Data Warehouse Service 8.3.0

Tools Guide

Issue	10
Date	2024-12-09

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are the property of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei Cloud and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Huawei Cloud Computing Technologies Co., Ltd.

Address: Huawei Cloud Data Center Jiaoxinggong Road
Qianzhong Avenue
Gui'an New District
Gui Zhou 550029
People's Republic of China

Website: https://www.huaweicloud.com/intl/en-us/

Contents

1 Overview

This document describes how to use GaussDB(DWS) tools, including client tools, as shown in **Table 1-1**, and server tools, as shown in **Table 1-2**.

The client tools can be obtained by referring to **Downloading Related Tools**.

The server tools are stored in the **$GPHOME/script** and **$GAUSSHOME/bin** paths on the database server.

Table 1-1 Client Tools

Tool	Description
gsql	A command-line interface (CLI) SQL client tool running on the Linux OS. It is used to connect to the database in a GaussDB(DWS) cluster and perform operation and maintenance on the database.
Data Studio	A client tool used to connect to a database. It provides a GUI for managing databases and objects, editing, executing, and debugging SQL scripts, and viewing execution plans. Data Studio can run on a 32-bit or 64-bit Windows OS. You can use it after decompression without installation.
GDS	Gauss Data Service (GDS) is a CLI tool running on the Linux OS. It efficiently handles the import and export of data using foreign tables. The GDS tool package needs to be installed on the server where the data source file is located. This server is called the data server or the GDS server.
DSC	Database Schema Convertor (DSC) is a CLI tool running on Linux or Windows. It migrates SQL scripts in Teradata or Oracle databases to SQL scripts applicable to GaussDB(DWS), enabling GaussDB(DWS) database rebuilding. DSC runs on the Linux OS. You can use it after decompression without installation.

Table 1-2 Server Tools

Tool	Description
gs_dump	gs_dump exports database information, such as the complete and consistent data of database objects (including databases, schemas, tables, and views), without affecting the normal access of users to the database.
gs_dumpall	gs_dumpall exports database information, such as the complete and consistent data of database objects, without affecting the normal access of users to the database.
gs_restore	gs_restore is a tool provided by GaussDB(DWS) to import data that is exported using gs_dump. It can also be used to import files that were exported using **gs_dump**.
gds_check	gds_check is used to check the GDS deployment environment, including the OS parameters, network environment, and disk usage. It also supports the recovery of system parameters. This helps detect potential problems during GDS deployment and running, improving the execution success rate.
gds_install	gds_install is a script tool used to install GDS in batches, improving GDS deployment efficiency.
gds_uninstall	gds_uninstall is a script tool used to uninstall GDS in batches.
gds_ctl	gds_ctl is a script tool used for starting or stopping GDS service processes in batches. You can start or stop GDS service processes, which use the same port, on multiple nodes at a time, and set a daemon for each GDS process during the startup.

2 Downloading Related Tools

CLI Client (Containing the GDS Package)

NOTE

8.0.x, 8.1.x, and 8.2.x in the tool download addresses in this section refer to the cluster version instead of the tool version.

Table 2-1 gsql download links

OS Type	Applicable OS	Download Link	Verification File
Microsoft Windows	Microsoft Windows x86_64: • Windows 7 or later • Windows Server 2008 or later	dws_8.1.x_gsql_for_windows.zip	dws_8.1.x_gsql_for_windows.zip.sha256
		dws_8.2.x_gsql_for_windows.zip	dws_8.2.x_gsql_for_windows.zip.sha256
Redhat x86_64	RHEL 6.4~7.6	dws_client_8.2.x_redhat_x64.zip	dws_client_8.2.x_redhat_x64.zip.sha256
		dws_client_8.1.x_redhat_x64.zip	dws_client_8.1.x_redhat_x64.zip.sha256
		dws_client_8.0.x_redhat_x64.zip	dws_client_8.0.x_redhat_x64.zip.sha256
SUSE x86_64	SLES 11.1~11.4, SLES 12.0~12.3	dws_client_8.2.x_suse_x64.zip	dws_client_8.2.x_suse_x64.zip.sha256
		dws_client_8.1.x_suse_x64.zip	dws_client_8.1.x_suse_x64.zip.sha256

OS Type	Applicable OS	Download Link	Verification File
		dws_client_8.0.x_suse_x64.zip	dws_client_8.0.x_suse_x64.zip.sha256
Euler Kunpeng_64	EulerOS 2.0 SP8	dws_client_8.1.x_euler_kunpeng_x64.zip	dws_client_8.1.x_euler_kunpeng_x64.zip.sha256
Redhat Kunpeng_64	CentOS-7.6-aarch64 and NeoKylin-7.6-aarch64 (adapted to Kunpeng 920 CPU)	dws_client_8.1.x_redhat_kunpeng_x64.zip	dws_client_8.1.x_redhat_kunpeng_x64.zip.sha256

Data Studio GUI client

Table 2-2 Data Studio download links

Applicable OS	Download Link	Verification File
Windows x64	Data_Studio_8.2.x_64.zip	Data_Studio_8.2.x_64.zip.sha256
	Data_Studio_8.1.x_64.zip	Data_Studio_8.1.x_64.zip.sha256
	Data_Studio_8.0.x_64.zip	Data_Studio_8.0.x_64.zip.sha256
Windows x86	Data_Studio_8.2.x_32.zip	Data_Studio_8.2.x_32.zip.sha256
	Data_Studio_8.1.x_32.zip	Data_Studio_8.1.x_32.zip.sha256
	Data_Studio_8.0.x_32.zip	Data_Studio_8.0.x_32.zip.sha256

DSC migration tool

The DSC migration tool is used for GaussDB(DWS) clusters of version 1.7.1 or later.

Table 2-3 DSC download link

Applicable OS	Download Link	Download Verification File
For details, see **Operating Environment**.	**DSC_8.3.1.111.zip**	**DSC_8.3.1.111.zip.sha256**

JDBC Driver

GaussDB(DWS) supports the open source JDBC driver: PostgreSQL JDBC driver 9.3-1103 or later.

Table 2-4 JDBC driver download address

Driver	Download Link	Download Verification File
DWS JDBC Driver	**dws_9.1.x_jdbc_driver.zip**	**dws_9.1.x_jdbc_driver.zip.sha256**
	dws_8.3.x_jdbc_driver.zip	**dws_8.3.x_jdbc_driver.zip.sha256**
	dws_8.2.x_jdbc_driver.zip	**dws_8.2.x_jdbc_driver.zip.sha256**
	dws_8.1.x_jdbc_driver.zip	**dws_8.1.x_jdbc_driver.zip.sha256**
DWS ARM JDBC Driver	**dws_euler_kunpeng_jdbc.zip**	**dws_euler_kunpeng_jdbc.zip.sha256**

ODBC Driver

GaussDB(DWS) also supports the open source ODBC driver: PostgreSQL ODBC 09.01.0200 or later.

Table 2-5 ODBC driver download address

Applicable OS	Download Link	Download Verification File
Microsoft Windows	**dws_8.2.x_odbc_driver_for_windows.zip**	**dws_8.2.x_odbc_driver_for_windows.zip.sha256**
	dws_8.1.x_odbc_driver_for_windows.zip	**dws_8.1.x_odbc_driver_for_windows.zip.sha256**
EulerOS Arm	**dws_8.2.x_odbc_driver_for_arm_euler.zip**	**dws_8.2.x_odbc_driver_for_arm_euler.zip.sha256**

Applicable OS	Download Link	Download Verification File
	dws_8.1.x_odbc_driver_for_arm_euler.zip	dws_8.1.x_odbc_driver_for_arm_euler.zip.sha256
Red Hat Arm	dws_8.2.x_odbc_driver_for_arm_redhat.zip	dws_8.2.x_odbc_driver_for_arm_redhat.zip.sha256
	dws_8.1.x_odbc_driver_for_arm_redhat.zip	dws_8.1.x_odbc_driver_for_arm_redhat.zip.sha256
Redhat x86_64	dws_8.2.x_odbc_driver_for_x86_redhat.zip	dws_8.2.x_odbc_driver_for_x86_redhat.zip.sha256
	dws_8.1.x_odbc_driver_for_x86_redhat.zip	dws_8.1.x_odbc_driver_for_x86_redhat.zip.sha256
SUSE x86_64	dws_8.2.x_odbc_driver_for_x86_suse.zip	dws_8.1.x_odbc_driver_for_x86_suse.zip.sha256
	dws_8.1.x_odbc_driver_for_x86_suse.zip	dws_8.1.x_odbc_driver_for_x86_suse.zip.sha256

3 gsql

3.1 Overview

Basic Functions

- **Connect to the database**: Use the gsql client to remotely connect to the GaussDB(DWS) database.

  NOTE

  If the gsql client is used to connect to a database, the connection timeout period will be 5 minutes. If the database has not correctly set up a connection and authenticated the identity of the client within this period, gsql will time out and exit.

  To resolve this problem, see **Troubleshooting**.
- **Run SQL statements**: Interactively entered SQL statements and specified SQL statements in a file can be run.
- **Run meta-commands**: Meta-commands help the administrator view database object information, query cache information, format SQL output, and connect to a new database. For details about meta-commands, see **Meta-Command Reference**.

Advanced Features

Table 3-1 lists the advanced features of gsql.

Table 3-1 Advanced features of gsql

Feature	Description
Variable	gsql provides a variable feature that is similar to the **shell** command of Linux. The following **\set** meta-command of gsql can be used to set a variable: **\set** *varname value* To delete a variable, run the following command: **\unset** *varname* **NOTE** • A variable is a key-value pair. The value length is determined by the special variable **VAR_MAX_LENGTH**. For details, see **Table 3-2**. • Variable names must consist of case-sensitive letters (including non-Latin letters), digits, and underscores (_). • If the **\set** *varname* meta-command (without the second parameter) is used, the variable is set without a value specified. • If the **\set** meta-command without parameters is used, values of all variables are displayed. For details about variable examples and descriptions, see **Variable**.
SQL substitution	Common SQL statements can be set to variables using the variable feature of gsql to simplify operations. For details about SQL substitution examples and descriptions, see **SQL substitution**.
Customized prompt	Prompts of gsql can be customized. Prompts can be modified by changing the reserved variables of gsql: *PROMPT1*, *PROMPT2*, and *PROMPT3*. These variables can be set to customized values or the values predefined by gsql. For details, see **Prompt**.
Client operation history record	gsql records client operation history. This function is enabled by specifying the **-r** parameter when a client is connected. The number of historical records can be set using the **\set** command. For example, **\set HISTSIZE 50** indicates that the number of historical records is set to **50**. **\set HISTSIZE 0** indicates that the operation history is not recorded. **NOTE** • The default number of historical records is **32**. The maximum number of historical records is **500**. If interactively entered SQL statements contain Chinese characters, only the UTF-8 encoding environment is supported. • For security reasons, the records containing sensitive words, such as **PASSWORD** and **IDENTIFIED**, are regarded sensitive and not recorded in historical information. This indicates that you cannot view these records in command output histories.

- Variable

  To set a variable, run the **\set** meta-command of gsql. For example, to set variable *foo* to **bar**, run the following command:

  ```
  \set foo bar
  ```


To quote the value of a variable, add a colon (:) before the variable. For example, to view the value of variable *foo*, run the following command:

```
\echo :foo
bar
```

This variable quotation method is suitable for regular SQL statements and meta-commands.

When the CLI parameter **--dynamic-param** (for details, see **Table 3-11**) is used or the special variable **DYNAMIC_PARAM_ENABLE** (for details, see **Table 3-2**) is set to **true**, you can execute the SQL statement to set the variable. The variable name is the column name in the SQL execution result and can be referenced using **${}**. For example:

```
\set DYNAMIC_PARAM_ENABLE true
SELECT 'Jack' AS "Name";
 Name
------
 Jack
(1 row)

\echo ${Name}
Jack
```

In the preceding example, the SELECT statement is used to set the **Name** variable, and the **${}** referencing method is used to obtain the value of the **Name** variable. In this example, the special variable **DYNAMIC_PARAM_ENABLE** controls this function. You can also use the CLI parameter **--dynamic-param** to control this function, for example, **gsql -d postgres -p 25308 --dynamic-param -r**.

NOTE

- Do not set variables when the SQL statement execution fails.
- If the SQL statement execution result is empty, set the column name as a variable and assign it with an empty string.
- If the SQL statement execution result is a record, set the column name as a variable and assign it with the corresponding string.
- If the SQL statement execution result contains multiple records, set the column name as a variable concatenated by specific characters, and then assign the value to the variable. The special variable **RESULT_DELIMITER** (for details, see **Table 3-2**) determines the specific character. The default delimiter is a comma (,).

Examples of setting variables by executing SQL statements:

```
\set DYNAMIC_PARAM_ENABLE true
CREATE TABLE student (id INT, name VARCHAR(32)) DISTRIBUTE BY HASH(id);
CREATE TABLE
INSERT INTO student VALUES (1, 'Jack'), (2, 'Tom'), (3, 'Jerry');
INSERT 0 3
-- Do not set variables when the SQL statement execution fails.
SELECT id, name FROM student ORDER BY idi;
ERROR:  column "idi" does not exist
LINE 1: SELECT id, name FROM student ORDER BY idi;
                                              ^
\echo ${id} ${name}
${id} ${name}

-- If the execution result contains multiple records, use specific characters to concatenate the values.
SELECT id, name FROM student ORDER BY id;
 id | name
----+-------
  1 | Jack
  2 | Tom
  3 | Jerry
(3 rows)
```



```
\echo ${id} ${name}
1,2,3 Jack,Tom,Jerry

-- If the execution result contains only one record, execute the following statement to set the variable:
SELECT id, name FROM student where id = 1;
 id | name
----+------
  1 | Jack
(1 row)

\echo ${id} ${name}
1 Jack

-- If the execution result is empty, assign the variable with an empty string as follows:
SELECT id, name FROM student where id = 4;
 id | name
----+------
(0 rows)

\echo ${id} ${name}
```

gsql pre-defines some special variables and plans the values of these variables. To ensure compatibility with later versions, do not use these variables for other purposes. For details about all special variables, see **Table 3-2**.

NOTE

- All the special variables consist of uppercase letters, digits, and underscores (_).
- To view the default value of a special variable, run the **\echo :***varname* meta-command, for example, **\echo :***DBNAME*.

Table 3-2 Setting special variables

Variable	Setting Method	Description
DBNAME	\set DBNAME *dbname*	Specifies the name of a connected database. This variable is set again when a database is connected.
ECHO	\set ECHO all | queries	• If this variable is set to **all**, only the query information is displayed. This has the same effect as specifying the **-a** parameter when gsql is used to connect to a database. • If this variable is set to **queries**, the command line and query information are displayed. This has the same effect as specifying the **-e** parameter when gsql is used to connect to a database.

Variable	Setting Method	Description
ECHO_HIDDEN	\set ECHO_HIDDEN on | off | noexec	When a meta-command (such as **\dg**) is used to query database information, the value of this variable determines the query behavior. ● If this variable is set to **on**, the query statements that are called by the meta-command are displayed, and then the query result is displayed. This has the same effect as specifying the **-E** parameter when gsql is used to connect to a database. ● If this variable is set to **off**, only the query result is displayed. ● If this variable is set to **noexec**, only the query information is displayed, and the query is not run.
ENCODING	\set ENCODING *encoding*	Specifies the character set encoding of the current client.
FETCH_COUNT	\set FETCH_COUNT *variable*	● If the value is an integer greater than **0**, for example, *n*, *n* lines will be selected from the result set to the cache and displayed on the screen when the **SELECT** statement is run. ● If this variable is not set or set to a value less than or equal to **0**, all results are selected at a time to the cache when the **SELECT** statement is run. **NOTE** Setting this variable to a proper value reduces memory usage. Generally, values from 100 to 1000 are proper.
HISTCONTROL	\set HISTCONTROL ignorespace | ignoredups | ignoreboth | none	● **ignorespace**: A line started with a space is not written to the historical record. ● **ignoredups**: A line that exists in the historical record is not written to the historical record. ● **ignoreboth**, **none**, or other values: All the lines read in interaction mode are saved in the historical record. **NOTE** **none** indicates that **HISTCONTROL** is not set.
HISTFILE	\set HISTFILE *filename*	Specifies the file for storing historical records. The default value is **~/.bash_history**.

Variable	Setting Method	Description
HISTSIZE	\set HISTSIZE *size*	Specifies the number of commands in the history command. The default value is **500**.
HOST	\set HOST *hostname*	Specifies the name of a connected host.
IGNOREEOF	\set IGNOREEOF *variable*	• If this variable is set to a number, for example, **10**, the first nine EOF characters (generally **Ctrl**+**C**) entered in gsql are neglected and the gsql program exits when the tenth **Ctrl**+**C** is entered. • If this variable is set to a non-numeric value, the default value is **10**. • If this variable is deleted, gsql exits when an EOF is entered.
LASTOID	\set LASTOID *oid*	Specifies the last OID, which is the value returned by an **INSERT** or **lo_import** command. This variable is valid only before the output of the next SQL statement is displayed.
ON_ERROR_ROLLBACK	\set ON_ERROR_ROLLBACK on | interactive | off	• If the value is **on**, an error that may occur in a statement in a transaction block is ignored and the transaction continues. • If the value is **interactive**, the error is ignored only in an interactive session. • If the value is **off** (the default value), the error triggers the rollback of the transaction block. In **on_error_rollback-on** mode, a **SAVEPOINT** is set before each statement of a transaction block, and an error triggers the rollback of the transaction block.
ON_ERROR_STOP	\set ON_ERROR_STOP on | off	• **on**: specifies that the execution stops if an error occurs. In interactive mode, gsql returns the output of executed commands immediately. • **off** (default value): specifies that an error, if occurring during the execution, is ignored, and the execution continues.
PORT	\set PORT *port*	Specifies the port number of a connected database.
USER	\set USER *username*	Specifies the connected database user.

