilege (EXECUTE ON PACKAGE "%s"."%s")
16498	42000	Only Btree indexes can be rebuilt
16499	42000	operation is not allowed on immutable table '%s'

Table A-11 SQL handling related error (16500 ~ 16599)

Error code	SQLSTATE	Message
16500	0A000	cannot execute on global session
16501	42000	size of log file is larger than maximum size of log file(%ld bytes).
16502	42000	recursive WITH clause must reference itself directly in one of the UNION ALL branches
16503	42000	recursive WITH clause must reference itself in the last UNION ALL branch
16504	42000	recursive WITH clause must use UNION ALL operation
16505	42000	recursive WITH clause must have column alias list
16506	42000	unsupported join in recursive WITH query
16507	42000	cycle mark value and non-cycle mark value must be one byte character
16508	42000	only recursive WITH element can have recursive search clause
16509	42000	unsupported operation in recursive branch of recursive WITH clause
16510	42000	unsupported use of WITH clause
16511	42000	cycle detected while executing recursive WITH query
16512	42000	CONNECT BY clause required in this query block
16513	42000	hierarchy function not allowed here
16514	42000	cycle detected while executing CONNECT BY Clause
16515	42000	ORDER SIBLINGS BY requires specific target expressions

Error code	SQLSTATE	Message
16516	42000	invalid argument expressions for SYS_CONNECT_BY_PATH
16517	42000	NOCYCLE keyword is required with CONNECT_BY_ISCYCLE
16518	42000	LOCAL_OFFLINE domain expected
16519	42000	global secondary index expected in cluster DML
16520	42000	long varying memory manager does not exist
16521	42000	invalid hash bucket count (must between 1 and %ld)
16522	42000	invalid cluster member position
16523	08S01	the database system is shutting down
16524	42000	package specification '%s' does not exist. package body requires its package specification
16525	42000	duplicate name found in WITH name list for WITH clause
16526	42000	only recursive WITH element can have recursive search clause
16527	42000	only recursive WITH element can have recursive cycle clause
16528	42000	illegal reference of a query name in WITH clause
16529	42000	recursive query name referenced more than once in recursive branch of recursive WITH clause element
16530	42000	number of WITH clause column names does not match number of elements in select list
16531	42000	sequence column name for SEARCH clause must not be part of the column alias list
16532	42000	cycle mark column name for CYCLE clause must not be part of the column alias list
16533	42000	sequence column for SEARCH clause must be different from the cycle mark column for CYCLE clause
16534	42000	element in sort specification list of SEARCH clause did not appear in the column alias list of the WITH clause element
16535	42000	element in cycle column list of CYCLE clause must appear in the column alias list of the WITH clause element
16536	42000	duplicate name found in search column list for SEARCH clause of WITH clause
16537	42000	duplicate name found in cycle column list for CYCLE clause of WITH clause
16538	42000	cycle mark value and non-cycle mark value must be one byte character string values
16539	42000	cycle value for CYCLE clause must be different from the non-cycle value
16540	42000	multiple WITH clauses not allowed
16541	42000	cannot affect row a second time
16542	42000	upsert not allowed for the row that caused more than one unique constraint violation
16543	42000	upsert is not allowed for the table with a deferrable constraint (%s.%s)
16544	01000	Warning: Package '%s.%s' has compilation errors
16545	42000	cloned table "%s"."%s" must have at least one online replica
16546	42000	sharded table "%s"."%s" must have at least one online replica of group '%s'

Error code	SQLSTATE	Message
16547	01000	of the total '%d' tables in the database, '%d' tables have offline replica
16548	01000	of the total '%d' tables in the member to be dropped, %d' tables have offline r eplica
16549	42000	invalid bind parameter
16550	HY000	failed to offline the already dropped cluster member
16551	HY000	shutdown cannot be performed on ipc
16552	42000	cannot execute on NODE SHARDED table
16553	42000	of the total '%d' tables, '%d' tables failed to drop offline segments
16554	42000	NODE SHARDED table cannot be synchronized
16555	42000	of the total '%d' tables, '%d' tables failed to synchronize
16556	42000	shard divisor is out of range(1 to %d)
16557	42000	duplicate window name specified
16558	42000	window does not exist
16559	42000	missing ORDER BY expression in the window specification
16560	42000	DISTINCT is not implemented for window functions
16561	42000	frame start cannot be UNBOUNDED FOLLOWING
16562	42000	frame end cannot be UNBOUNDED PRECEDING
16563	42000	frame starting from current row cannot have preceding rows
16564	42000	frame starting from following row cannot have preceding rows
16565	42000	window functions are not allowed here
16566	42000	window specification referencing a window name cannot have PARTITION BY cl ause
16567	42000	referenced window name with ORDER BY clause specified already
16568	42000	cannot reference a window name defined with WINDOW FRAME clause
16569	42000	window frame offset must not be negative
16570	42000	RANGE with offset PRECEDING/FOLLOWING requires exactly one ORDER BY col umn
16571	42000	RANGE with offset PRECEDING/FOLLOWING is not supported for column type
16572	42000	window frame offset must not be null
16573	42000	argument of ntile must be greater than zero
16574	42000	window order by is not allowed here
16575	42000	window frame is not allowed here
16576	42000	argument should be a function of expressions in PARTITION BY
16577	42000	offset argument is out of range
16578	42000	percentile value should be a number between 0 and 1
16579	42000	INDEX COALESCE only supports b-tree type index
16580	42000	table '%s' is not online on cluster domain '%s'
16581	42000	cannot find global transaction logfile
16582	42000	there are active cluster members in the target cluster group '%s'
16583	42000	cannot transactional service: %s

Error code	SQLSTATE	Message
16584	42000	index type '%s' exceeds maximum shard sequence

A.7 PSM Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-12 PSM related error

Error code	SQLSTATE	Message
17000	2F000	failed to initialize PSM processor layer
17001	2F000	not implemented feature (%s)
17002	2F000	duplicated label name
17003	2F000	duplicated exception name
17004	2F000	duplicated variable name
17005	2F000	not applicable expression type
17006	2F000	unknown variable or column name (%s)
17007	2F000	invalid expression
17008	2F000	wrong number of parameters
17009	2F000	NOT NULL constraint violation
17010	2F000	can not find label name
17011	2F000	duplicated field name
17012	2F000	unknown type name
17013	2F000	the type of record field should be scalar
17014	2F000	target host parameter of assignment is not bound in OUT mode
17015	2F000	cannot set constraint or initial value on non-scalar-type variable
17016	2F000	variables which have NOT NULL constraint should have initial values
17017	2F000	unhandled user exception (%s)
17018	2F000	pre-defined exception name not allowed in declare section
17019	2F000	OTHERS exception must be last among the exceptions of a block
17020	2F000	unknown exception
17021	2F000	Invalid error number for PRAGMA EXCEPTION_INIT
17022	2F000	A CASE statement must either list all possible cases or have an else clause
17023	2F000	duplicated error-code in same exception handler
17024	2F000	(%s) cannot be used as assignment target
17025	2F000	The command cannot be executed when global transaction is in the ACTIVE sta te
17026	2F000	Transaction is READ ONLY
17027	2F000	duplicated identifier
17028	2F000	out of scope
17029	2F000	too many depths of scope (limit=%d)
17030	2F000	invalid reference (%s)