Variable	Setting Method	Description
VERBOSITY	\set VERBOSITY terse | default | verbose	This variable can be set to **terse**, **default**, or **verbose** to control redundant lines of error reports. • **terse**: Only critical and major error texts and text locations are returned (which is suitable for single-line error information). • **default**: Critical and major error texts and text locations, error details, and error messages (possibly involving multiple lines) are all returned. • **verbose**: All error information is returned.
VAR_NOT_FOUND	\set VAR_NOT_FOUND default | null | error	You can set this parameter to **default**, **null**, or **error** to control the processing mode when the referenced variable does not exist. • **default**: Do not replace the variable and retain the original character string. • **null**: Replace the original character string with an empty character string. • **error**: Output error information and retain the original character string.
VAR_MAX_LENGTH	\set VAR_MAX_LENGTH *variable*	Specifies the variable value length. The default value is 4096. If the length of a variable value exceeds the specified parameter value, the variable value is truncated and an alarm is generated.
ERROR_LEVEL	\set ERROR_LEVEL transaction | statement	Indicates whether a transaction or statement is successful or not. Value options: **transaction** or **statement**. Default value: **transaction** • **statement**: ERROR records whether the previous SQL statement is executed successfully. • **transaction**: ERROR records whether the previous SQL statement is successfully executed or whether an error occurs during the execution of the previous transaction.

Variable	Setting Method	Description
ERROR	\set ERROR true | false	Indicates whether the previous SQL statement is successfully executed or whether an error occurs during the execution of the previous transaction. **false**: succeeded. **true**: failed. default value: **false** The setting can be updated by executing SQL statements. You are not advised to manually set this parameter.
LAST_ERROR_SQLSTATE	\set LAST_ERROR_SQLSTATE state	Error code of the previously failed SQL statement execution. The default value is **00000**. The setting can be updated by executing SQL statements. You are not advised to manually set this parameter.
LAST_ERROR_MESSAGE	\set LAST_ERROR_MESSAGE message	Error message of the previously failed SQL statement execution. The default value is an empty string. The setting can be updated by executing SQL statements. You are not advised to manually set this parameter.
ROW_COUNT	\set ROW_COUNT count	• If ERROR_LEVEL is set to **statement**, this parameter indicates the number of rows returned after the previous SQL statement is executed or the number of affected rows. • If ERROR_LEVEL is set to **transaction** and an internal error occurs when a transaction ends, this parameter indicates the number of rows returned by the last SQL statement of the transaction or the number of affected rows. Otherwise, this parameter indicates the number of rows returned by the last SQL statement or the number of affected rows. If the SQL statement fails to be executed, set this parameter to **0**. The default value is **0**. The setting can be updated by executing SQL statements. You are not advised to manually set this parameter.

Variable	Setting Method	Description
SQLSTATE	\set SQLSTATE state	• If ERROR_LEVEL is set to **statement**, this parameter indicates the status code of the previous SQL statement. • If ERROR_LEVEL is set to **transaction** and an internal error occurs when a transaction ends, this parameter indicates the status code of the last SQL statement in the transaction. Otherwise, this parameter indicates the status code of the previous SQL statement. The default value is **00000**. The setting can be updated by executing SQL statements. You are not advised to manually set this parameter.
LAST_SYS_CODE	\set LAST_SYS_CODE code	Returned value of the previous system command execution. The default value is **0**. The setting can be updated by using the meta-command **\!** to run the system command. You are not advised to manually set this parameter.
DYNAMIC_PARAM_ENABLE	\set DYNAMIC_PARAM_ENABLE true | false	Controls the generation of variables and the variable referencing method **${}** during SQL statement execution. The default value is **false**. • **true**: Generate variables when executing SQL statements, and support the **${}** variable referencing method. • **false**: Do not generate variables when executing SQL statements, and the **${}** variable referencing method is not supported either.

Variable	Setting Method	Description
CONVERT_QUOTE_IN_DYNAMIC_PARAM	\set CONVERT_QUOTE_IN_DYNAMIC_PARAM true | false	Specifies whether to escape single quotation marks, double quotation marks, and backslashes during dynamic variable parsing. The default value is **true**. • **true**: Indicates that during dynamic variable parsing, single quotation marks, double quotation marks, and backslashes (\) need to be escaped, and SQL substitution automatically escapes quotation marks and backslashes in variables. • **false**: Indicates that during dynamic variable parsing single quotation marks, double quotation marks, and backslashes (\) do not need to be escaped. SQL substitution does not process strings in variables. You need to manually escape them as needed. For details about the usage example, see **CONVERT_QUOTE_IN_DYNAMIC...**.
RESULT_DELIMITER	\set RESULT_DELIMITER delimiter	Controls the delimiter used for concatenating multiple records when variables are generated during SQL statement execution. The default delimiter is comma (,).

Variable	Setting Method	Description
COMPARE_STRATEGY	\set COMPARE_STRATEGY default | natural | equal	Used to specify the value comparison policy of the **\if** expression. The default value is **default**. • **default**: Specifies the default comparison policy. Only strings or numbers can be compared, and strings cannot be compared with numbers. Parameters inside single quotation marks (') are identified as strings, and parameters outside single quotation marks (') are identified as numbers. • **natural**: The default comparison policy is supported, and parameters that contain dynamic variables are identified as strings. When one side of the comparison operator is a number, try to convert the other side to a number, and then compare the numbers on both sides. If the conversion fails, an error is reported and the comparison result is false. • **equal**: Only the equality comparison is supported. The comparison is performed based on strings. For details, see **\if conditional block comparison rules and examples**.
COMMAND_ERROR_STOP	\set COMMAND_ERROR_STOP on | off	Determines whether to report the error and stop executing the meta-command when an error occurs during meta-command execution. By default, the meta-command execution is not stopped. For details, see the **example of using COMMAND_ERROR_STOP**.
INCOMPLETE_QUERY_ERROR	\set INCOMPLETE_QUERY_ERROR true | false	Indicates whether incomplete SQL statements are checked before process control meta-commands, such as **\if** **\goto** and **\for**, are executed. The default value is **false**. If a statement contains the incomplete characters, such as **(**,**'**,**"**,and **$$**, or does not end with a semicolon (;), an error is reported and the system exits. For details about the usage example, see **An example of using the special variable INCOMPLETE_QUERY_ERROR**.

Variable	Setting Method	Description
INCOMPLETE_QUERY_ERROR_CODE	\set INCOMPLETE_QUERY_ERROR_CODE 12	If **INCOMPLETE_QUERY_ERROR** is set to **true** and an incomplete statement is detected, an error is reported and the statement exits. You can set **INCOMPLETE_QUERY_ERROR_CODE** to configure the exit code. The default value is **-1**. For details about the usage example, see **An example of using the special variable INCOMPLETE_QUERY_ERROR_CODE**.

- The following is an example of using the special variables ERROR_LEVEL and ERROR:

When ERROR_LEVEL is set to **statement**, ERROR records whether the previous SQL statement is executed successfully. In the following example, when an SQL execution error occurs in a transaction and the transaction ends, the value of ERROR is **false**. In this case, ERROR only records whether the previous SQL statement ends successfully.

```
\set ERROR_LEVEL statement
begin;
BEGIN
select 1 as ;
ERROR:  syntax error at or near ";"
LINE 1: select 1 as ;
                    ^
end;
ROLLBACK
\echo :ERROR
false
```

When ERROR_LEVEL is set to **transaction**, ERROR can be used to capture SQL execution errors in a transaction. In the following example, when an SQL execution error occurs in a transaction and the transaction ends, the value of ERROR is **true**.

```
\set ERROR_LEVEL transaction
begin;
BEGIN
select 1 as ;
ERROR:  syntax error at or near ";"
LINE 1: select 1 as ;
                    ^
end;
ROLLBACK
\echo :ERROR
true
```

- The following is an example of using the special variable **COMMAND_ERROR_STOP**:

When **COMMAND_ERROR_STOP** is set to **on** and an error occurs during the meta-command execution, the error is reported and the meta-command execution is stopped. When this function is enabled, the execution error of the meta-command can be effectively detected.

When **COMMAND_ERROR_STOP** is set to **off** and an error occurs during the meta-command execution, related information is printed and the script continues to be executed.

```
\set COMMAND_ERROR_STOP on
\i /home/omm/copy_data.sql

select id, name from student;
```

When **COMMAND_ERROR_STOP** in the preceding script is set to **on**, an error message is displayed after the error is reported, and the script execution is stopped.

```
gsql:test.sql:2: /home/omm/copy_data.sql: Not a directory
```

When **COMMAND_ERROR_STOP** is set to **off**, an error message is displayed after the error is reported, and the **SELECT** statement continues to be executed.

```
gsql:test.sql:2: /home/omm/copy_data.sql: Not a directory
 id | name
----+------
  1 | Jack
(1 row)
```

– An example of using the special variable **INCOMPLETE_QUERY_ERROR**:

When **INCOMPLETE_QUERY_ERROR** is set to **true**, incomplete SQL statements are detected before process control meta-commands such as \if \goto \for, an error is reported, and the system exits.

NOTE

The following types of incomplete statements are detected:

- The SQL statement does not end with a semicolon (;).
- The brackets in the SQL statement do not match.
- The single quotation marks in the SQL statement do not match.
- The double quotation marks in the SQL statement do not match.
- **$$** in the SQL statement does not match.

Detection method:

- If the SQL statement does not end with a semicolon (;) or the brackets do not match, analyze the character string before running the process control meta-commands such as \if \goto \for, and delete the C language style comment (/**/) and any blank character string (including a space, tab character, or form feed. It is equivalent to [\f\n\r\t\v]). If there are other characters, an error is reported, indicating that there are remaining incomplete SQL statements, and the system exits.
- Single quotation marks, double quotation marks, and $$ do not match. If the \if, \elif, \else, \endif, \goto, \label, \for, \loop, \exit-for, and \end-for meta-commands are detected, an error is reported and the system exits.

The following is an example of SQL statements that do not end with a semicolon (;):

```
\set INCOMPLETE_QUERY_ERROR true
select 1 as id
\if ${ERROR}
    \echo 'find error'
    \q 12
\endif
```

In the preceding example, an SQL statement that does not end with a semicolon (;) exists before the \if meta-command. In this case, an error is reported when the \if command is executed.


```
$ gsql -X -d postgres -p 13500 --dynamic-param -a -f test.sql
\set INCOMPLETE_QUERY_ERROR true
select 1 as id
\if ${ERROR}
gsql:test.sql:3: ERROR: An incomplete SQL statement exists before the \if command.
gsql:test.sql:3: DETAIL: The SQL statement may not end with a semicolon. Please check.
```

The following is an example of SQL statements in which the brackets do not match:

```
\set INCOMPLETE_QUERY_ERROR true
insert into student values (1, 'jack';
\if ${ERROR}
    \echo 'find error'
    \q 12
\endif
```

In the preceding example, an SQL statement in which the brackets do not match exists before the \if meta-command. In this case, an error is reported when the \if command is executed.

```
$ gsql -X -d postgres -p 13500 --dynamic-param -a -f test.sql
\set INCOMPLETE_QUERY_ERROR true
insert into student values (1, 'jack';
\if ${ERROR}
gsql:test.sql:3: ERROR: An incomplete SQL statement exists before the \if command.
gsql:test.sql:3: DETAIL: There may be an unmatched ( in the SQL statement. Please check.
```

The following is an example of SQL statements in which the single quotation marks do not match:

```
\set INCOMPLETE_QUERY_ERROR true
select 'jack as name;
\if ${ERROR}
    \echo 'find error'
    \q 12
\endif
```

In the preceding example, an SQL statement in which the single quotation marks do not match exists before the \if meta-command. In this case, an error is reported when the \if command is executed.

```
$ gsql -X -d postgres -p 13500 --dynamic-param -a -f test.sql
\set INCOMPLETE_QUERY_ERROR true
select 'jack as name;
\if ${ERROR}
gsql:test.sql:3: ERROR: An incomplete SQL statement exists before the \if command.
gsql:test.sql:3: DETAIL: There may be an unmatched ' in the SQL statement. Please check.
```

The following is an example of SQL statements in which the double quotation marks do not match:

```
\set INCOMPLETE_QUERY_ERROR true
select 10001 as "ID;
\if ${ERROR}
    \echo 'find error'
    \q 12
\endif
```

In the preceding example, an SQL statement in which the double-brackets are incomplete exists before the **\if** meta-command. In this case, an error is reported and the system exits when the **\if** command is executed.

```
$ gsql -X -d postgres -p 13500 --dynamic-param -a -f test.sql
\set INCOMPLETE_QUERY_ERROR true
select 10001 as "ID;
\if ${ERROR}
gsql:test.sql:3: ERROR: An incomplete SQL statement exists before the \if command.
gsql:test.sql:3: DETAIL: There may be an unmatched " in the SQL statement. Please check.
```

The following is an example of SQL statements in which $$ does not match:

```
\set INCOMPLETE_QUERY_ERROR true
create or replace function gsql_dollar_quote_test()
```



```
returns integer
as
$BODY$
declare
    query text;
    dest text;
begin
    query := 'select count(*) from pg_class';
    execute immediate query into dest;
end;
$BODY
language 'plpgsql' not fenced;
call gsql_dollar_quote_test();
\if ${ERROR}
    \echo 'find error'
    \q 12
\endif
```

In the preceding example, an SQL statement in which $$ does not match exists before the **\if** meta-command. In this case, an error is reported and the system exits when the **\if** command is executed.

```
$ gsql -X -d postgres -p 13500  --dynamic-param -a -f test.sql
\set INCOMPLETE_QUERY_ERROR true
create or replace function gsql_dollar_quote_test()
returns integer
as
$BODY$
declare
    query text;
    dest text;
begin
    query := 'select count(*) from pg_class';
    execute immediate query into dest;
end;
$BODY
language 'plpgsql' not fenced;
call gsql_dollar_quote_test();
\if ${ERROR}
gsql:test.sql:16: ERROR: An incomplete SQL statement exists before the \if command.
gsql:test.sql:16: DETAIL: There may be an unmatched $$ in the SQL statement. Please check.
```

– An example of using the special variable **INCOMPLETE_QUERY_ERROR_CODE**:

If **INCOMPLETE_QUERY_ERROR** is **true**, you can set **INCOMPLETE_QUERY_ERROR_CODE** to configure the exit code when an incomplete statement is detected. The following is an example:

```
\set INCOMPLETE_QUERY_ERROR true
\set INCOMPLETE_QUERY_ERROR_CODE 20
insert into student values (1, 'jack';
\if ${ERROR}
    \echo 'find error'
    \q 12
\endif
```

In the preceding test case, if **INCOMPLETE_QUERY_ERROR_CODE** is set to **20**, the \ifcommand will exit when an incomplete statement is detected, and the exit code is the value of **INCOMPLETE_QUERY_ERROR_CODE**.

```
$ gsql -X -d postgres -p 13500  --dynamic-param -a -f test.sql
\set INCOMPLETE_QUERY_ERROR true
\set INCOMPLETE_QUERY_ERROR_CODE 20
insert into student values (1, 'jack';
\if ${ERROR}
gsql:test.sql:4: ERROR: An incomplete SQL statement exists before the \if command.
gsql:test.sql:4: DETAIL: There may be an unmatched ( in the SQL statement. Please check.
$ echo $?
20
```


- SQL substitution

  gsql, like a parameter of a meta-command, provides a key feature that enables you to substitute a standard SQL statement for a gsql variable. gsql also provides a new alias or identifier for the variable. To replace the value of a variable using the SQL substitution method, add a colon (:) in front of the variable. For example:

```
\set foo 'HR.areaS'
select * from :foo;
 area_id |       area_name
---------+------------------------
       4 | Iron
       3 | Desert
       1 | Wood
       2 | Lake
(4 rows)
```

  The above command queries the **HR.areaS** table.

  NOTICE

  The value of a variable is copied character by character, and even an asymmetric quote mark or backslash (\) is copied. Therefore, the input content must be meaningful.

  – The following is an example of using the special variable **CONVERT_QUOTE_IN_DYNAMIC_PARAM**:

  If **CONVERT_QUOTE_IN_DYNAMIC_PARAM** is set to **true**, quotation marks and backslashes in variables are automatically escaped during SQL substitution.

```
 \set DYNAMIC_PARAM_ENABLE true
 \set CONVERT_QUOTE_IN_DYNAMIC_PARAM true
 select '"'abc'"\\' as "SpecialCharacters";
   test
-----------
 "'abc"\\
(1 row)

-- Single quotation marks are escaped, but still displayed in the result.
 select '${SpecialCharacters}' as "test";
   test
-----------
 "'abc"\\
(1 row)

-- Single quotation marks and backslashes are escaped, but still displayed in the result.
 select E'${SpecialCharacters}' as "test";
   test
-----------
 "'abc"\\
(1 row)

-- Double quotation marks are escaped, but still displayed in the result.
-- The column name contains characters other than letters, digits, and underscores (_). Therefore, an error occurred.
 select 'test' as "${SpecialCharacters}";
error while saving the value of "'abc"\\, please check the column name which can only contain upper and lower case letters, numbers and '_'.
 "'abc"\\
-----------
 test
(1 row)
```


When **CONVERT_QUOTE_IN_DYNAMIC_PARAM** is set to false, strings in the variable are not processed during SQL substitution. You need to manually escape the strings as needed

NOTE

You are advised to use the default value **true**.

During SQL substitution, single quotation marks need to be escaped in **''**, single quotation marks and backslashes need to be escaped in **E''**, and double quotation marks need to be escaped in **""**. Quotation marks and backslashes need to be handled based on variable positions. This makes the variable logics in SQL substitution complex and error-prone.

```
\set DYNAMIC_PARAM_ENABLE true
\set CONVERT_QUOTE_IN_DYNAMIC_PARAM false
select '""abc""\\' as "SpecialCharacters";
  test
-----------
 ""abc''\\
(1 row)

-- Single quotation marks are not escaped. The result contains only one single quotation mark.
select '${SpecialCharacters}' as "test";
  test
----------
 ""abc'\\
(1 row)

-- Single quotation marks and backslashes are not escaped. The result contains only one single quotation mark and one backslash.
select E'${SpecialCharacters}' as "test";
 test
---------
 ""abc'\
(1 row)

-- Double quotation marks are not escaped. The result contains only one double quotation mark.
-- The column name contains characters other than letters, digits, and underscores (_). Therefore, an error occurred.
select 'test' as "${SpecialCharacters}";
error while saving the value of "abc''\\, please check the column name which can only contain upper and lower case letters, numbers and '_'.
 "abc''\\
----------
 test
(1 row)
```

- Prompt

  The gsql prompt can be set using the three variables in **Table 3-3**. These variables consist of characters and special escape characters.

Table 3-3 Prompt variables

Variable	Description	Example

PROMPT1	Specifies the normal prompt used when gsql requests a new command.
	The default value of *PROMPT1* is:
	`%/%R%#`
		PROMPT1 can be used to change the prompt.
		• Change the prompt to **[local]**:
		`\set PROMPT1 %M`
		`[local:/tmp/gaussdba_mppdb]`
		• Change the prompt to **name**:
		`\set PROMPT1 name`
		`name`
		• Change the prompt to **=**:
		`\set PROMPT1 %R`
		`=`

PROMPT2	Specifies the prompt displayed when more command input is expected. For example, it is expected if a command is not terminated with a semicolon (;) or a quote (") is not closed.	*PROMPT2* can be used to display the prompt:
		`\set PROMPT2 TEST`
		`select * from HR.areaS TEST;`
		` area_id |  area_name`
		`---------+--------------------`
		`  1 | Wood`
		`  2 | Lake`
		`  4 | Iron`
		`  3 | Desert`
		`(4 rows))`

PROMPT3	Specifies the prompt displayed when the **COPY** statement (such as **COPY FROM STDIN**) is run and data input is expected.	*PROMPT3* can be used to display the **COPY** prompt.
		`\set PROMPT3 '>>>>'`
		`copy HR.areaS from STDIN;`
		`Enter data to be copied followed by a newline.`
		`End with a backslash and a period on a line by itself.`
		`>>>>1 aa`
		`>>>>2 bb`
		`>>>>\.`

The value of the selected prompt variable is printed literally. However, a value containing a percent sign (%) is replaced by the predefined contents depending on the character following the percent sign (%). For details about the defined substitutions, see **Table 3-4**.

Table 3-4 Defined substitutions

Symbol	Description
%M	Specifies the full host name (with domain name). The full name is [local] if the connection is over a Unix domain socket, or [local:/dir/name] if the Unix domain socket is not at the compiled default location.
%m	Specifies the host name truncated at the first dot. It is [local] if the connection is over a Unix domain socket.

Symbol	Description
%>	Specifies the number of the port that the host is listening on.
%n	Specifies the database session user name.
%/	Specifies the name of the current database.
%~	Is similar to **%/**. However, the output is tilde (~) if the database is your default database.
%#	Uses **#** if the session user is the database administrator. Otherwise, uses **>**.
%R	• Normally uses **=** for *PROMPT1*, but **^** in single-line mode and **!** if the session is disconnected from the database (which may occur if **\connect** fails). • For PROMPT2, the sequence is replaced by a hyphen (-), asterisk (*), single quotation mark ('), double quotation mark ("), or dollar sign ($), depending on whether gsql is waiting for more input, or the query is not terminated, or the query is in the /* ... */ the comment, quotation mark, or dollar sign extension.
%x	Specifies the transaction status. • An empty string when it is not in a transaction block • An asterisk (*) when it is in a transaction block • An exclamation mark (!) when it is in a failed transaction block • A question mark (?) when the transaction status is indeterminate (for example, indeterminate due to no connections).
%digits	Is replaced with the character with the specified byte.
%:name	Specifies the value of the *name* variable of gsql.
%command	Specifies command output, similar to ordinary "back-tick" ("^") substitution.
%[. . . %]	Prompts can contain terminal control characters which, for example, change the color, background, or style of the prompt text, or change the title of the terminal window. For example: potgres=> \set PROMPT1 '%[%033[1;33;40m%]%n@%/%R%[%033[0m%]%#' The output is a boldfaced (1;) yellow-on-black (33;40) prompt on VT100-compatible, color-capable terminals.

Environment Variables

Table 3-5 Environment variables related to gsql

Name	Description
COLUMNS	If **\set columns** is set to **0**, this parameter controls the width of the wrapped format. This width determines whether the width output mode is changed to a vertical bar format in automatic expansion mode.
PAGER	If the query result cannot be displayed within one page, the query result will be redirected to the command. You can use the **\pset** command to disable the pager. Typically, the **more** or **less** command is used for viewing the query result page by page. The default value is platform-associated. **NOTE** Display of the **less** command is affected by the *LC_CTYPE* environmental variable.
PSQL_EDITOR EDITOR VISUAL	The **\e** and **\ef** commands use the editor specified by the environment variables. Variables are checked according to the list sequence. The default editor on Unix is vi.

PSQL_EDITOR_LINENUMBER_ARG	When the **\e** or **\ef** command is used with a line number parameter, this variable specifies the command-line parameter used to pass the starting line number to the editor. For editors, such as Emacs or vi, this is a plus sign. A space is added behind the value of the variable if whitespace is required between the option name and the line number. For example:
	`PSQL_EDITOR_LINENUMBER_ARG = '+'`
	`PSQL_EDITOR_LINENUMBER_ARG='--line '`
	A plus sign (+) is used by default on Unix.

PSQLRC	Specifies the location of the user's .gsqlrc file.
SHELL	Has the same effect as the **\!** command.
TMPDIR	Specifies the directory for storing temporary files. The default value is **/tmp**.

3.2 Downloading the Client

GaussDB(DWS) provides client tool packages that match the cluster versions. You can download the desired client tool package on the GaussDB(DWS) management console.

The client tool package contains the following:

- **Linux database connection tool gsql and the script for testing sample data**

Linux gsql is a Linux command line client running in Linux. It is used to connect to the database in a data warehouse cluster.

The script for testing sample data is used when you start an example.

- **Windows gsql**

  Windows gsql is a command line client running on the Windows OS. It is used to connect to the database in a data warehouse cluster.

  NOTE

  Only 8.1.3.101 and later cluster versions can be downloaded from the console.

- **GDS tool package**

  Gauss Data Service (GDS) is a data service tool. You can use the GDS tool to import a data file in a common file system to the GaussDB(DWS) database. The GDS tool package must be installed on the server where the data source file is located. The server where the data source file is located is called a data server or GDS server.

Downloading the Client

Table 3-6 gsql download links

OS Type	Applicable OS	Download Link	Verification File
Microsoft Windows	Microsoft Windows x86_64: • Windows 7 or later • Windows Server 2008 or later	dws_8.1.x_gsql_for_windows.zip	dws_8.1.x_gsql_for_windows.zip.sha256
		dws_8.2.x_gsql_for_windows.zip	dws_8.2.x_gsql_for_windows.zip.sha256
Redhat x86_64	RHEL 6.4~7.6	dws_client_8.2.x_redhat_x64.zip	dws_client_8.2.x_redhat_x64.zip.sha256
		dws_client_8.1.x_redhat_x64.zip	dws_client_8.1.x_redhat_x64.zip.sha256
		dws_client_8.0.x_redhat_x64.zip	dws_client_8.0.x_redhat_x64.zip.sha256
SUSE x86_64	SLES 11.1~11.4, SLES 12.0~12.3	dws_client_8.2.x_suse_x64.zip	dws_client_8.2.x_suse_x64.zip.sha256
		dws_client_8.1.x_suse_x64.zip	dws_client_8.1.x_suse_x64.zip.sha256
		dws_client_8.0.x_suse_x64.zip	dws_client_8.0.x_suse_x64.zip.sha256

OS Type	Applicable OS	Download Link	Verification File
Euler Kunpeng_64	EulerOS 2.0 SP8	dws_client_8.1.x_euler_kunpeng_x64.zip	dws_client_8.1.x_euler_kunpeng_x64.zip.sha256
Redhat Kunpeng_64	CentOS-7.6-aarch64 and NeoKylin-7.6-aarch64 (adapted to Kunpeng 920 CPU)	dws_client_8.1.x_redhat_kunpeng_x64.zip	dws_client_8.1.x_redhat_kunpeng_x64.zip.sha256

3.3 Instruction

3.3.1 Using the Linux gsql Client to Connect to a Cluster

This section describes how to connect to a database through an SQL client after you create a data warehouse cluster and before you use the cluster's database. GaussDB(DWS) provides the Linux gsql client that matches the cluster version for you to access the cluster through the cluster's public or private network address.

The gsql command line client provided by DWS runs on Linux. Before using it to remotely connect to a GaussDB(DWS) cluster, you need to prepare a Linux server for installing and running the gsql client. If you use a public network address to access the cluster, you can install the Linux gsql client on your own Linux server. Ensure that the Linux server has a public network address. If no EIPs are configured for your GaussDB(DWS) cluster, you are advised to create a Linux ECS for convenience purposes. For more information, see **(Optional) Preparing an ECS as the gsql Client Server**.

(Optional) Preparing an ECS as the gsql Client Server

For details about how to purchase an ECS, see **Purchasing an ECS** in the *Elastic Cloud Server Getting Started*.

The created ECS must meet the following requirements:

- The ECS and data warehouse cluster must belong to the same region and AZ.
- If you use the gsql client provided by GaussDB(DWS) to connect to the GaussDB(DWS) cluster, the ECS image must meet the following requirements:

  The image's OS must be one of the following Linux OSs supported by the gsql client:

  - The **Redhat x86_64** client can be used on the following OSs:
    - RHEL 6.4 to RHEL 7.6
    - CentOS 6.4 to CentOS 7.4

    - EulerOS 2.3
  - The **SUSE x86_64** client can be used on the following OSs:
    - SLES 11.1 to SLES 11.4
    - SLES 12.0 to SLES 12.3
  - The **Euler Kunpeng_64** client can be used on the following OS:
    - EulerOS 2.8
  - The **Stream Euler x86_64** client can be used on the following OS:

    EulerOS 2.2
  - The **Stream Euler Kunpeng_64** client can be used on the following OS:
    - EulerOS 2.8

- If the client accesses the cluster using the private network address, ensure that the created ECS is in the same VPC as the GaussDB(DWS) cluster.

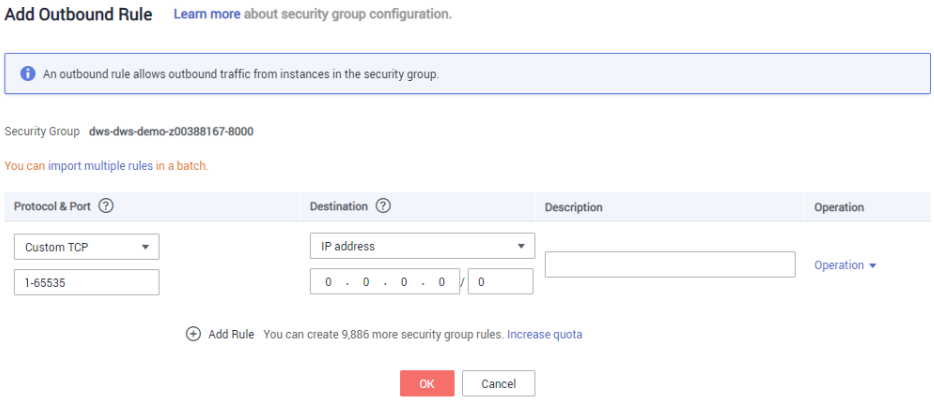
  For details about VPC operations, see **VPC and Subnet** in the *Virtual Private Cloud User Guide*.
- If the client accesses the cluster using the public network address, ensure that both the created ECS and GaussDB(DWS) cluster have an EIP.

  When purchasing an ECS, set **EIP** to **Buy now** or **Specify**.
- The security group rules of the ECS must enable communication between the ECS and the port that the GaussDB(DWS) cluster uses to provide services.

  For details about security group operations, see **Security Group** in the *Virtual Private Cloud User Guide*.

  Ensure that the security group of the ECS contains rules meeting the following requirements. If the rules do not exist, add them to the security group:
  - **Transfer Direction**: **Outbound**
  - **Protocol/Application**: The value must contain **TCP**, for example, **TCP** and **All**.
  - **Port**: The value must contain the database port that provides services in the GaussDB(DWS) cluster. For example, set this parameter to **1-65535** or a specific GaussDB(DWS) database port.
  - Destination: The IP address set here must contain the IP address of the GaussDB(DWS) cluster to be connected. **0.0.0.0/0** indicates any IP address.

Figure 3-1 Outbound rule

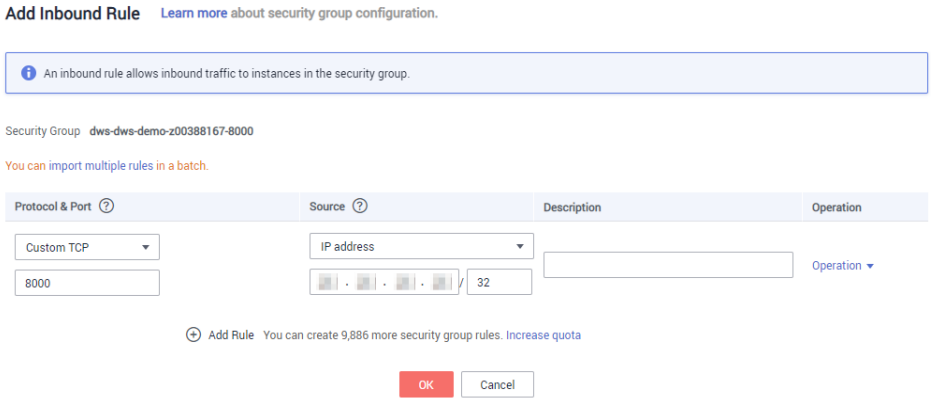


- The security group rules of the data warehouse cluster must ensure that GaussDB(DWS) can receive network access requests from clients.

  Ensure that the cluster's security group contains rules meeting the following requirements. If the rules do not exist, add them to the security group:

  - **Transfer Direction**: **Inbound**
  - **Protocol/Application**: The value must contain **TCP**, for example, **TCP** and **All**.
  - **Port**: Set this parameter to the database port that provides services in the data warehouse cluster, for example, **8000**.
  - **Source**: The IP address set here must contain the IP address of the GaussDB(DWS) client server, for example, **192.168.0.10/32**.

Figure 3-2 Inbound rule



Downloading the Linux gsql Client and Connecting to a Cluster

Step 1 Download the Linux gsql client by referring to **Downloading the Client**, and use an SSH file transfer tool (such as WinSCP) to upload the client to a target Linux server.

You are advised to download the gsql tool that matches the cluster version. That is, use gsql 8.1.x for clusters of 8.1.0 or later, and use gsql 8.2.x for clusters of 8.2.0 or later. To download gsql 8.2.x, replace **dws_client_8.1.x_redhat_x64.zip** with **dws_client_8.2.x_redhat_x64.zip**. The **dws_client_8.1.x_redhat_x64.zip** is used as an example.

The user who uploads the client must have the full control permission on the target directory on the host to which the client is uploaded.

Alternatively, you can remotely log in to the Linux server where the gsql is to be installed in SSH mode and run the following command in the Linux command window to download the Linux gsql client:

```
wget https://obs.ap-southeast-1.myhuaweicloud.com/dws/download/dws_client_8.1.x_redhat_x64.zip --no-check-certificate
```

Step 2 Use the SSH tool to remotely log in to the host where the client is installed.

For details about how to log in to an ECS, see **Login Using an SSH Password** in the *Elastic Cloud Server User Guide*.

Step 3 (Optional) To connect to the cluster in SSL mode, configure SSL authentication parameters on the host where the client is installed. For details, see **Establishing Secure TCP/IP Connections in SSL Mode**.

NOTE

The SSL connection mode is more secure than the non-SSL mode. You are advised to connect the client to the cluster in SSL mode.

Step 4 Run the following commands to decompress the client:

```
cd <Path for saving the client>
unzip dws_client_8.1.x_redhat_x64.zip
```

In the preceding commands:

- *<Path_for_storing_the_client>*: Replace it with the actual path.
- *dws_client_8.1.x_redhat_x64.zip*: This is the client tool package name of **RedHat x86**. Replace it with the actual name.

Step 5 Run the following command to configure the GaussDB(DWS) client:

```
source gsql_env.sh
```

If the following information is displayed, the GaussDB(DWS) client is successfully configured:

```
All things done.
```

Step 6 Connect to the database in the GaussDB(DWS) cluster using the gsql client. Replace the values of each parameter with actual values.

```
gsql -d <Database_name> -h <Cluster_address> -U <Database_user> -p <Database_port> -r
```

The parameters are described as follows:

- *Database_name*: Enter the name of the database to be connected. If you use the client to connect to the cluster for the first time, enter the default database **gaussdb**.
- **Cluster Address**: For details about how to obtain the cluster address, see **Obtaining the Cluster Connection Address**. If a public network address is used for connection, set this parameter to **Public Network Address** or **Public Network Domain Name**. If a private network address is used for connection, set this parameter to **Private Network Address** or **Private Network Domain Name**. If ELB is used for connection, set this parameter to **ELB Address**. If ELB is used for connection, set this parameter to the ELB address.
- *Database_user*: Enter the username of the cluster's database. If you use the client to connect to the cluster for the first time, set this parameter to the

default administrator configured during cluster creation, for example, **dbadmin**.

- *Database_port*: Enter the database port set during cluster creation.

For example, run the following command to connect to the default database **gaussdb** in the GaussDB(DWS) cluster:

```
gsql -d gaussdb -h 10.168.0.74 -U dbadmin -p 8000 -W password -r
```

If the following information is displayed, the connection succeeded:

```
gaussdb=>
```

----End

gsql Command Reference

For details about gsql commands, see **Meta-Command Reference**.