Error code	SQLSTATE	Message
17031	2F000	mismatch identifier type
17032	2F000	PSM compilation error
17033	2F000	duplicated cursor name
17034	2F000	return type of cursor is invalid
17035	2F000	invalid expression type
17036	2F000	cursor is not defined
17037	2F000	cursor is already open
17038	2F000	cursor is not open
17039	2F000	unknown cursor
17040	2F000	fetch target count mismatch
17041	2F000	execution fail
17042	2F000	bind variables not allowed for data definition operations
17043	2F000	no target-list for into clause
17044	2F000	returning-into need to bind variable with returning-into or using OUT clause
17045	2F000	no data found
17046	2F000	variables of record type can not be mixed with other variable
17047	2F000	only record-type variable allowed
17048	2F000	can not be used with INTO-clause and USING clause having OUT-modes
17049	2F000	can not use a expression in USING OUT binding
17050	2F000	query need to bind with INTO-clause for targets
17051	2F000	invalid statement-type
17052	2F000	Alias required in list of cursor to avoid duplicated target names
17053	2F000	schema or table object does not exist
17054	2F000	only record-type variable allowed
17055	2F000	this cursor must be declared with FOR-UPDATE to use with CURRENT OF
17056	2F000	invalid syntax error
17057	2F000	only select-into statement allowed
17058	2F000	source-name of return-type is not record-type or record-type variable
17059	2F000	source-name of rowtype must be a table, view, cursor, cursor-variable
17060	2F000	a cursor-variable with having return-type cannot be used in OPEN FOR dynamic-sql
17061	2F000	it can be allowed with only IN-Binding type
17062	2F000	duplicated routine name
17063	2F000	unknown routine name
17064	2F000	invalid routine name
17065	2F000	invalid routine object status
17066	2F000	a RAISE statement with no exception name must be inside an exception handler
17067	2F000	return types of Result Set variables or query do not match
17068	2F000	wrong number or types of arguments
17069	2F000	invalid variable usage

Error code	SQLSTATE	Message
17070	2F000	return statement in procedures or anonymous blocks cannot have <value> claus e
17071	2F000	return statement in functions should have <value> clause
17072	2F000	existing state of routine has been discarded
17073	2F003	DDLs or DCLs cannot be invoked during DMLs or SELECT statements are executi ng in ₩"%s₩".₩"%s₩"
17074	2F003	cannot begin transaction inside a query in ₩"%s₩".₩"%s₩"
17075	2F003	cannot commit or rollback transaction inside a query or DML in ₩"%s₩".₩"%s ₩"
17076	2F000	application error ₩"%s₩"
17077	2F000	the argument of collection type variable is not a constant expression
17078	2F000	a function with OUT-Parameter not allowed in expression
17079	2F000	a nested function not allowed in executing SQL
17080	2F002	cannot execute DML jobs inside a query
17081	2F000	program body of the routine is not defined
17082	2F000	bind parameter in target or return clause is not allowed
17083	2F000	Function "%s" returned without value
17084	2F000	The argument "%s" of routine "%s" is invalid
17085	2F000	a cursor (%s) already defined in package spec
17086	2F000	a cursor or routine (%s) declared in spec must be defined in package body
17087	2F000	existing state of package has been discarded
17088	2F000	invalid package object status
17089	2F000	cannot drop packages in use
17090	2F000	Cursor Variables cannot be declared as part of a package
17091	2F000	cannot perform nested updatable statements on cluster database
17092	2F000	invalid package name
17093	2F000	unsupported element data type
17094	2F000	IN cursor cannot be OPENed
17095	2F000	mismatch parameter count of routine(%s)
17096	2F000	mismatch parameter type of routine(%s)
17097	2F000	mismatch return type of function(%s)
17098	2F000	mismatch record field count of routine(%s)
17099	2F000	cycle detected when initialize package instance(%s)
17100	2F000	PSM(%s) compilation error
17101	2F000	PACKAGE(%s) Specification has compilation errors
17102	2F000	PACKAGE(%s) Body has compilation errors
17103	2F000	cannot use function in the same package
17104	2F000	invalid REFCURSOR handle
17105	2F000	invalid REFCURSOR binding
17106	2F000	unknown field name (%s)

Error code	SQLSTATE	Message
17107	2F000	only cursor variable allowed
17108	2F000	Implicit result cannot be returned through this statement
17109	2F000	table function need return table statement
17110	2F000	return table statement not allowed
17111	2F000	only select statement allowed
17112	2F000	mismatch field count of function table(%s)
17113	2F000	cursor that have never fetched is allowed

A.8 Session Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-13 Session related error

Error code	SQLSTATE	Message
18000	HY000	Invalid Statement
18001	HY024	Invalid attribute value
18002	HY092	Invalid attribute/option identifier
18003	42000	Transaction is READ ONLY
18004	42000	Statement cannot execute as atomic action
18005	42000	Preparable phase is open phase
18006	08S01	Communication link failure
18007	HYC00	not implemented feature, in a function %s
18008	HY014	Limit on the number of SQL statement exceeded
18009	42000	The command cannot be executed when global transaction is in the ACTIVE sta te
18010	07009	Invalid descriptor index
18011	HY000	Character set not equal(%s and %s)
18012	HY010	fetch out of sequence
18013	HYC00	Optional feature not implemented
18014	07001	Wrong number of parameters
18015	24000	Invalid cursor state
18016	42000	remote xa transaction (%d.%ld) already exists
18017	42000	remote xa transaction (%d.%ld) does not exist

A.9 ODBC Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-14 ODBC related error

Error code	SQLSTATE	Message
19000	HY009	Invalid use of null pointer
19001	HY092	Invalid attribute/option identifier
19002	08003	Connection not open
19003	IM001	Driver does not support this function
19004	IM004	Driver's SQLAllocHandle on SQL_HANDLE_ENV failed
19005	IM005	Driver's SQLAllocHandle on SQL_HANDLE_DBC failed
19006	08002	Connection name in use
19007	HY090	Invalid string or buffer length
19008	28000	Invalid authorization specification
19009	HY010	Function sequence error
19010	24000	Invalid cursor state
19011	HY012	Invalid transaction operation code
19012	HYC00	Optional feature not implemented
19013	HY003	Invalid application buffer type
19014	07009	Invalid descriptor index
19015	HY004	Invalid SQL data type
19016	HY105	Invalid parameter type
19017	HY009	Invalid argument value
19018	HY011	Attribute cannot be set now
19019	HY024	Invalid attribute value
19020	25000	Invalid transaction state
19021	HY000	General error
19022	07005	Prepared statement not a cursor-specification
19023	22001	String data, right truncated
19024	HY007	Associated statement is not prepared
19025	HY092	Option type out of range
19026	22002	Indicator variable required but not supplied
19027	01S01	Error in row
19028	HY016	Cannot modify an implementation row descriptor
19029	HY091	Invalid descriptor field identifier
19030	07006	Restricted parameter value violation
19031	22015	Interval field overflow