(Optional) Importing TPC-DS Sample Data Using gsql

GaussDB(DWS) users can import data from external sources to data warehouse clusters. This section describes how to import sample data from OBS to a data warehouse cluster and perform querying and analysis operations on the sample data. The sample data is generated based on the standard TPC-DS benchmark test.

TPC-DS is the benchmark for testing the performance of decision support. With TPC-DS test data and cases, you can simulate complex scenarios, such as big data set statistics, report generation, online query, and data mining, to better understand functions and performance of database applications.

NOTE

Currently, TPC-DS sample data can be imported only in the CN North-Beijing1 region.

Step 1 Use the SSH remote connection tool to log in to the server where the gsql client is installed and go to the gsql directory. The **/opt** directory is used as an example for storing the gsql client.

```
cd /opt
```

Step 2 Switch to the specified directory and set the AK and SK for importing sample data and the OBS access address.

```
cd sample
/bin/bash setup.sh -ak <Access_Key_Id> -sk <Secret_Access_Key> -obs_location obs.ap-southeast-1.myhuaweicloud.com
```

If the following information is displayed, the settings are successful:

```
setup successfully!
```

NOTE

<Access_Key_Id> and *<Secret_Access_Key>*: indicate the AK and SK, respectively. For details about how to obtain the AK and SK, see **Creating Access Keys (AK and SK)**. Then, replace the parameters in the statements with the obtained values.

Step 3 Go back to previous directory and run the gsql environment variables.


```
cd ..
source gsql_env.sh
cd bin
```

Step 4 Import the sample data to the data warehouse.

Command format:

```
gsql -d <Database name> -h <Public network address of the cluster> -U <Administrator> -p <Data warehouse port number> -f <Path for storing the sample data script> -r
```

Sample command:

```
gsql -d gaussdb -h 10.168.0.74 -U dbadmin -p 8000 -f /opt/sample/tpcds_load_data_from_obs.sql -r
```

NOTE

In the preceding command, sample data script **tpcds_load_data_from_obs.sql** is stored in the sample directory (for example, **/opt/sample/**) of the GaussDB(DWS) client.

After you enter the administrator password and successfully connect to the database in the cluster, the system will automatically create a foreign table to associate the sample data outside the cluster. Then, the system creates a target table for saving the sample data and imports the data to the target table using the foreign table.

The time required for importing a large dataset depends on the current GaussDB(DWS) cluster specifications. Generally, the import takes about 10 to 20 minutes. If information similar to the following is displayed, the import is successful.

```
Time:1845600.524 ms
```

Step 5 In the Linux command window, run the following commands to switch to a specific directory and query the sample data:

```
cd /opt/sample/query_sql/
/bin/bash tpcds100x.sh
```

Step 6 Enter the cluster's public network IP address, access port, database name, user who accesses the database, and password of the user as prompted.

- The default database name is **gaussdb**.
- Use the administrator username and password configured during cluster creation as the username and password for accessing the database.

After the query is complete, a directory for storing the query result, such as **query_output_20170914_072341**, will be generated in the current query directory, for example, **sample/query_sql/**.

----End

3.3.2 Using the Windows gsql Client to Connect to a Cluster

This section describes how to connect to a database through an SQL client after you create a data warehouse cluster and before you use the cluster's database. GaussDB(DWS) provides the Windows gsql client that matches the cluster version for you to access the cluster through the cluster's public or private network address.

Procedure

Step 1 Prepare a Windows ECS to install and run the gsql client. Windows Server 2008/ Windows 7 and later are supported.

Step 2 Download the Windows gsql client by referring to **Downloading the Client** and decompress the package to a local folder.

Figure 3-3 Windows gsql client folder

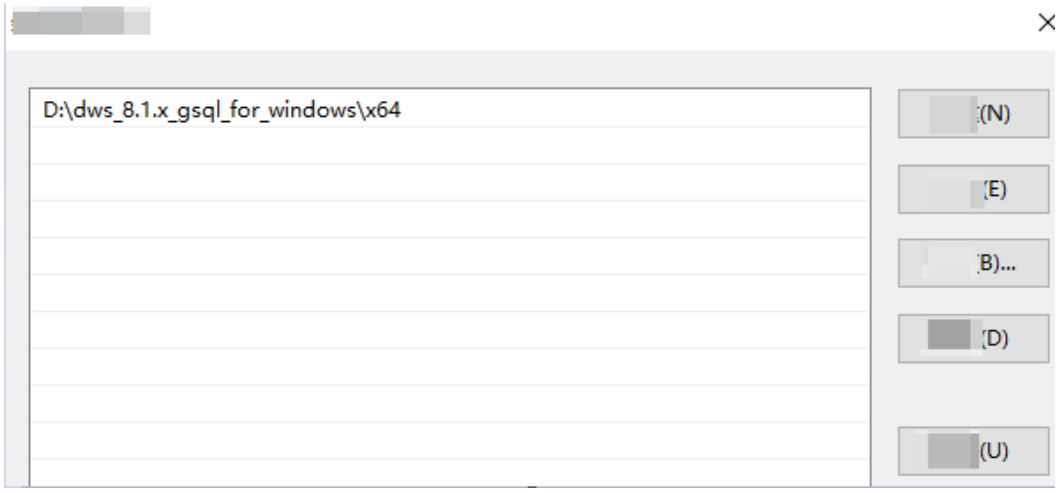
Step 3 Set environment variables. For a 32-bit OS, select the **x86** folder. For a 64-bit OS, select the **x64** folder.

Method 1: Configure environment variables in the CLI. Open the command prompt and run the **set path=***<window_gsql>***;%path%** command, where *<window_gsql>* indicates the folder path where the Windows gsql client was decompressed to in the previous step. For example:

```
set path=C:\Users\xx\Desktop\dws_8.1.x_gsql_for_windows\x64;%path%
```

Method 2: In the **Control Panel** window, search for **System** and click **View advanced system settings**. Click the **Advanced** tab, and click **Environment Variables**. Select the **Path** parameter and click **Edit**. Add the gsql path in the parameter value. For example:

Figure 3-4 Configuring Windows environment variables



Step 4 (Optional) To connect to the cluster in SSL mode, configure SSL authentication parameters on the server where the client is installed. For details, see **Establishing Secure TCP/IP Connections in SSL Mode**.

> **NOTE**
>
> The SSL connection mode is more secure than the non-SSL mode. You are advised to connect the client to the cluster in SSL mode.

Step 5 Run the following command to connect to the database in the GaussDB(DWS) cluster using the gsql client:

```
gsql -d <Database_name> -h <Cluster_address> -U <Database_user> -p <Database_port> -r
```

The parameters are as follows:

- **Database name**: Enter the name of the database to be connected. If you use the client to connect to the cluster for the first time, enter the default database **gaussdb**.
- **Cluster Address**: For details about how to obtain the cluster address, see **Obtaining the Cluster Connection Address**. If a public network address is used for connection, set this parameter to the public network domain name. If a private network address is used for connection, set this parameter to the private network domain name. If ELB is used for connection, set this parameter to **ELB Address**. If ELB is used for connection, set this parameter to the ELB address.
- **Database user**: Enter the username of the cluster's database. If you use the client to connect to the cluster for the first time, set this parameter to the default administrator configured during cluster creation, for example, **dbadmin**.
- **Database port**: Enter the database port set during cluster creation.

For example, run the following command to connect to the default database **gaussdb** in the GaussDB(DWS) cluster:

```
gsql -d gaussdb -h 10.168.0.74 -U dbadmin -p 8000 -W password -r
```

If the following information is displayed, the connection succeeded:

```
gaussdb=>
```

----End

Precautions

1. The default character encoding of the Windows command prompt is GBK, and the default value of **client_encoding** of Windows gsql is **GBK**. Some characters encoded using UTF-8 cannot be displayed in Windows gsql.

   Suggestion: Ensure the file specified using **-f** uses UTF-8 encoding, and set the default encoding format to **UTF-8** (**set client_encoding='utf-8';**).

2. Paths in Windows gsql must be separated by slashes (/), or an error will be reported. In a meta-command, the backslash (\) indicates the start of a meta-command. If the backslash is enclosed in single quotation marks ('\'), it is used for escape.

```
gaussdb=> \i D:\test.sql
D:: Permission denied
postgres=> \i D:/test.sql
id
----
 1
(1 row)
```


3. To use the **\!** metacommand to run a system command in Windows gsql, be sure to use the path separator required by the system command. Generally, the path separator is a backslash (\).
```
gaussdb=> \! type D:/test.sql
Incorrect syntax.
gaussdb=> \! type D:\test.sql
select 1 as id;
```
4. Windows gsql does not support the **\parallel** meta-command.
```
gaussdb=> \parallel
ERROR: "\parallel" is not supported in Windows.
```
5. In Linux shell, single quotation marks ('') and double quotation marks ("") can be used to enclose strings. In Windows, only double quotation marks can be used.
```
gsql -h 192.168.233.189 -p 8109 -d postgres -U odbcuser -W odbc_234 -c "select 1 as id"
 id
----
  1
(1 row)
```
If single quotation marks are used, an error will be reported and the input will be ignored.
```
gsql -h 192.168.233.189 -p 8109 -d postgres -U odbcuser -W odbc_234 -c 'select 1 as id'
gsql: warning: extra command-line argument "1" ignored
gsql: warning: extra command-line argument "as" ignored
gsql: warning: extra command-line argument "id'" ignored
ERROR:  unterminated quoted string at or near "'select"
LINE 1: 'select
```
6. If Windows gsql is idle for a long time after a connection is established, the connection session times out, and an SSL error is reported. In this case, you need to log in again. The following error is reported:
```
SSL SYSCALL error: Software caused connection abort (0x00002745/10053), remote datanode
<NULL>, error: Result too large
```
7. In Windows, press **Ctrl**+**C** to exit gsql. If **Ctrl**+**C** are pressed during input, the input will be ignored and you will be forced to exit gsql.

   Enter **as** and press **Ctrl**+**C**. After **\q** is displayed, exit gsql.
```
gaussdb=> select 1
gaussdb=> as \q
```
8. Windows gsql cannot connect to a database using the LATIN1 character encoding. The error information is as follows:
```
gsql: FATAL: conversion between GBK and LATIN1 is not supported
```
9. The location of the **gsqlrc.conf** file:

   The default **gsqlrc** path is **%APPDATA%/postgresql/gsqlrc.conf**. You can also set the path using the **PSQLRC** variable.
```
set PSQLRC=C:\Users\xx\Desktop\dws_8.1.x_gsql_for_windows\x64\gsqlrc.conf
```

gsql Command Reference

For details about gsql commands, see **Meta-Command Reference**.

3.3.3 Establishing Secure TCP/IP Connections in SSL Mode

GaussDB(DWS) supports the standard SSL. As a highly secure protocol, SSL authenticates bidirectional identification between the server and client using digital signatures and digital certificates to ensure secure data transmission. To support SSL connection, GaussDB(DWS) has obtained the formal certificates and keys for the server and client from the CA certification center. It is assumed that the key and certificate for the server are **server.key** and **server.crt** respectively; the

key and certificate for the client are **client.key** and **client.crt** respectively, and the name of the CA root certificate is **cacert.pem**.

The SSL connection mode is more secure. By default, the SSL feature in a cluster allows SSL and non-SSL connections from the client. For security purposes, you are advised to connect to the cluster via SSL from the client. Ensure the certificate, private key, and root certificate of the GaussDB(DWS) server have been configured by default. To forcibly use an SSL connection, configure the **require_ssl** parameter in the **Require SSL Connection** area of the cluster's **Security Settings** page on the GaussDB(DWS) management console. Require SSL Connection on the Security Settings page of the cluster. For more information, see **Configuring SSL Connection** and **Combinations of SSL Connection Parameters on the Client and Server**.

The client or JDBC/ODBC driver needs to use SSL connection. Configure related SSL connection parameters in the client or application code. The GaussDB(DWS) management console provides the SSL certificate required by the client. The SSL certificate contains the default certificate, private key, root certificate, and private key password encryption file required by the client. Download the SSL certificate to the host where the client is installed, and specify the path of the certificate on the client. For more information, see **Configuring Digital Certificate Parameters Related to SSL Authentication on the gsql Client** and **SSL Authentication Modes and Client Parameters**.

NOTE

Using the default certificate may pose security risks. To improve system security, you are advised to periodically change the certificate to prevent password cracking. If you need to replace the certificate, contact the database customer service.

Configuring SSL Connection

Prerequisites

- Changes made to security configuration parameters require a cluster restart to take effect. Otherwise, the cluster will be temporarily unavailable.
- To modify the cluster's security configuration, ensure that the following conditions are met:
  - The cluster status is **Available** or **Unbalanced**.
  - The **Task Information** cannot be set to **Creating snapshot**, **Scaling out**, **Configuring**, or **Restarting**.

Procedure

Step 1 Log in to the GaussDB(DWS) console.

Step 2 Choose **Dedicated Clusters** > **Clusters** in the navigation pane.

Step 3 In the cluster list, click the name of a cluster. On the page that is displayed, click **Security Settings**.

By default, **Configuration Status** is **Synchronized**, which indicates that the latest database result is displayed.

Step 4 In the **SSL Connection** area, enable **Require SSL Connection** (recommended).

 indicates the function is enabled. The **require_ssl** is set to **1**, indicating that the server forcibly requires the SSL connection.

 indicates the function is disabled (default value). The **require_ssl** parameter is set to **0**, indicating that the server does not require SSL connections. For details about how to configure the **require_ssl** parameter, see **require_ssl (Server)**.

NOTE

- If the gsql client or ODBC driver provided by GaussDB(DWS) is used, GaussDB(DWS) supports the TLSv1.2 SSL protocol.
- If the JDBC driver provided by GaussDB(DWS) is used, GaussDB(DWS) supports SSL protocols, such as SSLv3, TLSv1, TLSv1.1, and TLSv1.2. The SSL protocol used between the client and the database depends on the Java Development Kit (JDK) version used by the client. Generally, JDK supports multiple SSL protocols.

Step 5 Click **Apply**.

The system automatically saves the SSL connection settings. On the **Security Settings** page, **Configuration Status** is **Applying**. After **Configuration Status** changes to **Synchronized**, the settings have been saved and taken effect.

----End

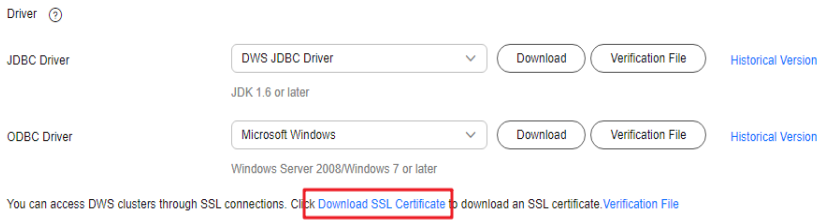
Configuring Digital Certificate Parameters Related to SSL Authentication on the gsql Client

After a data warehouse cluster is deployed, the SSL authentication mode is enabled by default. The server certificate, private key, and root certificate have been configured by default. You need to configure the client parameters.

Step 1 Log in to the GaussDB(DWS) console. In the navigation pane, choose **Client Connections**.

Step 2 In the **Driver** area, click **download an SSL certificate**.

Figure 3-5 Downloading an SSL certificate



Step 3 Use a file transfer tool (such as WinSCP) to upload the SSL certificate to the host where the client is installed.

For example, save the downloaded certificate **dws_ssl_cert.zip** to the **/home/dbadmin/dws_ssl/** directory.

Step 4 Use an SSH remote connection tool (such as PuTTY) to log in to the host where the gsql client is installed and run the following commands to go to the directory where the SSL certificate is stored and decompress the SSL certificate:


```
cd /home/dbadmin/dws_ssl/
unzip dws_ssl_cert.zip
```

Step 5 Run the export command and configure digital certificate parameters related to SSL authentication on the host where the gsql client is installed.

There are two SSL authentication modes: bidirectional authentication and unidirectional authentication. The client environment variables to be configured vary according to the authentication mode. For details, see **SSL Authentication Modes and Client Parameters**.

The following parameters must be configured for bidirectional authentication:

```
export PGSSLCERT="/home/dbadmin/dws_ssl/sslcert/client.crt"
export PGSSLKEY="/home/dbadmin/dws_ssl/sslcert/client.key"
export PGSSLMODE="verify-ca"
export PGSSLROOTCERT="/home/dbadmin/dws_ssl/sslcert/cacert.pem"
```

The following parameters must be configured for unidirectional authentication:

```
export PGSSLMODE="verify-ca"
export PGSSLROOTCERT="/home/dbadmin/dws_ssl/sslcert/cacert.pem"
```

NOTICE

- You are advised to use bidirectional authentication for security purposes.
- The environment variables configured for a client must contain the absolute file paths.

Step 6 Change the client private key permissions.

The permissions on the client's root certificate, private key, certificate, and encrypted private key file must be **600**. If the permissions do not meet the requirement, the client cannot connect to the cluster in SSL mode.

```
chmod 600 client.key
chmod 600 client.crt
chmod 600 client.key.cipher
chmod 600 client.key.rand
chmod 600 cacert.pem
```

----End

SSL Authentication Modes and Client Parameters

There are two SSL authentication modes: bidirectional authentication and unidirectional authentication. Table **Table 3-7** shows the differences between these two modes. You are advised to use bidirectional authentication for security purposes.

Table 3-7 Authentication modes

Authentication Mode	Description	Environment Variables Configured on a Client	Maintenance
Bidirectional authentication (recommended)	The client verifies the server's certificate and the server verifies the client's certificate. The connection can be set up only after the verifications are successful.	Set the following environment variables: • PGSSLCERT • PGSSLKEY • PGSSLROOTCERT • PGSSLMODE	This authentication mode is applicable to scenarios that require high data security. When using this mode, you are advised to set the **PGSSLMODE** client variable to **verify-ca** for network data security purposes.
Unidirectional authentication	The client verifies the server's certificate, whereas the server does not verify the client's certificate. The server loads the certificate information and sends it to the client. The client verifies the server's certificate according to the root certificate.	Set the following environment variables: • PGSSLROOTCERT • PGSSLMODE	To prevent TCP-based security attacks, you are advised to use the SSL certificate authentication. In addition to configuring the client root certificate, you are advised to set the **PGSSLMODE** variable to **verify-ca** on the client.

Configure environment variables related to SSL authentication on the client. For details, see **Table 3-8**.

NOTE

The path of environment variables is set to */home/dbadmin***/dws_ssl/** as an example. Replace it with the actual path.

Table 3-8 Client parameters

Environment Variable	Description	Value Description
PGSSLCERT	Specifies the certificate files for a client, including the public key. Certificates prove the legal identity of the client and the public key is sent to the remote end for data encryption.	The absolute path of the files must be specified, for example: export PGSSLCERT='*/home/dbadmin/dws_ssl/sslcert/client.crt*' (No default value)
PGSSLKEY	Specifies the client private key file used to decrypt the digital signatures and the data encrypted using the public key.	The absolute path of the files must be specified, for example: export PGSSLKEY='*/home/dbadmin/dws_ssl/sslcert/client.key*' (No default value)
PGSSLMODE	Specifies whether to negotiate with the server about SSL connection and specifies the priority of the SSL connection.	Values and meanings: • **disable**: only tries to establish a non-SSL connection. • **allow**: tries to establish a non-SSL connection first, and then an SSL connection if the first attempt fails. • **prefer**: tries to establish an SSL connection first, and then a non-SSL connection if the first attempt fails. • **require**: only tries to establish an SSL connection. If there is a CA file, perform the verification according to the scenario in which the parameter is set to **verify-ca**. • **verify-ca**: tries to establish an SSL connection and check whether the server certificate is issued by a trusted CA. • **verify-full**: GaussDB(DWS) does not support this mode. Default value: **prefer** **NOTE** When an external client accesses a cluster, the error message "ssl SYSCALL error" is displayed on some nodes. In this case, run **export PGSSLMODE="allow"** or **export PGSSLMODE="prefer"**.

Environment Variable	Description	Value Description
PGSSLROOTCERT	Specifies the root certificate file for issuing client certificates. The root certificate is used to verify the server certificate.	The absolute path of the files must be specified, for example: export PGSSLROOTCERT='*/home/dbadmin/dws_ssl/sslcert/certca.pem*' Default value: null
PGSSLCRL	Specifies the certificate revocation list file, which is used to check whether a server certificate is in the list. If the certificate is in the list, it is invalid.	The absolute path of the files must be specified, for example: export PGSSLCRL='*/home/dbadmin/dws_ssl/sslcert/sslcrl-file.crl*' Default value: null

Combinations of SSL Connection Parameters on the Client and Server

Whether the client uses the SSL encryption connection mode and whether to verify the server certificate depend on client parameter **sslmode** and server (cluster) parameters **ssl** and **require_ssl**. The parameters are as follows:

- **ssl (Server)**

  The **ssl** parameter indicates whether to enable the SSL function. **on** indicates that the function is enabled, and **off** indicates that the function is disabled.

  - The default value is **on** and you cannot set this parameter on the GaussDB(DWS) console.

- **require_ssl (Server)**

  The **require_ssl** parameter specifies whether the server forcibly requires SSL connection. This parameter is valid only when **ssl** is set to **on**. **on** indicates that the server forcibly requires SSL connection. **off** indicates that the server does not require SSL connection.

  - The default value is **off**. You can set the **require_ssl** parameter in the **Require SSL Connection** area of the cluster's **Security Settings** page on the GaussDB(DWS) console.

- **sslmode (Client)**

  You can set this parameter in the SQL client tool.

  - In the gsql command line client, this parameter is the **PGSSLMODE** parameter.
  - On the Data Studio client, this parameter is the **SSL Mode** parameter.

The combinations of client parameter **sslmode** and server parameters **ssl** and **require_ssl** are as follows.

Table 3-9 Combinations of SSL connection parameters on the client and server

ssl (Server)	sslmode (Client)	require_ssl (Server)	Result
on	disable	on	The server requires SSL, but the client disables SSL for the connection. As a result, the connection cannot be set up.
	disable	off	The connection is not encrypted.
	allow	on	The connection is encrypted.
	allow	off	The connection is not encrypted.
	prefer	on	The connection is encrypted.
	prefer	off	The connection is encrypted.
	require	on	The connection is encrypted.
	require	off	The connection is encrypted.
	verify-ca	on	The connection is encrypted and the server certificate is verified.
	verify-ca	off	The connection is encrypted and the server certificate is verified.
off	disable	on	The connection is not encrypted.
	disable	off	The connection is not encrypted.
	allow	on	The connection is not encrypted.
	allow	off	The connection is not encrypted.
	prefer	on	The connection is not encrypted.
	prefer	off	The connection is not encrypted.
	require	on	The client requires SSL, but SSL is disabled on the server. Therefore, the connection cannot be set up.
	require	off	The client requires SSL, but SSL is disabled on the server. Therefore, the connection cannot be set up.
	verify-ca	on	The client requires SSL, but SSL is disabled on the server. Therefore, the connection cannot be set up.
	verify-ca	off	The client requires SSL, but SSL is disabled on the server. Therefore, the connection cannot be set up.

3.4 Online Help

Procedure

- When a database is being connected, run the following commands to obtain the help information:
  ```
  gsql --help
  ```
  The following information is displayed:
  ```
  ......
  Usage:
    gsql [OPTION]... [DBNAME [USERNAME]]

  General options:
    -c, --command=COMMAND    run only single command (SQL or internal) and exit
    -d, --dbname=DBNAME      database name to connect to (default: "postgres")
    -f, --file=FILENAME      execute commands from file, then exit
  ......
  ```
- After the database is connected, run the following commands to obtain the help information:
  ```
  help
  ```
  The following information is displayed:
  ```
  You are using gsql, the command-line interface to gaussdb.
  Type:  \copyright for distribution terms
         \h for help with SQL commands
         \? for help with gsql commands
         \g or terminate with semicolon to execute query
         \q to quit
  ```

Task Example

Step 1 View the **gsql** help information. For details about the commands, see **Table 3-10**.

Table 3-10 gsql online help

Description	Example
View copyright information.	\copyright

Description	Example
View the help information about SQL statements supported by GaussDB(DWS).	View the help information about SQL statements supported by GaussDB(DWS). For example, view all SQL statements supported by GaussDB(DWS). \h Available help: ABORT ALTER DATABAE ALTER DATA SOURCE ... ... For example, view parameters of the **CREATE DATABASE** command: \help CREATE DATABASE Command: CREATE DATABASE Description: create a new database Syntax: CREATE DATABASE database_name [[WITH] {[OWNER [=] user_name]| [TEMPLATE [=] template]| [ENCODING [=] encoding]| [LC_COLLATE [=] lc_collate]| [LC_CTYPE [=] lc_ctype]| [DBCOMPATIBILITY [=] compatibility_type]| [TABLESPACE [=] tablespace_name]| [CONNECTION LIMIT [=] connlimit]}[...]];
View help information about gsql commands.	For example, view commands supported by **gsql**. \? General \copyright show PostgreSQL usage and distribution terms \g [FILE] or ; execute query (and send results to file or |pipe) \h(\help) [NAME] help on syntax of SQL commands, * for all commands \q quit gsql ... ...

----End

3.5 Command Reference

For details about gsql parameters, see **Table 3-11**, **Table 3-12**, **Table 3-13**, and **Table 3-14**.

Table 3-11 Common parameters

Parameter	Description	Value Range
-c, --command=COMMAND	Specifies that gsql runs a string command and then exits.	-

Parameter	Description	Value Range
-C, --set-file=FILENAME	Uses the file as the command source instead of interactive input. After processing the file, gsql does not exit and continues to process other contents.	An absolute path or relative path that meets the OS path naming convention
-d, --dbname=DBNAME	Specifies the name of the database to be connected.	A character string.
-D, --dynamic-param	Controls the generation of variables and the **${}** variable referencing method during SQL statement execution. For details, see **Variable**.	-
-f, --file=FILENAME	Specifies that files are used as the command source instead of interactively-entered commands. After the files are processed, exit from gsql. If *FILENAME* is - (hyphen), then standard input is read.	An absolute path or relative path that meets the OS path naming convention
-l, --list	Lists all available databases and then exits.	-
-v, --set, --variable=NAME=VALUE	Sets the gsql variable *NAME* to *VALUE*. For details about variable examples and descriptions, see **Variable**.	-
-X, --no-gsqlrc	Does not read the startup file (neither the system-wide **gsqlrc** file nor the user's **~/.gsqlrc** file). **NOTE** The startup file is **~/.gsqlrc** by default or it can be specified by the environment variable *PSQLRC*.	-
-1 ("one"), --single-transaction	When gsql uses the **-f** parameter to execute a script, **START TRANSACTION** and **COMMIT** are added to the start and end of the script, respectively, so that the script is executed as one transaction. This ensures that the script is executed successfully. If the script cannot be executed, the script is invalid. **NOTE** If the script has used START TRANSACTION, COMMIT, and ROLLBACK, this parameter is invalid.	-
-?, --help	Displays help information about gsql CLI parameters, and exits.	-
-V, --version	Prints the gsql version and exits.	-

Table 3-12 Input and output parameters

Parameter	Description	Value Range
-a, --echo-all	Prints all input lines to standard output as they are read. **CAUTION** When this parameter is used in some SQL statements, sensitive information, such as user passwords, may be disclosed. Use this parameter with caution.	-
-e, --echo-queries	Copies all SQL statements sent to the server to standard output as well. **CAUTION** When this parameter is used in some SQL statements, sensitive information, such as user passwords, may be disclosed. Use this parameter with caution.	-
-E, --echo-hidden	Echoes the actual queries generated by **\d** and other backslash commands.	-
-k, --with-key=KEY	Uses gsql to decrypt imported encrypted files. **NOTICE** For key characters, such as the single quotation mark (') or double quotation mark (") in shell commands, Linux shell checks whether the input single quotation mark (') or double quotation mark (") matches. If it does not match, Linux shell regards that the user input is unfinished and waits for more input instead of entering the gsql program.	-
-L, --log-file=FILENAME	Writes normal output destination and all query output into the **FILENAME** file. **CAUTION** • When this parameter is used in some SQL statements, sensitive information, such as user passwords, may be disclosed. Use this parameter with caution. • This parameter retains only the query result in the corresponding file, so that the result can be easily found and parsed by other invokers (for example, automatic O&M scripts). Logs about gsql operation are not retained.	An absolute path or relative path that meets the OS path naming convention
-m, --maintenance	Allows a cluster to be connected when a two-phase transaction is being restored. **NOTE** The parameter is for engineers only. When this parameter is used, gsql can be connected to the standby server to check data consistency between the primary server and standby server.	-

Parameter	Description	Value Range
-n, --no-libedit	Closes the command line editing.	-
-o, --output=FILENAME	Puts all query output into the **FILENAME** file.	An absolute path or relative path that meets the OS path naming convention
-q, --quiet	Indicates the quiet mode and no additional information will be printed.	By default, gsql displays various information.
-s, --single-step	Runs in single-step mode. This indicates that the user is prompted before each command is sent to the server. This parameter can also be used for canceling execution. This parameter can be used to debug scripts. **CAUTION** When this parameter is used in some SQL statements, sensitive information, such as user passwords, may be disclosed. Use this parameter with caution.	-
-S, --single-line	Runs in single-row mode where a new line terminates an SQL statement in the same manner as a semicolon does.	-

Table 3-13 Parameters specifying output formats

Parameter	Description	Value Range
-A, --no-align	Switches to unaligned output mode.	The default output mode is aligned.
-F, --field-separator=STRING	Specifies the field separator. The default is the vertical bar (|).	-
-H, --html	Turns on the HTML tabular output.	-
-P, --pset=VAR[=ARG]	Specifies the print option in the \pset format in the command line. **NOTE** The equal sign (=), instead of the space, is used here to separate the name and value. For example, enter **-P format=latex** to set the output format to **LaTeX**.	-

Parameter	Description	Value Range
-R, --record-separator=STRING	Specifies the record separators.	-
-r	Enables the function of recording historical operations on the client.	This function is disabled by default.
-t, --tuples-only	Prints only tuples.	-
-T, --table-attr=TEXT	Specifies options to be placed within the HTML table tag. Use this parameter with the **-H,--html** parameter to specify the output to the HTML format.	-
-x, --expanded	Turns on the expanded table formatting mode.	-
-z, --field-separator-zero	Sets the field separator in the unaligned output mode to be blank. Use this parameter with the **-A, --no-align** parameter to switch to unaligned output mode.	-
-0, --record-separator-zero	Sets the record separator in the unaligned output mode to be blank. Use this parameter with the **-A, --no-align** parameter to switch to unaligned output mode.	-
-g	Displays separators for all SQL statements and specified files. **NOTE** The **-g** parameter must be configured with the **-f** parameter.	-

Table 3-14 Connection parameters

Parameter	Description	Value Range
-h, --host=HOSTNAME	Specifies the host name of the machine on which the server is running or the directory for the Unix-domain socket.	If the host name is omitted, gsql connects to the server of the local host over the Unix domain socket or over TCP/IP to connect to local host without the Unix domain socket.
-p, --port=PORT	Specifies the port number of the database server. You can modify the default port number using the **-p, --port=PORT** parameter.	The default value is **8000**.
-U, --username=USERNAME	Specifies the user that accesses a database. **NOTE** • If a user is specified to access a database using this parameter, a user password must be provided together for identity verification. You can enter the password interactively or use the **-W** parameter to specify a password. • To connect to a database, add an escape character before any dollar sign ($) in the user name.	A string. The default user is the current user that operates the system.
-W, --password=PASSWORD	Specifies a password when the **-U** parameter is used to connect to a remote database. **NOTE** To connect to a database, add an escape character before any backslash (\) or back quote (`) in the password. If this parameter is not specified but database connection requires your password, you will be prompted to enter your password in interactive mode. The maximum length of the password is 999 bytes, which is restricted by the maximum value of the GUC parameter **password max length**.	This parameter must meet the password complexity requirement.

3.6 Meta-Command Reference

This section describes meta-commands provided by gsql after the GaussDB(DWS) database CLI tool is used to connect to a database. A gsql meta-command can be anything that you enter in gsql and begins with an unquoted backslash.

Precautions

- The format of the gsql meta-command is a backslash (\) followed by a command verb, and then a parameter. The parameters are separated from the command verb and from each other by any number of whitespace characters.
- To include whitespace in a parameter, you can quote it with single quotation marks (''). To include single quotation marks in a parameter, add a backslash in front of it. Anything contained in single quotation marks is furthermore subject to C-like substitutions for \n (new line), \t (tab), \b (backspace), \r (carriage return), \f (form feed), \digits (octal), and \xdigits (hexadecimal).
- Within a parameter, text enclosed in double quotation marks ("") is taken as a command line input to the shell. The command output (with any trailing newline removed) is taken as a parameter.
- If an unquoted argument begins with a colon (:), the parameter is taken as a gsql variable and the value of the variable is used as the parameter value instead.
- Some commands take an SQL identifier (such as a table name) as a parameter. These parameters follow the SQL syntax rules: Unquoted letters are forced to lowercase, while double quotation marks ("") protect letters from case conversion and allow incorporation of whitespace into the identifier. Within double quotation marks, paired double quotation marks reduce to a single double quotation mark in the result name. For example, **FOO"BAR"BAZ** is interpreted as **fooBARbaz**, and **"Aweird""name"** becomes **A weird"name**.
- Parameter parsing stops when another unquoted backslash appears. An unquoted backslash is taken as the beginning of a new meta-command. The special sequence \\ (two backslashes) marks the end of parameters and continues parsing SQL statements if any. In this way, SQL statements and gsql commands can be freely mixed in a row. However, the parameters of a meta-command cannot continue beyond the end of a line in any situations.

Meta-command

For details about meta-commands, see **Table 3-15**, **Table 3-16**, **Table 3-17**, **Table 3-18**, **Table 3-20**, **Table 3-22**, **Table 3-23**, **Table 3-24**, and **Table 3-26**.

NOTICE

FILE mentioned in the following commands indicates a file path. This path can be an absolute path such as **/home/gauss/file.txt** or a relative path, such as **file.txt**. By default, a **file.txt** is created in the path where the user runs gsql commands.

Table 3-15 Common meta-commands

Parameter	Description	Value Range
\copyright	Displays GaussDB(DWS) version and copyright information.	-
\g [FILE] or ;	Performs a query operation and sends the result to a file or pipe.	-
\h(\help) [NAME]	Provides syntax help on the specified SQL statement.	If the name is not specified, then gsql will list all the commands for which syntax help is available. If the name is an asterisk (*), syntax help on all SQL statements is displayed.

Parameter	Description	Value Range
\parallel [on [num]|off]	Controls the parallel execution function. • **on**: The switch is enabled and the maximum number of concurrently executed tasks is **num**. • **off**: This switch is disabled. **NOTE** • Parallel execution is not allowed in a running transaction and a transaction is not allowed to be stared during parallel execution. • Parallel execution of **\d** meta-commands is not allowed. • If **SELECT** statements are run concurrently, customers can accept the problem that the return results are displayed randomly but they cannot accept it if a core dump or process response failure occurs. • **SET** statements are not allowed in concurrent tasks because they may cause unexpected results. • The DISCARD command cannot be used in **\parallel**. • Temporary tables cannot be created. If temporary tables are required, create them before parallel execution is enabled, and use them only in the parallel execution. Temporary tables cannot be created in parallel execution. • When **\parallel** is executed, *num* independent gsql processes can be connected to the database server. • The duration of all jobs in **\parallel** cannot exceed the value of **session_timeout**. Otherwise, the connection may be interrupted during concurrent execution.	The default value of *num* is **1024**. **NOTICE** • The maximum number of connections allowed by the server is determined based on **max_connection** and the number of current connections. • Set the value of *num* based on the allowed number of connections.
\q [value]	Exits the gsql program. In a script file, this command is run only when a script terminates. The exit code is determined by the value.	-

Table 3-16 Buffer query meta-commands

Parameter	Description
\e [FILE] [LINE]	Use an external editor to edit the query buffer or file.