Error code	SQLSTATE	Message
19032	22003	Numeric value out of range
19033	01004	String data, right truncated
19034	22018	Invalid character value for cast specification
19035	01S07	Fractional truncation
19036	22008	Datetime field overflow
19037	22007	Invalid datetime format
19038	08S01	Communication link failure
19039	HY000	An invalid SQL type has occurred
19040	34000	Invalid cursor name
19041	HY000	Failed to atomic execution
19042	HY110	Invalid driver completion
19043	IM012	DRIVER keyword syntax error
19044	08001	Client unable to establish connection
19045	HY096	Information type out of range
19046	IM002	Data source not found and no default driver specified
19047	HY106	Fetch type out of range
19048	HY100	Uniqueness option type out of range
19049	HY101	Accuracy option type out of range
19050	HY097	Column type out of range
19051	HY098	Scope type out of range
19052	HY099	Nullable type out of range
19053	HY021	Inconsistent descriptor information
19054	01S02	Option value changed
19055	HY109	Invalid cursor position
19056	HY107	Row value out of range
19057	HYT00	Timeout expired
19058	2C000	Invalid or unknown NLS parameter value specified
19059	HY020	Attempt to concatenate a null value
19060	HY019	Non-character and non-binary data sent in pieces
19061	HY000	Invalid 'odbc.ini' specification. property : [%s]
19062	IM008	Dialog failed
19063	40001	Transaction rollback
19064	40003	Statement completion unknown
19065	08S01	Communication link failure
19066	01000	General warning
19067	HY104	Invalid precision or scale value
19068	40003	Retry the transactional operations
19069	08S01	Failed to communicate with locator
19070	08006	Unable to get cluster system information
19071	HY000	startup cannot be performed on ipc

A.10 JDBC Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-15 JDBC related error

Error code	SQLSTATE	Message
21000	HV000	Not used error
21001	46001	Invalid connection url[%s]: [%s]
21002	46110	Unsupported feature: [%s]
21003	08S01	Invalid protocol: [%s]
21004	HV000	Invalid method call for read-only ResultSet
21005	08003	The Connection is already closed
21006	08003	The Statement is already closed
21007	08003	The ResultSet is already closed
21008	HV000	Invalid method call for forward-only ResultSet
21009	HV000	Invalid Wrapper type: [%s]
21010	HV000	Invalid argument: [%s]
21011	08000	Cannot connect to the server: %s
21012	08S01	Communication link failure: %s
21013	HV000	A mandatory property is omitted: [%s]
21014	HV000	Not a select query for executeQuery method
21015	HV000	The statement has produced a ResultSet
21016	HV000	This method cannot be called for a PreparedStatement
21017	HV000	Parameter type[%s] is mismatch with previous type[%s] during batch
21018	HV000	Lack of parameter: [%s]'th parameter
21019	HV000	No batch job added
21020	HV000	Batch job added
21021	22003	The value[%s] is out of range of [%s] type
21022	22026	The value length is mismatch[%s != %s]
21023	HV000	The value([%s] type) cannot be converted to [%s] type
21024	08000	The connection timeout expired: [%s millis]
21025	HV000	The column is not found: [%s]
21026	3B000	Savepoint error: %s
21027	HV000	Cannot execute the query in auto-commit mode
21028	24000	Invalid cursor position
21029	HV000	No column has been read
21030	HV000	Encoding error: [%s]
21031	HV000	Decoding error: [%s]

Error code	SQLSTATE	Message
21032	HV000	Stream error: [%s]
21033	HV000	Trying to read the deleted row
21034	HV000	Invalid interval format string: [%s]
21035	HV000	The Statement has an opened ResultSet
21036	HV000	The RowId object is unknown
21037	08000	The failover failed
21038	07009	The parameterIndex is not valid: [%s]
21039	HY000	No data read
21040	HY000	Invalid locator protocol version
21041	HY000	not matched locator protocol command sequence number
21042	HY000	Unable to open the file '%s'
21043	HY000	Unable to read the file '%s'
21044	HY000	Unable to close the file '%s'
21045	HY000	invalid INI file format
21046	HY000	Parameter[%s]'s input data type is mismatched with output data type
21047	40001	Retry the transactional operations again
21048	08006	Unable to get cluster system information
21049	HY000	The Statement does not have a ResultSet
21050	08006	Unable to get location information
21051	HY000	Not exist node information - %s
21052	HY000	Operation on freed LOB
21053	HY000	Invalid sql : %s
21054	HY000	Can not combine auto-generated keys with batch update

A.11 Embedded SQL Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-16 Embedded SQL run-time library related error

Error code	SQLSTATE	Message
23000	42000	syntax error
23001	0A000	feature not supported
23002	HY000	Bind variable "%s" was not declared
23003	HY000	Application can't be initialized
23004	HY000	Application can't be finalized
23005	RD000	Service is not available
23006	RD000	Statement is not prepared
23007	08001	Client can't establish SQL-connection
23008	HY000	cursor "%s" is not open
23009	HY000	Application can't create semaphore
23010	HY000	Application can't destroy semaphore
23011	HY000	Application can't create allocator
23012	HY000	Application can't destroy allocator
23013	HY000	Application can't create symbol table
23014	HY000	Application can't destroy symbol table
23015	IM004	Driver's SQLAllocHandle on SQL_HANDLE_ENV failed
23016	IM001	Driver does not support ODBC 3.0
23017	HY000	Application can't allocate memory
23018	42000	Statement "%s" was not prepared
23019	42000	Descriptor name is too long
23020	HYC00	Not implemented feature, in a function %s
23021	07004	USING clause required for dynamic parameters
23022	07001	USING clause: illegal variable name/number
23023	07001	USING clause: not all variables bound
23024	07007	INTO clause required for result fields
23025	07001	INTO clause: illegal variable name/number
23026	07001	INTO clause: not all variables bound
23027	22002	null value, no indicator parameter
23028	07005	cusor "%s" statement is not a query
23029	08003	connection not open
23030	34000	cursor "%s" is not updatable
23031	08002	connection name "%s" in use

Error code	SQLSTATE	Message
23032	08002	duplicate connection context
23033	HY000	GROUPID type is only available with global connection
23034	42000	SELECT INTO returns too many rows
23035	24000	cursor "%s" is not declared
23036	24000	cursor "%s" is not closed
23037	24000	cursor "%s" is already opened
23038	24000	cursor "%s" is not a dynamic cursor
23039	HY000	invalid cursor