Parameter	Description
\ef [FUNCNAME [LINE]]	Use an external editor to edit the function definition. If **LINE** is specified, the cursor will point to the specified line of the function body.
\p	Prints the current query buffer to the standard output.
\r	Resets (clears) the query buffer.
\w FILE	Outputs the current query buffer to a file.

Table 3-17 Input and output meta-commands

Parameter	Description
\copy { table [(column_list)] | (query) } { from | to } { filename | stdin | stdout | pstdin | pstdout } [with] [binary] [oids] [delimiter [as] 'character'] [null [as] 'string'] [csv [header] [quote [as] 'character'] [escape [as] 'character'] [force quote column_list | *] [force not null column_list]]	After logging in to the database on any **gsql** client, you can import and export data. This is an operation of running the **SQL COPY** command, but not the server that reads or writes data to a specified file. Instead, data is transferred between the server and the local file system. This means that the accessibility and permissions of the file are the permissions of the local user rather than the server. The initial database user permission is not required. **NOTE** **\copy** only applies to small-batch data import with uniform formats but poor error tolerance capability. GDS or **COPY** is preferred for data import.
\echo [STRING]	Writes a character string to the standard output.
\i FILE	Reads content from *FILE* and uses them as the input for a query.
\i+ FILE KEY	Runs commands in an encrypted file.
\ir FILE	Is similar to **\i**, but resolves relative path names differently.
\ir+ FILE KEY	Is similar to **\i**, but resolves relative path names differently.
\o [FILE]	Saves all query results to a file.
\qecho [STRING]	Prints a character string to the query result output.

NOTE

In Table 3-18, **S** indicates that the system object is displayed, and **+** indicates that additional object descriptions are displayed. **PATTERN** specifies the name of an object to be displayed.

Table 3-18 Information display meta-commands

Parameter	Description	Value Range	Example
\d[S+]	Lists all tables, views, and sequences of all schemas in the search_path. When objects with the same name exist in different schemas in the search_path, only the object in the schema that ranks first in the search_path is displayed.	-	Lists all tables, views, and sequences of all schemas in the search_path. \d
\d[S+] NAME	Lists the structure of specified tables, views, and indexes.	-	Lists the structure of table **a**. \dtable+ a
\d+ [PATTERN]	Lists all tables, views, and indexes.	If **PATTERN** is specified, only tables, views, and indexes whose names match **PATTERN** are displayed.	Lists all tables, views, and indexes whose names start with **f**. \d+ f*
\da[S] [PATTERN]	Lists all available aggregate functions, together with their return value types and the data types.	If **PATTERN** is specified, only aggregate functions whose names match **PATTERN** are displayed.	Lists all available aggregate functions whose names start with **f**, together with their return value types and the data types. \da f*
\db[+] [PATTERN]	Lists all available tablespaces.	If **PATTERN** is specified, only tablespaces whose names match **PATTERN** are displayed.	Lists all available tablespaces whose names start with **p**. \db p*

Parameter	Description	Value Range	Example
\dc[S+] [PATTERN]	Lists all available conversions between character sets.	If **PATTERN** is specified, only conversions whose names match **PATTERN** are displayed.	Lists all available conversions between character sets. \dc *
\dC[+] [PATTERN]	Lists all type conversions.	If **PATTERN** is specified, only conversions whose names match **PATTERN** are displayed.	Lists all type conversion whose patten names start with **c**. \dC c*
\dd[S] [PATTERN]	Lists descriptions about objects matching **PATTERN**.	If **PATTERN** is not specified, all visible objects are displayed. The objects include aggregations, functions, operators, types, relations (table, view, index, sequence, and large object), and rules.	Lists all visible objects. \dd
\ddp [PATTERN]	Lists all default permissions.	If **PATTERN** is specified, only permissions whose names match **PATTERN** are displayed.	Lists all default permissions. \ddp
\dD[S+] [PATTERN]	Lists all available domains.	If **PATTERN** is specified, only domains whose names match **PATTERN** are displayed.	Lists all available domains. \dD
\ded[+] [PATTERN]	Lists all Data Source objects.	If **PATTERN** is specified, only objects whose names match **PATTERN** are displayed.	Lists all Data Source objects. \ded

Parameter	Description	Value Range	Example
\det[+] [PATTERN]	Lists all external tables.	If **PATTERN** is specified, only tables whose names match **PATTERN** are displayed.	Lists all external tables. `\det`
\des[+] [PATTERN]	Lists all external servers.	If **PATTERN** is specified, only servers whose names match **PATTERN** are displayed.	Lists all external servers. `\des`
\deu[+] [PATTERN]	Lists user mappings.	If **PATTERN** is specified, only information whose name matches **PATTERN** is displayed.	Lists user mappings. `\deu`
\dew[+] [PATTERN]	Lists foreign-data wrappers.	If **PATTERN** is specified, only data whose name matches **PATTERN** is displayed.	Lists foreign-data wrappers. `\dew`
\df[antw][S+] [PATTERN]	Lists all available functions, together with their parameters and return types. **a** indicates an aggregate function, **n** indicates a common function, **t** indicates a trigger, and **w** indicates a window function.	If **PATTERN** is specified, only functions whose names match **PATTERN** are displayed.	Lists all available functions, together with their parameters and return types. `\df`
\dF[+] [PATTERN]	Lists all text search configurations.	If **PATTERN** is specified, only configurations whose names match **PATTERN** are displayed.	Lists all text search configurations. `\dF+`

Parameter	Description	Value Range	Example
\dFd[+] [PATTERN]	Lists all text search dictionaries.	If **PATTERN** is specified, only dictionaries whose names match **PATTERN** are displayed.	Lists all text search dictionaries. `\dFd`
\dFp[+] [PATTERN]	Lists all text search parsers.	If **PATTERN** is specified, only analyzers whose names match **PATTERN** are displayed.	Lists all text search parsers. `\dFp`
\dFt[+] [PATTERN]	Lists all text search templates.	If **PATTERN** is specified, only templates whose names match **PATTERN** are displayed.	Lists all text search templates. `\dFt`
\dg[+] [PATTERN]	Lists all database roles. **NOTE** Since the concepts of "users" and "groups" have been unified into "roles", this command is now equivalent to **\du**. The two commands are all reserved for forward compatibility.	If **PATTERN** is specified, only roles whose names match **PATTERN** are displayed.	List all database roles whose names start with **j** and end with **e**. `\dg j?e`
\dl	This is an alias for **\lo_list**, which shows a list of large objects.	-	Lists all large objects. `\dl`
\dL[S+] [PATTERN]	Lists available procedural languages.	If **PATTERN** is specified, only languages whose names match **PATTERN** are displayed.	Lists available procedural languages. `\dL`
\dn[S+] [PATTERN]	Lists all schemas (namespace).	If **PATTERN** is specified, only schemas whose names match **PATTERN** are displayed. By default, only schemas you created are displayed.	Lists information about all schemas whose names start with **d**. `\dn+ d*`

Parameter	Description	Value Range	Example
\do[S] [PATTERN]	Lists available operators with their operand and return types.	If **PATTERN** is specified, only operators whose names match **PATTERN** are displayed. By default, only operators you created are displayed.	Lists available operators with their operand and return types. \do
\dO[S+] [PATTERN]	Lists collations.	If **PATTERN** is specified, only collations whose names match **PATTERN** are displayed. By default, only collations you created are displayed.	Lists collations. \dO
\dp [PATTERN]	Lists tables, views, and related permissions. The following result about **\dp** is displayed: rolename=xxxx/yyyy --Assigning permissions to a role =xxxx/yyyy --Assigning permissions to public *xxxx* indicates the assigned permissions, and *yyyy* indicates the roles that are assigned to the permissions. For details about permission descriptions, see **Table 3-19**.	If **PATTERN** is specified, only tables and views whose names match **PATTERN** are displayed.	Lists tables, views, and related permissions. \dp
\drds [PATTERN1 [PATTERN2]]	Lists all modified configuration parameters. These settings can be for roles, for databases, or for both. **PATTERN1** and **PATTERN2** indicate a role pattern and a database pattern, respectively.	If **PATTERN** is specified, only collations whose names match **PATTERN** are displayed. If the default value is used or * is specified, all settings are listed.	Lists all modified configuration parameters of the database. \drds *

Parameter	Description	Value Range	Example
\dRp[+] [PATTERN]	Lists all publications. This meta-command is supported only by clusters of version 8.2.0.100 or later.	If **PATTERN** is specified, only publications whose names match the pattern are listed.	Lists all publications. \dRp
\dRs[+] [PATTERN]	Lists all subscriptions. This meta-command is supported only by clusters of version 8.2.0.100 or later.	If **PATTERN** is specified, only subscriptions whose names match the pattern are listed.	Lists all subscriptions. \dRs
\dT[S+] [PATTERN]	Lists all data types.	If **PATTERN** is specified, only types whose names match **PATTERN** are displayed.	Lists all data types. \dT
\du[+] [PATTERN]	Lists all database roles. **NOTE** Since the concepts of "users" and "groups" have been unified into "roles", this command is now equivalent to **\dg**. The two commands are all reserved for forward compatibility.	If **PATTERN** is specified, only roles whose names match **PATTERN** are displayed.	Lists all database roles. \du
\dE[S+] [PATTERN] \di[S+] [PATTERN] \ds[S+] [PATTERN] \dt[S+] [PATTERN] \dv[S+] [PATTERN]	In this group of commands, the letters E, i, s, t, and v stand for a foreign table, index, sequence, table, or view, respectively. You can specify any or a combination of these letters sequenced in any order to obtain an object list. For example, **\dit** lists all indexes and tables. If a command is suffixed with a plus sign (+), physical dimensions and related descriptions of each object will be displayed. **NOTE** This version does not support sequences.	If **PATTERN** is specified, only objects whose names match **PATTERN** are displayed. By default, only objects you created are displayed. You can specify **PATTERN** or **S** to view other system objects.	Lists all indexes and views. \div

Parameter	Description	Value Range	Example
\dx[+] [PATTERN]	Lists installed extensions.	If **PATTERN** is specified, only extensions whose names match **PATTERN** are displayed.	Lists installed extensions. `\dx`
\l[+]	Lists the names, owners, character set encoding, and permissions of all databases on the server.	-	Lists the names, owners, character set encoding, and permissions of all databases on the server. `\l`

\sf[+] FUNCNAME	Shows function definitions.
	NOTE
	If the function name contains parentheses, enclose the function name with quotation marks and add the parameter type list following the double quotation marks. Also enclose the list with parentheses.
		-	Assume a function function_a and a function func()name. This parameter will be as follows:
			`\sf function_a`
			`\sf "func()name"(argtype1, argtype2)`

\z [PATTERN]	Lists all tables, views, and sequences in the database and their access permissions.	If a pattern is given, it is a regular expression, and only matched tables, views, and sequences are displayed.	Lists all tables, views, and sequences in the database and their access permissions. `\z`

Table 3-19 Permission descriptions

Parameter	Description
r	SELECT: allows users to read data from specified tables and views.
w	UPDATE: allows users to update columns for specified tables.
a	INSERT: allows users to insert data to specified tables.
d	DELETE: allows users to delete data from specified tables.

Parameter	Description
D	TRUNCATE: allows users to delete all data from specified tables.
x	REFERENCES: allows users to create foreign key constraints.
t	TRIGGER: allows users to create a trigger on specified tables.
X	EXECUTE: allows users to use specified functions and the operators that are realized by the functions.
U	USAGE: ● For procedural languages, allows users to specify a procedural language when creating a function. ● For schemas, allows users to access objects includes in specified schemas. ● For sequences, allows users to use the nextval function.
C	CREATE: ● For databases, allows users to create schemas within a database. ● For schemas, allows users to create objects in a schema. ● For tablespaces, allows users to create tables in a tablespace and set the tablespace to default one when creating databases and schemas.
c	CONNECT: allows users to access specified databases.
T	TEMPORARY: allows users to create temporary tables.
A	ANALYZE|ANALYSE: allows users to analyze tables.
L	**ALTER**: allows users to modify tables or schemas.
P	**DROP**: allows users to delete tables or schemas.
v	**VACUUM**: allows users to perform VACUUM on tables.
arwdDxtA, vLP	ALL PRIVILEGES: grants all available permissions to specified users or roles at a time. The table-level permissions ALTER, DROP, and VACUUM and schema permissions ALTER and DROP are added to the **vLP** permission group in cluster versions 8.1.3 and later.
*	Authorization options for preceding permissions

Table 3-20 Formatting meta-commands

Parameter	Description
\a	Controls the switchover between unaligned mode and aligned mode.
\C [STRING]	Sets the title of any table being printed as the result of a query or cancels such a setting.
\f [STRING]	Sets a field separator for unaligned query output.
\H	• If the text format schema is used, switches to the HTML format. • If the HTML format schema is used, switches to the text format.
\pset NAME [VALUE]	Sets options affecting the output of query result tables. For details about the value of **NAME**, see **Table 3-21**.
\t [on|off]	Switches the information and row count footer of the output column name.
\T [STRING]	Specifies attributes to be placed within the table tag in HTML output format. If the parameter is not configured, the attributes are not set.
\x [on|off|auto]	Switches expanded table formatting modes.

Table 3-21 Adjustable printing options

Option	Description	Value Range
border	The value must be a number. In general, a larger number indicates wider borders and more table lines.	• The value is an integer greater than 0 in HTML format. • The value range in other formats is as follows: – 0: no border – 1: internal dividing line – 2: table frame

Option	Description	Value Range
expanded (or x)	Switches between regular and expanded formats.	• When expanded format is enabled, query results are displayed in two columns, with the column name on the left and the data on the right. This format is useful if the data does not fit the screen in the normal "horizontal" format. • The expanded format is used when the query output is wider than the screen. Otherwise, the regular format is used. The regular format is effective only in the aligned and wrapped formats.
fieldsep	Specifies the field separator to be used in unaligned output format. In this way, you can create tab- or comma-separated output required by other programs. To set a tab as field separator, type **\pset fieldsep '\t'**. The default field separator is a vertical bar ('|').	-
fieldsep_zero	Sets the field separator to be used in unaligned output format to zero bytes.	-
footer	Enables or disables the display of table footers.	-

Option	Description	Value Range
format	Selects the output format. Unique abbreviations are allowed. (That means a single letter is sufficient.)	Value range: • **unaligned**: Write all columns of a row on one line, separated by the currently active column separator. • **aligned**: This format is standard and human-readable. • **wrapped**: This format is similar to **aligned**, but includes the packaging cross-line width data value to suit the width of the target field output. • **html**: This format output table to the markup language for a document. The output is not a complete document. • **latex**: This format output table to the markup language for a document. The output is not a complete document. • **troff-ms:** This format output table to the markup language for a document. The output is not a complete document.
null	Sets a character string to be printed in place of a null value.	By default, nothing is printed, which can easily be mistaken for an empty character string.
numericlocale	Enables or disables the display of a locale-specific character to separate groups of digits to the left of the decimal marker.	• **on**: The specified separator is displayed. • **off**: The specified separator is not displayed If this parameter is ignored, the default separator is displayed.

Option	Description	Value Range
pager	Controls the use of a pager for query and gsql help outputs. If the PAGER environment variable is set, the output is piped to the specified program. Otherwise, a platform-dependent default is used.	• **on**: The pager is used for terminal output that does not fit the screen. • **off**: The pager is not used. • **always**: The pager is used for all terminal output regardless of whether it fits the screen.
recordsep	Specifies the record separator to be used in unaligned output format.	-
recordsep_zero	Specifies the record separator to be used in unaligned output format to zero bytes.	-
tableattr (or T)	Specifies attributes to be placed inside the HTML table tag in HTML output format (such as cellpadding or bgcolor). Note that you do not need to specify border here because it has been used by **\pset border**. If no value is given, the table attributes do not need to be set.	-
title	Specifies the table title for any subsequently printed tables. This can be used to give your output descriptive tags. If no value is given, the title does not need to be set.	-
tuples_only (or t)	Enables or disables the tuples-only mode. Full display may show extra information, such as column headers, titles, and footers. In tuples-only mode, only the table data is displayed.	-

Table 3-22 Connection meta-commands

Parameter	Description	Value Range
\c[onnect] [DBNAME|- USER|- HOST|- PORT|-]	Connects to a new database. (The current database is **gaussdb**.) If a database name contains more than 63 bytes, only the first 63 bytes are valid and are used for connection. However, the database name displayed in the gsql CLI is still the name before the truncation. **NOTE** If the database login user is changed during reconnection, you need to enter the password of the new user. The maximum length of the password is 999 bytes, which is restricted by the maximum value of the GUC parameter **password max length**.	-
\encoding [ENCODING]	Sets the client character set encoding.	This command shows the current encoding if it has no parameter.
\conninfo	Outputs information about the current database connection.	-

Table 3-23 OS meta-commands

Parameter	Description	Value Range
\cd [DIR]	Changes the current working directory.	An absolute path or relative path that meets the OS path naming convention
\setenv NAME [VALUE]	Sets the **NAME** environment variable to **VALUE**. If **VALUE** is not provided, do not set the environment variable.	-
\timing [on|off]	Toggles a display of how long each SQL statement takes, in milliseconds.	• The value **on** indicates that the setting is enabled. • The value **off** indicates that the setting is disabled.
\! [COMMAND]	Escapes to a separate Unix shell or runs a Unix command.	-

Table 3-24 Variable meta-commands

Parameter	Description
\prompt [TEXT] NAME	Prompts the user to use texts to specify a variable name.
\set [NAME [VALUE]]	Sets the *NAME* internal variable to **VALUE**. If more than one value is provided, *NAME* is set to the concatenation of all of them. If only one parameter is provided, the variable is set with an empty value. Some common variables are processed in another way in **gsql**, and they are the combination of uppercase letters, numbers, and underscore. **Table 3-25** describes a list of variables that are processed in a way different from other variables.
\set-multi NAME [VALUE] \end-multi	Sets the internal variable NAME to VALUE that can consist of multiple lines of character strings. When **\set-multi** is used, the second parameter must be provided. For details, see the following example of using the **\set-multi** meta-command. **NOTE** The meta-commands in **\set-multi** and **\end-multi** will be ignored.
\unset NAME	Deletes the variable name of **gsql**.

\set-multi meta-command example

Sample file **test.sql**:

```
\set-multi multi_line_var
select
    id,name
from
    student;
\end-multi
\echo multi_line_var is "${multi_line_var}"
\echo ------------------------
\echo result is
${multi_line_var}
```

gsql -d gaussdb -p 25308 --dynamic-param -f test.sql execution result:

```
multi_line_var is "select
        id,name
from
        student; "
------------------------
result is
 id | name
----+-------
  1 | Jack
  2 | Tom
  3 | Jerry
  4 | Danny
(4 rows)
```


Run the **\set-multi \end-multi** command to set the variable **multi_line_var** to one SQL statement and obtain the variable through dynamic variable parsing.

Sample file **test.sql**:

```
\set-multi multi_line_var
select 1 as id;
select 2 as id;
\end-multi
\echo multi_line_var is "${multi_line_var}"
\echo -------------------------
\echo result is
${multi_line_var}
```

gsql -d -p 25308 --dynamic-param -f test.sql execution result:

```
multi_line_var is "select 1 as id;
select 2 as id;"
-------------------------
result is
 id
----
  1
(1 row)

 id
----
  2
(1 row)
```

Run the **\set-multi \end-multi** command to set the variable **multi_line_var** to two SQL statements and obtain the variable through dynamic variable parsing. Because the content in the variable ends with a semicolon (;), gsql sends the SQL statement and obtains the printed execution result.

Table 3-25 Common \set commands

Command	Description	Value Range
\set VERBOSITY value	This variable can be set to **default**, **verbose**, or **terse** to control redundant lines of error reports.	Value range: **default**, **verbose**, **terse**
\set ON_ERROR_STOP value	If this variable is set, the script execution stops immediately. If this script is invoked from another script, that script will be stopped immediately as well. If the primary script is invoked using the **-f** option rather than from one **gsql** session, **gsql** will return error code 3, indicating the difference between the current error and critical errors. (The error code for critical errors is 1.)	Value range: **on/off**, **true/false**, **yes/no**, **1/0**

Command	Description	Value Range
\set RETRY [retry_times]	Determines whether to enable the retry function if statement execution encounters errors. The parameter **retry_times** specifies the maximum number of retry times and the default value is **5**. Its value ranges from **5** to **10**. If the retry function has been enabled, when you run the **\set RETRY** command again, the retry function will be disabled. The configuration file **retry_errcodes.conf** shows a list of errors. If these errors occur, retry is required. This configuration file is placed in the same directory as that for executable gsql programs. This configuration file is configured by the system rather than by users and cannot be modified by the users. The retry function can be used in the following 13 error scenarios: ● YY001: TCP communication errors. Print information: **Connection reset by peer**. (reset between CN and DN) ● YY002: TCP communication errors. Print information: **Connection reset by peer**. (reset between DN and DN) ● YY003: Lock timeout. Print information: **Lock wait timeout...**/**wait transaction** *xxx* **sync time exceed** *xxx*. ● YY004: TCP communication errors. Print information: **Connection timed out**. ● YY005: Failed to issue SET commands. Print information: **ERROR SET query**. ● YY006: Failed to apply for memory. Print information: **memory is temporarily unavailable**. ● YY007: Communication library error. Print information: **Memory allocate error**. ● YY008: Communication library error. Print information: **No data in buffer**. ● YY009: Communication library error. Print information: **Close because release memory**. ● YY010: Communication library error. Print information: **TCP disconnect**. ● YY011: Communication library error. Print information: **SCTP disconnect**.	Value range of **retry_times**: **5** to **10**

Command	Description	Value Range
	• YY012: Communication library error. Print information: **Stream closed by remote**. • YY013: Communication library error. Print information: **Wait poll unknown error**. • YY014: Invalid snapshot. Print information: **Snapshot invalid**. • YY015: Failed to receive connection. Print information: **Connection receive wrong**. • 53200: Out of memory. Print information: **Out of memory**. • 08006: GTM error. Print information: **Connection failure**. • 08000: Failed to communicate with DNs due to connection errors. Print information: **Connection exception**. • 57P01: System shutdown by administrators. Print information: **Admin shutdown**. • XX003: Remote socket is disabled. Print information: **Stream remote close socket**. • XX009: Duplicate query IDs. Print information: **Duplicate query id**. • YY016: Concurrent stream query and update. Print information: **Stream concurrent update**. • CG003: Memory allocation error. Print information: **Allocate Error**. • CG004: Fatal error. Print information: **Fatal error**. • F0011: Temporary file reading error. Print information: **File error**. If an error occurs, **gsql** queries connection status of all CNs and DNs. If the connection status is abnormal, **gsql** sleeps for 1 minute and tries again. In this case, the retries in most of the primary/standby switchover scenarios are involved.	

Command	Description	Value Range
	NOTE 1. Statements in transaction blocks cannot be retried upon a failure. 2. Retry is not supported if errors are found using ODBC or JDBC. 3. For SQL statements with unlogged tables, the retry is not supported if a node is faulty. 4. If a CN or GTM is faulty, the retry on the gsql client is not supported. 5. For gsql client faults, the retry is not supported.	

Table 3-26 Large object meta-commands

Parameter	Description
\lo_list	Displays all GaussDB(DWS) large objects currently stored in the database and their descriptions.

Table 3-27 Flow control meta-commands

Parameter	Description	Value Range
\if EXPR \elif EXPR \else \endif	This set of meta-commands can implement nested conditional blocks: • A conditional block starts with **\if** and ends with **\endif**. • Any number of **\elif** clauses or a single **\else** clause can exist between **\if** and **\endif**. • The **\if** and **\elif** commands support Boolean expression calculations and can check whether two strings are equal. • **\elif** cannot be used between **\else** and **\endif**.	• The Boolean expression calculation is the same as that of gsql: **true**/**false**, **yes**/**no**, **on**/**off**, and **1**/**0**. Any other value is considered as **true**. • The operator and the string must be separated using spaces. • Number comparison and string comparison are supported. The comparison rules are controlled by the inherent variable **COMPARE_STRATEGY**. For details, see **Table 3-2**. In the default comparison rule, single quotation marks (') are used to separate strings and numbers. For details about the examples of different rules, see **\if conditional block comparison rules and examples**. • The following operators can be used to compare the values of numbers and strings and perform equality comparison: <, <=, >, >=, ==, !=, and <>.

Parameter	Description	Value Range
\goto LABEL \label LABEL	This set of meta-commands can be used to implement unconditional redirections: • The **\label** meta-command is used to create a label. • The **\goto** meta-command is used to redirect upward or downward. **NOTE** • The interactive mode is not supported. • The **\label** meta-command is not supported in the **\if** statement block. • To avoid unexpected results, you are not advised to use the **\goto** and **\label** commands in transactions or PL/SQL statement blocks.	• The label value is case sensitive. • The value of a label can contain uppercase letters, lowercase letters, digits, and underscores (_). • The maximum length of a label is 32 characters (including **\0**). If the length exceeds 32 characters, the label will be truncated and an alarm will be generated. • If the label name following **\label** appears repeatedly in the same session, an error is reported.

Parameter	Description	Value Range
\for \loop \exit-for \end-for	This set of meta-commands can be used to implement loops: • The loop block starts with **\for** and ends with **\end-for**. • The condition between **\for** and **\loop** is a loop condition. Only SQL statements are supported. Variable iteration is not supported. For example, **\for (i=0; i<100; ++i)** is not supported. • If there are multiple SQL statements in the loop condition, the execution result of the last SQL statement is used as the loop condition. The SQL statement used as a loop condition cannot end with a semicolon (;). • The loop body exists between **\loop** and **\end-for**. You can run **\exit-for** to exit the loop. • The **\for** loop block supports multi-layer nesting. **NOTE** • The interactive mode is not supported. • **\for** loop blocks cannot be used in **\parallel**. • To avoid unexpected results, you are not advised to use the **\for** loop block in transactions or PL/SQL statement blocks. • The **\label** meta-command is not supported in the **\for** loop block. • Anonymous blocks cannot be used between **\for** and **\loop**.	-

An example of using flow control meta-commands is as follows:

- **\if** conditional block use example:

  Sample file **test.sql**:

```
SELECT 'Jack' AS "Name";

\if ${ERROR}
   \echo 'An error occurred in the SQL statement'
   \echo ${LAST_ERROR_MESSAGE}
\elif '${Name}' == 'Jack'
   \echo 'I am Jack'
```



```
\else
    \echo 'I am not Jack'
\endif
```

gsql -d -p 25308 --dynamic-param -f test.sql execution result:

```
 Name
------
 Jack
(1 row)

I am Jack
```

The preceding execution result indicates that the first SQL statement is successfully executed and the **Name** variable is set. Therefore, the **\elif** branch is executed and the output is **I am Jack**. For details about the usage of the special variables **ERROR** and **LAST_ERROR_MESSAGE**, see **Table 3-2**.

- **\if** conditional block comparison rules and examples
  - **default**: Specifies the default comparison policy. Only strings or numbers can be compared, and strings cannot be compared with numbers. Parameters inside single quotation marks (') are identified as strings, and parameters outside single quotation marks (') are identified as numbers.

  The file **test.sql** is used as an example.

  ```
  \set Name 'Jack'
  \set ID 1002

  -- Parameters inside single quotation marks (') are identified as strings for comparison.
  \if '${Name}' != 'Jack'
      \echo 'I am not Jack'
  -- Without single quotation marks ('), parameters are identified as numbers for comparison.
  \elif ${ID} > 1000
      \echo 'Jack\'id is bigger than 1000'
  \else
      \echo 'error'
  \endif
  ```

  gsql -d -p 25308 --dynamic-param -f test.sql execution result:

  ```
  Jack'id is bigger than 1000
  ```

  If a single quotation mark (') is used on one side of the operator and not used on the other side, the comparison is performed between a string and a number. Such comparison is not supported and an error is reported.

  ```
  postgres=> \set Name 'Jack'
  postgres=> \if ${Name} == 'Jack'
  ERROR: left[Jack] is a string without quote or number, and right['Jack'] is a string with quote, \if or \elif does not support this expression.
  WARNING: The input with quote are treated as a string, and the input without quote are treated as a number.
  postgres@> \endif
  ```

  - **natural**: The default comparison policy is supported, and parameters that contain dynamic variables are also identified as strings. When one side of the comparison operator is a number, try to convert the other side to a number, and then compare the numbers on both sides. If the conversion fails, an error is reported and the comparison result is false.
    - Parameters that contain dynamic variables can be identified as strings when single quotation marks (') are used, for example, 'Jack', or the string contains a dynamic variable (**${VAR}** or **:VAR**), for example, ${Name}_data. If both of the preceding conditions are met, for example, '${Name}_data', parameters that contain dynamic variables can also be identified as strings.

- For parameters that cannot be identified as strings, try to identify them as numbers. If a parameter cannot be converted to a number, an error is reported. For example, **1011Q1** does not use single quotation marks (') or contain dynamic variables, and cannot be converted to a number.
- If one side of the comparison operator is not identified as a string or number, the comparison fails and an error is reported.
- If one side of the comparison operator is identified as a number, the comparison is performed based on numbers. If the other side cannot be converted to a number, an error is reported.
- If both sides of the comparison operator are identified as strings, the comparison is performed based on strings.

The **test.sql** file is an example of comparing strings.

```
\set COMPARE_STRATEGY natural
SELECT 'Jack' AS "Name";

-- The comparison result is equivalent to that of '${Name}' > 'Jack'.
\if ${Name} == 'Jack'
    \echo 'I am Jack'
\else
    \echo 'I am not Jack'
\endif
```

gsql -d -p 25308 --dynamic-param -f test.sql execution result:

```
 Name
------
 Jack
(1 row)

I am Jack
```

The **test.sql** file is an example of comparing numbers.

```
\set COMPARE_STRATEGY natural
SELECT 1022 AS id;

-- If ${id} == '01022' is used, the result is not equal because strings on both sides are compared.
\if ${id} == 01022
    \echo 'id is 1022'
\else
    \echo 'id is not 1022'
\endif
```

gsql -d -p 25308 --dynamic-param -f test.sql execution result:

```
  id
------
 1022
(1 row)

id is 1022
```

Examples of comparison errors are shown as follows.

```
-- One side of the operator cannot be identified as a string or number.
postgres=> \set COMPARE_STRATEGY natural
postgres=> \if ${Id} > 123sd
ERROR: The right[123sd] can not be treated as a string or a number. A numeric string should contain only digits and one decimal point, and a string should be enclosed in quote or contain dynamic variables, please check it.
-- Numbers on one side of the operator cannot be correctly converted.
postgres=> \set COMPARE_STRATEGY natural
postgres=> \if ${Id} <> 11101.1.1
ERROR: The right[11101.1.1] can not be treated as a string or a number. A numeric string should
```



```
contain only digits and one decimal point, and a string should be enclosed in quote or contain dynamic variables, please check it.
```

  - **equal**: Only the equality comparison is supported. The comparison is performed based on strings.

  The file **test.sql** is used as an example.

```
\set COMPARE_STRATEGY equal
SELECT 'Jack' AS "Name";

\if ${ERROR}
    \echo 'An error occurred in the SQL statement'
-- If the value is set to equal, only the equality comparison is supported. An error is reported when the values are compared, and there is no delimiter. The following comparison result is equivalent to that of ${Name} == Jack.
\elif '${Name}' == 'Jack'
    \echo 'I am Jack'
\else
    \echo 'I am not Jack'
\endif
```

  gsql -d -p 25308 --dynamic-param -f test.sql execution result:

```
 Name
------
 Jack
(1 row)

I am Jack
```

- **\goto \label** redirection example:

  Sample file **test.sql**:

```
\set Name Tom

\goto TEST_LABEL
SELECT 'Jack' AS "Name";

\label TEST_LABEL
\echo ${Name}
```

  gsql -d -p 25308 --dynamic-param -f test.sql execution result:

```
Tom
```

  The preceding execution result indicates that the **\goto** meta-command directly executes the **\echo** command without re-assigning a value to the variable **Name**.

- Example of using **\if** conditional block and the **\goto \label** together

  Sample file **test.sql**:

```
\set Count 1

\label LOOP
\if ${Count} != 3
    SELECT ${Count} + 1 AS "Count";
    \goto LOOP
\endif

\echo Count = ${Count}
```

  gsql -d -p 25308 --dynamic-param -f test.sql execution result:

```
 Count
-------
     2
(1 row)

 Count
-------
     3
```



```
(1 row)

Count = 3
```

The preceding execution result indicates that a simple loop is implemented through the combination of the **\if** conditional block and **\goto \label**.

- Example of Using **\for** Loop Blocks

  To demonstrate this function, the example data is as follows:

```
create table student (id int, name varchar(32));
insert into student values (1, 'Jack');
insert into student values (2, 'Tom');
insert into student values (3, 'Jerry');
insert into student values (4, 'Danny');

create table course (class_id int, class_day varchar(5), student_id int);
insert into course values (1004, 'Fri', 2);
insert into course values (1003, 'Tue', 1);
insert into course values (1003, 'Tue', 4);
insert into course values (1002, 'Wed', 3);
insert into course values (1001, 'Mon', 2);
```

  \for loop use sample file **test.sql**:

```
\for
select id, name from student order by id limit 3 offset 0
\loop
    \echo -[ RECORD ]+-----
    \echo id '\t'| ${id}
    \echo name '\t'| ${name}
\end-for
```

  gsql -d -p 25308 --dynamic-param -f test.sql execution result:

```
-[ RECORD ]+-----
id      | 1
name    | Jack
-[ RECORD ]+-----
id      | 2
name    | Tom
-[ RECORD ]+-----
id      | 3
name    | Jerry
```

  The preceding execution result indicates that the loop block is used to traverse the execution result of the SQL statement. More statements can appear between **\loop** and **\end-for** to implement complex logic.

  If the SQL statement used as a loop condition fails to be executed or the result set is empty, the statement between **\loop** and **\end-for** will not be executed.

  Sample file **test.sql**:

```
\for
select id, name from student_error order by id limit 3 offset 0
\loop
    \echo -[ RECORD ]+-----
    \echo id '\t'| ${id}
    \echo name '\t'| ${name}
\end-for
```

  gsql -d -p 25308 --dynamic-param -f test.sql execution result:

```
gsql:test.sql:3: ERROR:  relation "student_error" does not exist
LINE 1: select id, name from student_error order by id limit 3 offse...
                             ^
```

  The preceding command output indicates that the **student_error** table does not exist. Therefore, the SQL statement fails to be executed, and the statement between **\loop** and **\end-for** is not executed.

- **\exit-for** exits the loop.

  Sample file **test.sql**:

  ```
  \for
  select id, name from student order by id
  \loop
      \echo ${id} ${name}
      \if ${id} == 2
          \echo find id(2), name is ${name}
          \exit-for
      \endif
  \end-for
  ```

  gsql -d -p 25308 --dynamic-param -f test.sql execution result:

  ```
  1 Jack
  2 Tom
  find id(2), name is Tom
  ```

  If the **student** table contains more than two rows of data and **id** is set to **2**, run the **\exit-for** command to exit the loop. This process is also used together with the **\if** condition block.

- **\for** loop nesting

  Sample file **test.sql**:

  ```
  \for
  select id, name from student order by id limit 2 offset 0
  \loop
      \echo ${id} ${name}
      \for
      select
      class_id, class_day
      from course
      where student_id = ${id}
      order by class_id
      \loop
          \echo '  '${class_id}, ${class_day}
      \end-for
  \end-for
  ```

  gsql -d -p 25308 --dynamic-param -f test.sql execution result:

  ```
  1 Jack
    1003, Tue
  2 Tom
    1001, Mon
    1004, Fri
  ```

  Obtain the information about Jack and Tom in the **course** table through the two-layer loop.