Table A-17 Embedded SQL precompile related error

Error code	SQLSTATE	Message
41000	42000	syntax error
41001	0A000	feature not supported
41002	42000	Host variable "%s" not declared
41003	42000	Cursor "%s" not declared
41004	42000	"%s": file not exist
41005	42000	expression type does not match usage
41006	42000	Fatal error while doing embedded SQL preprocessing
41007	HY000	Precompiler can't create symbol table
41008	HY000	Precompiler can't destroy symbol table
41009	42000	identifier is too long
41010	42000	token "%s" is not valid in preprocessor expressions
41011	42000	multi-character character constant
41012	42000	floating constant in preprocessor expression
41013	42000	invalid suffix "%s" on integer constant
41014	42000	cursor name "%s" already declared
41015	07004	cursor "%s" is not a dynamic cursor
41016	42000	"%s" has no member named "%s"
41017	42000	storage size of "%s" isn't known
41018	42000	variable "%s" must be integer type
41019	42000	variable "%s" must be string type
41020	42000	invalid indicator variable type: indicator variable "%s" must have numeric type(char/short/int/long)
41021	42000	duplicate "%s"
41022	42000	both "%s" and "%s" in declaration specifiers
41023	42000	two or more data types in declaration specifiers
41024	42000	C++ punctuation sequences are not permitted
41025	42000	host variable expression has invalid type
41026	42000	typedef "%s" has invalid

Error code	SQLSTATE	Message
41027	42000	too many 'define' macro (%d)
41028	42000	'%s' macro is already defined at line %d, in file %s
41029	42000	missing host variable
41030	42000	invalid getting groupid variable type: variable "%s" must have signed numeric type
41031	42000	invalid sql statement: getting groupid is available with select/update/delete/insert statement
41032	HY000	cursor variable type can not use an array

A.12 Communication Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-18 Communication related error

Error code	SQLSTATE	Message
24000	08S01	Communication link failure
24001	08S01	Invalid communication protocol
24002	22000	Invalid GOLDILOCKS_DATA Directory
24003	22000	not supported function
24004	22000	Invalid interprocess protocol
24005	22000	Unable to attach the shared memory segment. unable to access the file '%s'
24006	HV090	Invalid string length or buffer length
24007	22000	not matched command sequence number
24008	HYT01	Connection timeout expired

Table A-19 Server communication related error

Error code	SQLSTATE	Message
52000	08S01	Communication link failure
52001	08000	Invalid communication protocol
52002	HY000	Not enough cm unit memory
52003	HY000	exceeded maximum packet allocation timeout
52004	08001	exceeded maximum number of ipc channels
52005	HYT00	Timeout expired

A.13 ServerLibrary Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-20 ServerLibrary related error

Error code	SQLSTATE	Message
25000	0A000	feature not supported
25001	08001	Server is not running
25002	HY000	Environment Variable "%s" is not defined
25003	08000	Invalid protocol
25004	HY000	STARTUP must be executed in dedicated session
25005	HY000	session killed
25006	HY000	remote session is not opened
25007	42000	the property '%s' of cluster member '%s' is not compatible with the cluster system
25008	42000	the scn '%s' of cluster member '%s' is mismatch
25009	HY000	number of shared servers must be greater than or equal to number of request queues
25010	42000	invaild startup phase
25011	HY000	startup cannot be performed on ipc

A.14 gsql/ gsqlnet Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-21 gsql/ gsqlnet related error

Error code	SQLSTATE	Message
40000	42000	syntax error
40001	0A000	feature not supported
40002	HY000	Bind variable "%s" not declared
40003	42000	"%s": table or view does not exist.
40004	42000	"%s": index does not exist.
40005	RD000	Service is not available
40006	RD000	Statement is not prepared
40007	42000	'%s': duplicate statement
40008	42000	'%s': statement does not exist
40009	42000	'%s': statement is not prepared
40010	42000	insufficient privileges
40011	42000	driver does not support %s function
40012	42000	invalid object for describe
40013	42000	driver does not support the requested connection mode(%s)
40014	HY000	the rollback was caused by an unspecified reason
40015	HY000	the rollback was caused by a communication failure
40016	HY000	a deadlock was detected
40017	HY000	a condition that violates the integrity of the resources was detected
40018	HY000	the resource manager rolled back the transaction branch for a reason not on thi s list
40019	HY000	a protocol error occurred in the resource manager
40020	HY000	a transaction branch took too long
40021	HY000	may retry the transaction branch
40022	HY000	the inclusive upper bound of the rollback codes
40023	HY000	resumption must occur where suspension occurred
40024	HY000	the transaction branch may have been heuristically completed
40025	HY000	the transaction branch has been heuristically committed
40026	HY000	the transaction branch has been heuristically rolled back
40027	HY000	the transaction branch has been heuristically committed and rolled back
40028	HY000	routine returned with no effect and may be re-issued
40029	HY000	the transaction branch was read-only and has been committed
40030	HY000	asynchronous operation already outstanding

Error code	SQLSTATE	Message
40031	HY000	a resource manager error occurred in the transaction branch
40032	HY000	the XID is not valid
40033	HY000	invalid arguments were given
40034	HY000	routine invoked in an improper context
40035	HY000	resource manager unavailable
40036	HY000	the XID already exists
40037	HY000	resource manager doing work outside transaction
40038	HY000	unknown xa error
40039	HY000	marked the transaction branch rollback-only for unspecified reason
40040	HY000	cannot open the resource
40041	HY000	cannot close the resource
40042	HY000	the sql is empty
40043	HY000	invalid host variable type
40044	08003	connection does not exist
40045	HY000	invalid NUMSIZE value: NUMSIZE must be between %d and %d
40046	HY000	invalid PAGESIZE value: PAGESIZE must be between %d and %d
40047	HY000	invalid LINESIZE value: LINESIZE must be between %d and %d
40048	HY000	invalid COLSIZE value: COLSIZE must be between %d and %d
40049	HY000	invalid HISTORY value: HISTORY must be less equal than %d
40050	HY000	invalid DDLSIZE value: DDLSIZE must be between %d and %d
40051	HY000	invalid object identifier
40052	HY000	not enough DDLSIZE. use command: ₩set ddlsize {n}
40053	HY000	confirmation password mismatch
40054	HY000	invalid locator ini property - '%s'
40055	HY000	not exist member '%s' in file - '%s'
40056	HY000	some of nodes were failed to startup
40057	HY000	not specified valid location information
40058	HY000	some of nodes were failed to shutdown
40059	HY000	failed to connect to original driver node '%s' after startup
40060	HY000	cannot execute 'cstartup' or 'cshutdown' command in DA mode
40061	HY000	currently connected node is inactive
40062	HY000	file not exist -'%s'

A.15 gloader/ gloadernet Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-22 gloader/ gloadernet related error

Error code	SQLSTATE	Message
42000	RD000	Service is not available
42001	HY000	invalid control file format
42002	HY000	buffer exceeded overflow area
42003	HY000	open same file(%s)
42004	HY000	occur invalid handle error
42005	HY000	occur error while operating SQLGetDiagRec/SQLGetDiagField
42006	HY000	using same character for delimiter and qualifier
42007	HY000	invalid thread unit count.(1~32)
42008	HY000	invalid array size.(1~65535)
42009	HY000	unavailable multithreading in export mode.
42010	08S01	Communication link failure
42011	HY000	invalid file size (minimum size : %d)
42012	HY000	unavailable file size in text mode.
42013	HY000	invalid column type.
42014	HY000	invalid column count.
42015	HY000	invalid data size.
42016	HY000	invalid header of binary file.
42017	HY000	file is empty.
42018	HY000	Qualifier must be 1 length character.
42019	HY000	LF and CR can not be used for delimiter or qualifier.
42020	HY000	qualifiers and terminators must not be subsets of each other
42021	HY000	table name must be given in case of absence of control file.
42022	HY000	can not fetch schema name.
42023	22001	byte length of data greater than column length.
42024	HY000	invalid direct io size

A.16 gmaster Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-23 gmaster related error

Error code	SQLSTATE	Message
44000	HY000	database is already mounted
44001	HY000	database is already local opened
44002	HY000	database is already opened
44003	HY000	system thread is already created
44004	HY000	failed to create system threads (%s)
44005	HY000	member '%s' does not exist
44006	HY000	fail to send packet through communication context
44007	HY000	failed to start up a process '%s'

A.17 glsnr Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-24 glsnr related error

Error code	SQLSTATE	Message
45000	HY000	Listener configuration file "%s" doesn't exist.
45001	HY000	Environment Variable "%s" is not defined.
45002	HY000	Listener is already started.
45003	HY000	overflow fd
45004	08000	Invalid communication protocol
45005	08004	access denied
45006	HY000	Operation timed out
45007	HY000	shared mode inactive
45008	HY000	Invalid property value [DEDICATED | SHARED] : %s
45009	HY000	Invalid property value [NO | INVITED | EXCLUDED] : %s
45010	HY000	Invalid property value [YES | NO | 1 | 0]: %s
45011	HY000	Invalid property value [%ld ~ %ld]: %s
45012	HY000	Invalid property value [%ld ~ %ld]: %s
45013	HY000	buffer over flow %s

A.18 cyclone Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-25 cyclone related error

Error code	SQLSTATE	Message
46000	RD000	Service is not available
46001	HY000	"%s": file does not exist
46002	HY000	configure file processing error
46003	HY000	"%s": table does not exist
46004	HY000	"%s": schema does not exist
46005	HY000	"%s.%s": previously Added. Maybe duplicated
46006	HY000	"%s%s": table must have a primary key
46007	HY000	internal error occurred (%s)
46008	HY000	invalid Datatype(%d)
46009	HY000	"%s.%s" table must set supplemental log or primary key.
46010	HY000	group [%s] is already running
46011	HY000	cyclone is not running
46012	HY000	GOLDILOCKS_DATA system environment is invalid
46013	HY000	"%s" log file reused or invalid. restart cyclone with '--reset' option
46014	HY000	fail to analyze flow[%s]
46015	HY000	Invalid Port Number (1024 ~ 49151) [%s]
46016	HY000	Communication link failure
46017	HY000	Invalid Protocol value (%s)
46018	HY000	Master disconnect abnormally
46019	HY000	Protocol Error Occurred (%s)
46020	HY000	Already Slave Connected
46021	HY000	Invalid Transaction Slot Id
46022	HY000	Invalid Supplemental Log (%d)
46023	HY000	Invalid operation for DDL
46024	HY000	Invalid value to analyze
46025	HY000	Invalid group name
46026	HY000	Not set master ip
46027	HY000	Invalid capture information
46028	HY000	There is no group in configure
46029	HY000	Duplicate group name (%s)
46030	HY000	Invalid Value : %s(%ld)
46031	HY000	Too low value : %s(%ld)

Error code	SQLSTATE	Message
46032	HY000	Invalid Host Port(%d)
46033	HY000	Redo Log file Read Timeout (%s)
46034	HY000	Invalid Archive Log File (%s)
46035	HY000	Invalid Temporary File (%s)
46036	HY000	Fail to write file (%s)
46037	HY000	Fail to Update State (%s)
46038	HY000	Fail to set Propagate mode at Session
46039	HY000	There is no active redo log file.
46040	HY000	There is no restart information. Master Database must be started first.
46041	HY000	Can Not Add New Table. Master Database must be started first.(%s.%s)
46042	HY000	Master Database must be started for the first time.
46043	HY000	Invalid Meta File(%s).
46044	HY000	Redo Log File does not exist(%s).
46045	HY000	USER_ID or USER_PW does not exist.
46046	HY000	Slave is not running with --sync option.
46047	HY000	Invalid Connect Information for sync (%s)
46048	HY000	(%s) value is too long
46049	HY000	Invalid encrypt/decrypt key
46050	HY000	no '--key' option
46051	HY000	password already exists (use one of USER_PW or USER_ENCRYPT_PW)
46052	HY000	reset or sync table (%s.%s) has never participated in replication
46053	HY000	reset or sync argument is invalid (%s)
46054	HY000	Table meta file has broken. please restart with '--reset all'
46055	HY000	protocol timeout
46056	HY000	Slave stopped normally
46057	HY000	Cannot use reset and sync option at the same time.
46058	HY000	There is no (%s.%s) table to sync (--sync option)
46059	HY000	(%s.%s) table (%s) column is not acceptable(GENERATED ALWAYS AS IDENTITY)
46060	HY000	driver does not support %s function
46061	HY000	data type is not supported.
46062	HY000	Cluster information does not matched(before groupid(%d), memberid(%d)/current groupid(%d), memberid(%d))
46063	HY000	Invalid cluster information at configure file(%s)
46064	HY000	Fail to rollback transaction
46065	HY000	Slave is running with --sync option
46066	HY000	MYSQL_DATABASE configuration must needed to connect MySQL/MariaDB
46067	HY000	DB2_DATABASE configuration must needed to connect DB2

A.19 LogMirror Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-26 LogMirror related error

Error code	SQLSTATE	Message
47000	RD000	Service is not available
47001	HY000	Invalid Protocol value (%s)
47002	HY000	"%s": file does not exist
47003	HY000	configure file processing error
47004	HY000	invalid Control file
47005	HY000	Communication link failure
47006	HY000	LogMirror is already running
47007	HY000	GOLDILOCKS_DATA system environment is invalid
47008	HY000	There is no Shared Memory Area for LogMirror
47009	HY000	Invalid Value : %s(%s)
47010	HY000	Invalid port : (%d)
47011	HY000	Master disconnect abnormally
47012	HY000	Invalid Log File(%s)
47013	HY000	Archive Log File does not exist
47014	HY000	Archive Log file Read Timeout (%s)
47015	HY000	LogMirror is not running
47016	HY000	Connection Information does not exist
47017	HY000	Infiniband Receive Fail. (%ld)
47018	HY000	GOLDILOCKS Shutdown or Give-up LogMirror Service.
47019	HY000	Invalid encrypt/decrypt key
47020	HY000	no '--key' option
47021	HY000	password already exists(use one of USER_PW or USER_ENCRYPT_PW)
47022	HY000	protocol heartbeat timeout

A.20 cymon Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-27 cymon related error

Error code	SQLSTATE	Message
48000	HY000	GOLDILOCKS_DATA system environment is invalid
48001	HY000	"%s": file does not exist
48002	RD000	Service is not available
48003	HY000	configure file processing error
48004	HY000	Duplicate group name (%s)
48005	HY000	Invalid Value : %s(%ld)
48006	HY000	Invalid group name
48007	HY000	Invalid Port Number (1 ~ 65535)
48008	HY000	There is no group in configure
48009	HY000	cymon is already running
48010	HY000	Communication link failure
48011	HY000	Protocol Error Occurred (%s)
48012	HY000	Already Monitor Connected
48013	HY000	cymon is not running
48014	HY000	internal error occurred (%s)
48015	HY000	Fail to update monitoring info
48016	HY000	USER_ID or USER_PW does not exist.
48017	HY000	Invalid encrypt/decrypt key
48018	HY000	no '--key' option
48019	HY000	password already exists(use one of USER_PW or USER_ENCRYPT_PW)

A.21 gdispatcher Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-28 gdispatcher related error

Error code	SQLSTATE	Message
50000	HY000	Dispatcher is already started
50001	HY000	overflow fd

A.22 gbalancer Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-29 gbalancer related error

Error code	SQLSTATE	Message
51000	HY000	overflow fd

A.23 gsyncher Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-30 gsyncher related error

Error code	SQLSTATE	Message
53000	HY000	invalid phase(%s): executable phase is %s
53001	HY000	GOLDILOCKS_DATA system environment is invalid
53002	HY000	gmaster is active
53003	HY000	gsyncher is already running
53004	HY000	Shared memory is cleared

A.24 Cluster Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-31 Cluster related error

Error code	SQLSTATE	Message
56000	HY000	location information of member '%s' already exists
56001	HY000	location information of member '%s' does not exist
56002	HY000	location information of member '%s' is modified by concurrent execution
56003	HY000	has inactive member (%s)
56004	HY000	location file is corrupted - '%s'
56005	HY000	connections of some cluster members are broken
56006	40000	transaction rollback: failed to prepare global transaction
56007	HY000	must be accessible to at least one of replicas
56008	40000	transaction rollback: failed to synchronize replicas
56009	HY000	remote server (member ID: %ld) is abnormally terminated
56010	40000	transaction rollback: remote session is abnormally terminated
56011	HY000	a shared session cannot modify NUMA property
56012	HY000	no connection was established to recover in-doubt transactions
56013	HY000	cannot resolve in-doubt transaction
56014	HY000	MAX_NODE_COUNT property value '%d' must be equal to or less than '%d'
56015	HY000	'%s' property value must be greater than or equal to '%s' property value '%d'
56016	HY000	all members affected by the transaction are inactive - cannot guarantee durability of the transaction
56017	40000	transaction rollback: failed to open remote session
56018	HY000	protocol payload size(%ld) exceed maximum packet size(%ld)
56019	HY000	failover busy
56020	HY000	internal error (%s, %d)
56021	HY000	exceeded maximum response timeout
56022	HY000	failed to connect to an host
56023	HY000	invalid cluster communication protocol
56024	HY000	sequence '%s' is offline
56025	HY000	cannot synchronize because some members were disconnected
56026	HY000	identity column '%s' of table '%s' is offline
56027	HY000	cannot resolve in-doubt transaction '%d.%d.%ld' because the transaction remains unfinished
56028	HY000	the property '%s' is not compatible with the cluster system
56029	HY000	unable to open a remote session for read only statement

Error code	SQLSTATE	Message
56030	HY000	total cdispatcher threads '%d' must be equal to or less than '%d'

A.25 cserver Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-32 cserver related error

Error code	SQLSTATE	Message
57000	HY000	sample error

A.26 cdispatcher Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-33 cdispatcher related error

Error code	SQLSTATE	Message
58000	HY000	the sender failed to connect to the member(%d)
58001	HY000	the socket of target member '%s' is already disconnected
58002	HY000	the location information for member '%s' is not found
58003	HY000	the connection information is invalid; %s %d is either equal to or greater than %d
58004	HY000	the connection timeout from remote(%d)
58005	HY000	the disconnection timeout from remote(%d)
58006	HY000	invalid connection state transition
58007	HY000	dispatcher thread counts of two hosts do not match
58008	HY000	failed to start '%s' thread of cluster dispatcher
58009	HY000	internal error (%s, %d)
58010	HY000	broken socket
58011	HY000	old cluster connection is still alive
58012	HY000	failover event is in progress
58013	HY000	the connection can be established at LOCAL OPEN or OPEN phase
58014	HY000	old cluster sessions are still alive
58015	HY000	Timeout expired - %s
58016	HY000	cancel dequeue operation - database is going shutdown
58017	HY000	join database is in progress
58018	HY000	connection request from old cluster
58019	HY000	member position is out of range(0 to %d) - local(%d), remote(%d)
58020	HY000	the cluster protocol version '%d.%d' of cluster member '%d' is not compatible with the cluster protocol version '%d.%d' of cluster member '%d'

A.27 gtrclogger Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-34 gtrclogger related error

Error code	SQLSTATE	Message
59000	HY000	exceeded port cnt
59001	HY000	gtrclogger is already running
59002	HY000	gtrclogger is not running
59003	HY000	Operation timed out
59004	22000	Invalid GOLDILOCKS_DATA directory

A.28 glocator Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-35 glocator related error

Error code	SQLSTATE	Message
60000	HY000	syntax error
60001	HY000	file does not exist - '%s'
60002	HY000	GOLDILOCKS_DATA system environment is invalid
60003	HY000	Invalid value - %s(%s)
60004	HY000	file name is too long - '%s'
60005	HY000	glocator is already started.
60006	HY000	Location file is corrupted - '%s'
60007	HY000	Property value is out of length - '%s'
60008	HY000	Invalid property value [YES | NO | 1 | 0] - '%s'
60009	HY000	Invalid property value [%ld ~ %ld] - '%s'
60010	HY000	Invalid property value [%ld ~ %ld] - '%s'
60011	HY000	File already exists - '%s'
60012	HY000	Location file exceeded max size(%ld) - over size(%ld)
60013	HY000	Not exist node information - '%s'
60014	HY000	Exceeded response time for failover.
60015	HY000	Node is on failover - '%s'
60016	HY000	Need more alternate locator host information.
60017	HY000	Invalid configure file specification - property:[%s]
60018	HY000	Failed to propagate - '%s:%d'
60019	HY000	Not exist service information - '%s'
60020	HY000	'%s' node is already connected.
60021	HY000	overflow fd.

A.29 gloctl Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-36 gloctl related error

Error code	SQLSTATE	Message
61000	HY000	Syntax error
61001	HY000	File does not exist - '%s'
61002	HY000	Can not find any ini object in file - '%s'
61003	HY000	Dsn does not exist - '%s'
61004	HY000	ADD MEMBER keyword syntax is error - '%s'
61005	HY000	ADD SERVICE keyword syntax is error - '%s'
61006	HY000	Invalid property value [YES | NO | 1 | 0] - '%s'
61007	HY000	Invalid property value [%ld ~ %ld] - '%s'
61008	HY000	Invalid property value [%ld ~ %ld] - '%s'
61009	HY000	STL_ENV_DB_DATA system environment is invalid
61010	HY000	Keyword syntax is invalid - '%s'
61011	HY000	Data value is invalid - '%s'

A.30 gmon Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-37 gmon related error

Error code	SQLSTATE	Message
62000	HY000	gmon is already running
62001	HY000	process(%ld) did not respond signal
62002	HY000	GOLDILOCKS_DATA system environment is invalid

A.31 gagent Related Error

- Error code: GOLDILOCKS-specific error code
- SQLSTATE: SQL standard state code
- Message: Error message

Table A-38 gagent related error

Error code	SQLSTATE	Message
63000	HY000	GOLDILOCKS_DATA system environment is invalid
63001	HY000	Property value is out of length - '%s'
63002	HY000	Invalid property value [YES | NO | 1 | 0] - '%s'
63003	HY000	Invalid property value [%ld ~ %ld] - '%s'
63004	HY000	Invalid property value [%ld ~ %ld] - '%s'
63005	HY000	gagent is already started.
63006	HY000	file name is too long - '%s'
63007	HY000	file does not exist - '%s'
63008	HY000	Database is not cluster.
63009	HY000	Invalid configure file specification - property:[%s]
63010	HY000	Invalid phase(%s): executable phase is between %s and %s
63011	HY000	Property PORT of dsn(%s) is not set in file 'odbc.ini'
63012	08000	Timeout expired.
63013	HYT00	Unable to establish connection.

Appendix B.

Wait Event

B.1 Wait Event

Performance views which is related to the wait event is as follows.

- v$system_event **V$SYSTEM_EVENT**
- v$session_event **V$SESSION_EVENT**
- v$session_wait **V$SESSION_WAIT**
- v$wait_event_name **V$WAIT_EVENT_NAME**
- v$wait_event_class_name **V$WAIT_EVENT_CLASS_NAME**

B.2 Class of Wait Event

- Administrative: Waits resulting from DBA commands that cause users to wait (for example, an index rebuild)
- Application: Waits resulting from user application code (for example, lock waits caused by row level l ocking or explicit lock commands)
- Cluster: Waits related to cluster resources (for example, global cache resources)
- Commit: This wait class only comprises one wait event - wait for redo log write confirmation after a c ommit (that is, 'log file sync')
- Concurrency: Waits for internal database resources (for example, latches)
- Configuration: Waits caused by inadequate configuration of database or instance resources (for exa mple, undersized log file sizes, shared pool size)
- Idle: Waits that signify the session is inactive, waiting for work (for example, 'gsql message from clien t')
- Network: Waits related to network messaging
- Other: Waits which should not typically occur on a system
- Scheduler: Resource manager related waits
- System IO: Waits for background process IO
- User IO: Waits for user IO

B.3 Item of Wait Event

ENQUEUE: GDISPATCHER REQUEST

It is the time of which GDISPATCHER enqueues a request in a shared mode.
Parameter: None

ENQUEUE: SHARED-SERVER RESPONSE

It is the time of which a shared server enqueues a response in a shared mode.
Parameter: None

DEQUEUE: SHARED-SERVER REQUEST

It is the time of which a shared server dequeues a request in a shared mode.
Parameter: None

DEQUEUE: GDISPATCHER RESPONSE

It is the time of which GDISPATCHER dequeues a response in a shared mode.
Parameter: None

SEND: DEDICATE-SERVER SPOOLED RESPONSE

It is the time of which a dedicate server sends the spooled response to the client in a dedicate mode.

Parameter	Description
send data size	Bytes of the data to be sent

SEND: DEDICATE-SERVER RESPONSE

It is the time of which a dedicate server sends a response to the client in a dedicate mode.

Parameter	Description
send data size	Bytes of the data to be sent

RECV: DEDICATE-SERVER REQUEST

It is the time of which a dedicate server receives a request from the client in a dedicate mode.
Parameter: None

SEND: GDISPATCHER RESPONSE

It is the time of which GDISPATCHER sends a response to a client in a shared mode.

Parameter	Description
send data size	Bytes of the data to be sent

RECV: GDISPATCHER REQUEST

It is the time of which GDISPATCHER receives a request from a client in a shared mode.
Parameter: None

ENQUEUE: CLUSTER REQUEST

It is the time of enqueuing a request of cluster.
Parameter: None

ENQUEUE: CLUSTER BROADCAST REQUEST

It is the time of enqueuing a broadcast request of cluster.
Parameter: None

DEQUEUE: CLUSTER RESPONSE

It is the time of dequeuing a response of cluster.
Parameter: None

SEND: CDISPATCHER

It is the time of sending in cluster CDISPATCHER.
Parameter: None

RECV: CDISPATCHER

It is the time of receiving in cluster CDISPATCHER.
Parameter: None

GMASTER: ARCHIVE LOG

It is the time of processing archive logs in gmaster.
Parameter: None

GMASTER: CHECKPOINT

It is the time of processing checkpoints in gmaster.
Parameter: None

GMASTER: IO SLAVE

It is the time of processing IO slave in gmaster.
Parameter: None

GMASTER: LOG FLUSH

It is the time of processing archive logs in gmaster.
Parameter: None

GMASTER: PAGE FLUSH

It is the time of processing page flush in gmaster.
Parameter: None

WRITE: TRACE LOG

It is the time of writing trace logs.
Parameter: None

WRITE: COPY ARCHIVING LOG

It is the time of copying archiving logs.
Parameter: None

WRITE: BACKUP CTRL FILE

It is the time of backing up control files.
Parameter: None

WRITE: RESTORE CTRL FILE

It is the time of restoring the control file.
Parameter: None

READ: ARCHIVE LOG

It is the time of reading archive log files.
Parameter: None

READ: CTRL FILE

It is the time of reading the control file.
Parameter: None

WRITE: LOG FILE

It is the time of writing log files.
Parameter: None

WRITE: PAGE FILE

It is the time of writing page files.
Parameter: None

WRITE: CTRL FILE

It is the time of writing the control file.
Parameter: None

WRITE: REMOVE DATA FILE

It is the time of removing the data file.
Parameter: None

WRITE: JOURNAL BUFFER

It is the time of writing journal buffers.
Parameter: None

READ: JOURNAL BUFFER

It is the time of reading journal buffers.
Parameter: None

WAIT TRANSACTION

It is the time of waiting for transactions.

Parameter	Description
wait transaction id	Transaction id for which to wait

WAIT OTHER TRANSACTION

It is the time of waiting for other transactions to be terminated.

Parameter	Description
wait transaction id	ID of the waiting transaction
target transaction id	ID of the transaction to be terminated

WAIT ENABLE LOGGING

It is the time of waiting until when the logging is available.
Parameter: None

WAIT LOG FLUSHER

It is the time of waiting for the log flusher.

Parameter	Description
send data size	Bytes of the data to be sent

WAIT PAGE FLUSHER

It is the time of waiting for the page flusher.

Parameter	Description
send data size	Bytes of the data to be sent

WAIT XA CONTEXT

It is the time of waiting for the XA context.
Parameter: None

LATCH: LOG BUFFER

It is the time of waiting for the log buffer latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: PROCESS MANAGER

It is the time of waiting for the process manager latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: ENV MGR

It is the time of waiting for the env manager latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: SESSION ENV MGR

It is the time of waiting for the session env manager latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: PCH

It is the time of waiting for the Page Control Header (PCH) latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: PAGE

It is the time of waiting for the page latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: PENDING LOG

It is the time of waiting for the pending log latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: ALLOC TRANS

It is the time of waiting for the allocate transaction latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: UNDO SEGMENT

It is the time of waiting for the undo segment latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: CLUSTER LOCATION

It is the time of waiting for the cluster location latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: DICT HASH ELEMENT AGING

It is the time of waiting for the dict hash element aging latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: DICT HASH RELATED AGING

It is the time of waiting for the dict hash related aging latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: FILE MANAGER

It is the time of waiting for the file manager latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: TRACE LOG

It is the time of waiting for the trace log latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: STATIC HASH

It is the time of waiting for the static hash latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: STATIC HASH BUCKET

It is the time of waiting for the static hash bucket latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: SQL HANDLE

It is the time of waiting for SQL handle latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: XA CONTEXT HASH

It is the time of waiting for XA context hash latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: PLAN CLOCK

It is the time of waiting for the plan clock latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: XA CONTEXT

It is the time of waiting for XA context latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: MEM CONTROLLER

It is the time of waiting for the memory controller latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: DYNAMIC MEM

It is the time of waiting for the dynamic memory latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: PROPERTY

It is the time of waiting for the property latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: ATTACH SHM

It is the time of waiting for the attack shared memory latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: BACKUP TBS

It is the time of waiting for the backup tablespace latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: DATABASE COMPONENT

It is the time of waiting for the database component latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: TABLESPACE

It is the time of waiting for the tablespace latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: BACKUP DATABASE

It is the time of waiting for the backup database latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: JOURNAL BUFFER

It is the time of waiting for the journal buffer latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: JOURNAL BUFFER ENTRY

It is the time of waiting for the journal buffer entry latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: JOURNAL WRITE BUFFER

It is the time of waiting for the journal write buffer latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: LOCK ITEM

It is the time of waiting for the lock item latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: RECORD HASH

It is the time of waiting for the record hash latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: DEADLOCK

It is the time of waiting for the deadlock latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: SEQUENCE

It is the time of waiting for the sequence latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: LOG STREAM

It is the time of waiting for the log stream latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: BUILD AGABLE SCN

It is the time of waiting for the build agable SCN latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: TRANSACTION TABLE

It is the time of waiting for the transaction table latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: SESSION LINK HASH

It is the time of waiting for the session link hash latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: ALLOC XA CONTEXT

It is the time of waiting for the allocate XA context latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: SEQUENCE GLOBALX

It is the time of waiting for the sequence global latch X.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: SEQUENCE GLOBALY

It is the time of waiting for the sequence global latch Y.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: TRANSACTION LOG FILE

It is the time of waiting for the transaction logfile latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

ASYNC RESPONSE

It is the time of waiting for the response from the remote node in the cluster environment.
Parameter: None

ASYNC TRANSACTION

It is the time of waiting for the response for the ASYNC TRANSACTION's COMMIT from the remote node in the cluster environment.
Parameter: None

ASYNC COMMIT

N/A

GMASTER: BUFFER FLUSH

It is the time of executing the buffer flush by the io slave of gmaster.
Parameter: None

LATCH: BUFFER HASH BUCKET

It is the time of waiting for the buffer hash bucket's latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

READ: PAGE FILE

It is the time of reading pages from the disk to read the disk tablespace's pages.
Parameter: None

LATCH: BUFFER CHECKPOINT LIST

N/A

WRITE: CHANGE TRACKING FILE

It is the time of writing the disk tablespace's change tracking file.
Parameter: None

WAIT FREE BUFFER

It is the time of waiting for the free buffer, to read the pages in the disk tablespace.
Parameter: None

GLOBAL SEQUENCE: LOCK AND QUERY

It is the time of waiting for the global latch by all members to synchronize the global sequence in cluster, and the time of waiting to get the latest information of the global sequence.
Parameter: None

GLOBAL SEQUENCE: SYNC

It is the time of synchronizing the global sequence in cluster.
Parameter: None

LATCH: SEQUENCE GLOBAL NEXT

It is the time of waiting for the sequence global next latch in cluster.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

LATCH: BUFFER LRU LIST

It is the time of waiting for the buffer lru list latch.

Parameter	Description
address	The address of the latch for which the process is waiting
tries	A count of the number of times the process tried to get the latch

WAIT DIRTY PAGE LIMIT

It is the time of waiting until the number of dirty pages in the system becomes smaller than **BUFFER_DIRTY_PAGE_LIMIT** to access the pages cached in the buffer.
Parameter: None

WAIT BUFFER READ COMPLETE

It is the time of waiting for the disk read to be completed to access the page of the disk tablespace.
Parameter: None

Appendix C.

Open Source License

Apache License, Version 2.0

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sectio ns 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting th e License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled b y, or are under common control with that entity. For the purposes of this definition, "control" means (i) t he power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial o wnership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to softw are source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source fo rm, including but not limited to compiled object code, generated documentation, and conversions to oth er media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the L icense, as indicated by a copyright notice that is included in or attached to the work (an example is provid ed in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived fr om) the Work and for which the editorial revisions, annotations, elaborations, or other modifications repr esent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any mo difications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licens or for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to subm it on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of el ectronic, verbal, or written communication sent to the Licensor or its representatives, including but not li

mited to communication on electronic mailing lists, source code control systems, and issue tracking syste ms that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the W ork, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution ha s been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, w orldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Deriv ative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License.

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, w orldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent licen se to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such licens e applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contributio n(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counter claim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes dire ct or contributory patent infringement, then any patent licenses granted to You under this License for tha t Work shall terminate as of the date such litigation is filed.

4. Redistribution.

You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with o r without modifications, and in Source or Object form, provided that You meet the following conditions:

1. You must give any other recipients of the Work or Derivative Works a copy of this License; and

2. You must cause any modified files to carry prominent notices stating that You changed the files; and

3. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, paten t, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

4. If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that Yo u distribute must include a readable copy of the attribution notices contained within such NOTICE fil e, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of t he following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display gen

erated by the Derivative Works, if and wherever such third-party notices normally appear. The conte nts of the NOTICE file are for informational purposes only and do not modify the License. You may a dd Your own attribution notices within Derivative Works that You distribute, alongside or as an adde ndum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or differe nt license terms and conditions for use, reproduction, or distribution of Your modifications, or for any suc h Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions.

Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work b y You to the Licensor shall be under the terms and conditions of this License, without any additional term s or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any se parate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks.

This License does not grant permission to use the trade names, trademarks, service marks, or product na mes of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty.

Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contribut or provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIN D, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INF RINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible f or determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability.

In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unl ess required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or conseq uential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malf unction, or any and all other commercial damages or losses), even if such Contributor has been advised o f the possibility of such damages.

9. Accepting Warranty or Additional Liability.

While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with thi s License. However, in accepting such obligations, You may act only on Your own behalf and on Your sol e responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor b y reason of your accepting any such warranty or additional liability.