PATTERN

The various **\d** commands accept a **PATTERN** parameter to specify the object name to be displayed. In the simplest case, a pattern is just the exact name of the object. The characters within a pattern are normally folded to lower case, similar to those in SQL names. For example, **\dt FOO** will display the table named **foo**. As in SQL names, placing double quotation marks (") around a pattern prevents them being folded to lower case. If you need to include a double quotation mark (") in a pattern, write it as a pair of double quotation marks ("") within a double-quote sequence, which is in accordance with the rules for SQL quoted identifiers. For example, **\dt "FOO""BAR"** will be displayed as a table named **FOO"BAR** instead of **foo"bar**. You cannot put double quotation marks around just part of a pattern, which is different from the normal rules for SQL names. For example, **\dt FOO"FOO"BAR** will be displayed as a table named **fooFOObar** if just part of a pattern is quoted.

Whenever the **PATTERN** parameter is omitted completely, the **\d** commands display all objects that are visible in the current schema search path, which is equivalent to using an asterisk (*) as the pattern. An object is regarded to be visible if it can be referenced by name without explicit schema qualification. To see all objects in the database regardless of their visibility, use a dot within double quotation marks (*.*) as the pattern.

Within a pattern, the asterisk (*) matches any sequence of characters (including no characters) and a question mark (?) matches any single character. This notation is comparable to Unix shell file name patterns. For example, **\dt int*** displays tables whose names begin with **int**. But within double quotation marks, the asterisk (*) and the question mark (?) lose these special meanings and are just matched literally.

A pattern that contains a dot (.) is interpreted as a schema name pattern followed by an object name pattern. For example, **\dt foo*.*bar*** displays all tables (whose names include **bar**) in schemas starting with **foo**. If no dot appears, then the pattern matches only visible objects in the current schema search path. Again, a dot within double quotation marks loses its special meaning and is matched literally.

Advanced users can use regular-expression notations, such as character classes. For example [0-9] can be used to match any digit. All regular-expression special characters work as specified in "POSIX regular expressions" in the *Developer Guide*, except the following characters:

- A dot (.) is used as a separator.
- An asterisk (*) is translated into an asterisk prefixed with a dot (.*), which is a regular-expression marking.
- A question mark (?) is translated into a dot (.).
- A dollar sign ($) is matched literally.

You can write ?, (*R*+|), (*R*|), and *R* to the following pattern characters: ., *R*.*, and *R*.?. The dollar sign ($) does not need to work as a regular-expression character since the pattern must match the whole name, which is different from the usual interpretation of regular expressions. In other words, the dollar sign ($) is automatically appended to your pattern. If you do not expect a pattern to be anchored, write an asterisk (*) at its beginning or end. All regular-expression special characters within double quotation marks lose their special meanings and are matched literally. Regular-expression special characters in operator name patterns (such as the **\do** parameter) are also matched literally.

3.7 Troubleshooting

Low Connection Performance

- The database kernel slowly runs the initialization statement.

  Problems are difficult to locate in this scenario. Try using the **strace** Linux trace command.

  ```
  strace gsql -U MyUserName -W {password} -d postgres -h 127.0.0.1 -p 23508 -r -c '\q'
  ```

  The database connection process will be printed on the screen. If the following statement takes a long time to run:


```
sendto(3, "Q\0\0\0\25SELECT VERSION()\0", 22, MSG_NOSIGNAL, NULL, 0) = 22
poll([{fd=3, events=POLLIN|POLLERR}], 1, -1) = 1 ([{fd=3, revents=POLLIN}])
```

It indicates that **SELECT VERSION()** statement was run slowly.

After the database is connected, you can run the **explain performance select version()** statement to find the reason why the initialization statement was run slowly. For details, see "Introduction to the SQL Execution Plan" in the *Developer Guide*.

An uncommon scenario is that the disk of the machine where the CN resides is full or faulty, affecting queries and leading to user authentication failures. As a result, the connection process is suspended. To solve this problem, simply clear the data disk space of the CN.

- TCP connection is set up slowly.

  Adapt the steps of troubleshooting slow initialization statement execution. Use **strace**. If the following statement was run slowly:

  ```
  connect(3, {sa_family=AF_FILE, path="/home/test/tmp/gaussdb_llt1/.s.PGSQL.61052"}, 110) = 0
  ```

  Or

  ```
  connect(3, {sa_family=AF_INET, sin_port=htons(61052), sin_addr=inet_addr("127.0.0.1")}, 16) = -1
  EINPROGRESS (Operation now in progress)
  ```

  It indicates that the physical connection between the client and the database was set up slowly. In this case, check whether the network is unstable or has high throughput.

Problems in Setting Up Connections

- gsql: could not connect to server: No route to host

  This problem occurs generally because an unreachable IP address or port number was specified. Check whether the values of **-h** and **-p** parameters are correct.

- gsql: FATAL: Invalid username/password,login denied.

  This problem occurs generally because an incorrect user name or password was entered. Contact the database administrator to check whether the user name and password are correct.

- The "libpq.so" loaded mismatch the version of gsql, please check it.

  This problem occurs because the version of **libpq.so** used in the environment does not match that of **gsql**. Run the **ldd gsql** command to check the version of the loaded **libpq.so**, and then load correct **libpq.so** by modifying the environment variable **LD_LIBRARY_PATH**.

- gsql: symbol lookup error: xxx/gsql: undefined symbol: libpqVersionString

  This problem occurs because the version of **libpq.so** used in the environment does not match that of **gsql** (or the PostgreSQL **libpq.so** exists in the environment). Run the **ldd gsql** command to check the version of the loaded **libpq.so**, and then load correct **libpq.so** by modifying the environment variable **LD_LIBRARY_PATH**.

- gsql: connect to server failed: Connection timed out

  Is the server running on host "xx.xxx.xxx.xxx" and accepting TCP/IP connections on port xxxx?

  This problem is caused by network connection faults. Check the network connection between the client and the database server. If you cannot ping

from the client to the database server, the network connection is abnormal. Contact network management personnel for troubleshooting.

```
ping -c 4 10.10.10.1
PING 10.10.10.1 (10.10.10.1) 56(84) bytes of data.
From 10.10.10.1: icmp_seq=2 Destination Host Unreachable
From 10.10.10.1 icmp_seq=2 Destination Host Unreachable
From 10.10.10.1 icmp_seq=3 Destination Host Unreachable
From 10.10.10.1 icmp_seq=4 Destination Host Unreachable
--- 10.10.10.1 ping statistics ---
4 packets transmitted, 0 received, +4 errors, 100% packet loss, time 2999ms
```

- gsql: FATAL: permission denied for database "postgres"

  DETAIL: User does not have CONNECT privilege.

  This problem occurs because the user does not have the permission to access the database. To solve this problem, perform the following steps:

  a. Connect to the database as the system administrator **dbadmin**.

     gsql -d *postgres* **-U** *dbadmin* -p *8000*

  b. Grant the users with the access permission to the database.

     GRANT CONNECT ON DATABASE postgres TO user1;

     > NOTE
     >
     > Common misoperations may also cause a database connection failure, for example, entering an incorrect database name, user name, or password. In this case, the client tool will display the corresponding error messages.
     >
     > **gsql -d** *postgres* **-p** *8000*
     > gsql: FATAL: database "postgres" does not exist
     >
     > **gsql -d** *postgres* **-U** *user1* **-W** *gauss@789* **-p** *8000*
     > gsql: FATAL: Invalid username/password,login denied.

- gsql: FATAL: sorry, too many clients already, active/non-active: 197/3.

  This problem occurs because the number of system connections exceeds the allowed maximum. Contact the database administrator to release unnecessary sessions.

  You can check the number of connections as described in **Table 3-28**.

  You can view the session status in the **PG_STAT_ACTIVITY** view. To release unnecessary sessions, use the pg_terminate_backend function.

  ```
  select datid,pid,state from pg_stat_activity;
   datid |       pid       | state
  -------+-----------------+--------
   13205 | 139834762094352 | active
   13205 | 139834759993104 | idle
  (2 rows)
  ```

  The value of pid is the thread ID of the session. Terminate the session using its thread ID.

  ```
  SELECT PG_TERMINATE_BACKEND(139834759993104);
  ```

  If information similar to the following is displayed, the session is successfully terminated:

  ```
  PG_TERMINATE_BACKEND
  ----------------------
   t
  (1 row)
  ```


Table 3-28 Viewing the numbers of connections

Description	Command
View the upper limit of a user's connections.	Run the following command to view the upper limit of user **USER1**'s connections. **-1** indicates that no connection upper limit is set for user **USER1**. SELECT ROLNAME,ROLCONNLIMIT FROM PG_ROLES WHERE ROLNAME='user1'; rolname | rolconnlimit ---------+-------------- user1 | -1 (1 row)
View the number of connections that have been used by a user.	Run the following command to view the number of connections that have been used by user **user1**. **1** indicates the number of connections that have been used by user **user1**. SELECT COUNT(*) FROM V$SESSION WHERE USERNAME='user1'; count ------- 1 (1 row)
View the upper limit of connections to database.	Run the following command to view the upper limit of connections used by **postgres**. **-1** indicates that no upper limit is set for the number of connections that have been used by **postgres**. SELECT DATNAME,DATCONNLIMIT FROM PG_DATABASE WHERE DATNAME='postgres'; datname | datconnlimit ----------+-------------- postgres | -1 (1 row)
View the number of connections that have been used by a database.	Run the following command to view the number of connections that have been used by **postgres**. **1** indicates the number of connections that have been used by **postgres**. SELECT COUNT(*) FROM PG_STAT_ACTIVITY WHERE DATNAME='postgres'; count ------- 1 (1 row)
View the total number of connections that have been used by users.	Run the following command to view the number of connections that have been used by users: SELECT COUNT(*) FROM V$SESSION; count ------- 10 (1 row)

- gsql: wait xxx.xxx.xxx.xxx:xxxx timeout expired

  When **gsql** initiates a connection request to the database, a 5-minute timeout period is used. If the database cannot correctly authenticate the client request

and client identity within this period, **gsql** will exit the connection process for the current session, and will report the above error.

Generally, this problem is caused by the incorrect host and port (that is, the *xxx* part in the error information) specified by the **-h** and **-p** parameters. As a result, the communication fails. Occasionally, this problem is caused by network faults. To resolve this problem, check whether the host name and port number of the database are correct.

- gsql: could not receive data from server: Connection reset by peer.

  Check whether CN logs contain information similar to "FATAL: cipher file "/data/coordinator/server.key.cipher" has group or world access". This error is usually caused by incorrect tampering with the permissions for data directories or some key files. For details about how to correct the permissions, see related permissions for files on other normal instances.

- gsql: FATAL: GSS authentication method is not allowed because XXXX user password is not disabled.

  In **pg_hba.conf** of the target CN, the authentication mode is set to **gss** for authenticating the IP address of the current client. However, this authentication algorithm cannot authenticate clients. Change the authentication algorithm to **sha256** and try again.

  NOTE

  - Do not modify the configurations of database cluster hosts in the **pg_hba. conf** file. Otherwise, the database may become faulty.
  - You are advised to deploy service applications outside the database cluster.

Other Faults

- There is a core dump or abnormal exit due to the bus error.

  Generally, this problem is caused by changes in loading the shared dynamic library (.so file in Linux) during process running. Alternatively, if the process binary file changes, the execution code for the OS to load machines or the entry for loading a dependent library will change accordingly. In this case, the OS kills the process for protection purposes, generating a core dump file.

  To resolve this problem, try again. In addition, do not run service programs in a cluster during O&M operations, such as an upgrade, preventing such a problem caused by file replacement during the upgrade.

  NOTE

  A possible stack of the core dump file contains dl_main and its function calling. The file is used by the OS to initialize a process and load the shared dynamic library. If the process has been initialized but the shared dynamic library has not been loaded, the process cannot be considered completely started.

4 Data Studio

4.1 About Data Studio

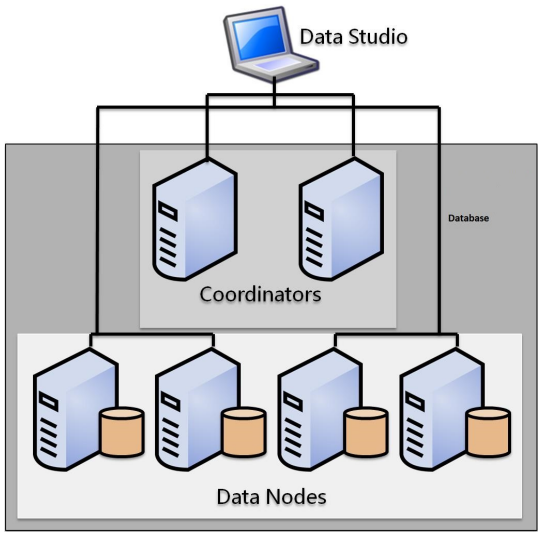
4.1.1 Overview

Data Studio provides a graphical interface which supports essential features of the database. This simplifies database development and application building tasks.

Data Studio allows the database developer to manage and create database objects (database, schema, functions, procedures, tables, sequences, columns, indexes, constraints, views, and tablespaces), execute SQL statements or SQL scripts, edit and execute PL/SQL statements, as well as import and export table data.

Data Studio also allows the database developer to debug and fix defects in the PL/SQL code using debugging operations such as **Step Into**, **Step Out**, **Step Over**, **Continue**, and **Terminate**.

The following figure provides the operational context of database and Data Studio.



Data Studio GUI

Table 4-1 Introduction to the GUI

GUI Name	Description
Main Menu	Provides basic operations of Data Studio.
Toolbar	Provides entries to common operations.
SQL Terminal tab	Executes SQL statements and functions/procedures.
PL/SQL Viewer tab	Displays the content of function/procedures.
CallStack pane	Displays the execution stack.
Breakpoints pane	Displays all breakpoints that have been set.
Variables pane	Shows variables and their values.
SQL Assistant tab	Displays suggestion or reference for the information entered in the **SQL Terminal** and **PL/SQL Viewer**.
Result tab	Displays the result(s) of an executed function/ procedure, or an SQL statement.
Message tab	Displays the process output, standard input, standard output, and standard errors.
Object Browser pane	Contains a hierarchical tree display of database connection(s) and related database objects to which users can access. All created schemas, except the public schemas, are grouped under **Catalogs** by default, and user schemas are grouped under **Schemas** of the corresponding database. **NOTE** **Object Browser** displays only the objects that meet the permission requirements of the current user.
Minimized Window Panel pane	Used to open the **Callstack**, **Breakpoints**, and **Variables** panes. This pane is displayed only when Callstack, Breakpoints or Variables pane or all three are minimized.
Search Toolbar	Used to search objects from **Object browser**.

Performance Specifications

The loading and operation performance of Data Studio depends on the number of objects to be loaded in **Object Browser**, including tables, views, and columns.

Memory consumption also depends on the number of loaded objects.

To improve object loading performance and better utilize memory, you are advised to divide an object into multiple namespaces, and to avoid using namespaces that contain a large number of objects and cause data skew. By default, Data Studio

loads the namespaces in the *search_path* set for the user logged in. Other namespaces and objects are loaded only when needed.

To improve performance, you are advised to load all objects. Do not load objects based on user permissions. **Table 4-2** describes the minimum access permissions required to the listed objects in **Object Browser**.

Table 4-2 Minimum permission requirements

Object Type	Permission Type	Object Browser - Minimum Permission
Databases	Create, Connect, Temporary/Temp, All	Connect
Schema	Create, Usage, All	Usage
Tables	Select, Insert, Update, Delete, Truncate, References, All	Select
Columns	Select, Insert, Update, References, All	Select
Views	Select, Insert, Update, Delete, Truncate, References, All	Select
Sequences	Usage, Select, Update, All	Usage
Functions	Execute, All	Execute

To improve the performance of the finding and replacing operations, you are advised to break a line that contains more than 10,000 characters into multiple short lines.

The following test items and results can help you learn the performance of Data Studio.

Recommended maximum memory (current version)		1.4 GB
Performance (The database contains a 150 KB table and a 150 KB view, each containing three columns. The maximum memory configuration is used.)		
>	Time taken to refresh namespaces in **Object Browser**	15s
>	Time taken for initial loading and expanding of all tables/views in **Object Browser**	90s-120s
>	Time taken for subsequent loading and expanding of all tables/views in **Object Browser**	<10s
>	Total used memory	700 MB

NOTE

The performance data is for reference only. The actual performance may vary according to the application scenario.

4.1.2 Constraints and Limitations

This section describes the constraints and limitations for using Data Studio.

Character Encoding

If the SQL statement, DDL, object name, or data to be viewed contains Chinese characters, set the character encoding to **GBK** if it is supported by the OS.

Connection Management

On the **Advanced** tab of the **New Connection** and **Edit Connection** pages, commas (,) are used as separators in the **include** and **exclude** columns. Therefore, a schema name that contains a comma (,) is not supported.

Database Tables

- On the **Index** tab of the table creation wizard, the sequence of the selected columns in the list view cannot be retained after columns are deleted.
- When an operation has completed, and if the Data Studio window is not the active window of the operating system, then the message dialog is shown only when the Data Studio window becomes active.
- The following limitations are applicable to operations in **Editing Table Data**:
  - Entering expression values in the **Edit Table Data** tab is not supported.
  - Data Studio allows editing of fetched records only.
  - The Editing table filter feature will not highlight search words within HTML tags such as <, &, >.
  - A cell containing single '&' in it will not be displayed in the tooltip. A cell containing two consecutive '&' will display as a single '&' in the tooltip.
  - Row focus is not retained on a newly added row. You need to click the desired cell.

Function/Procedure

Functions/Procedures created in the SQL Terminal or the **Create Function/Procedure** wizard must end with "/" to indicate the end of functions/procedures. Statements entered after a function/procedure without "/" at the end will be treated as a single query and may trigger errors during execution.

General

- A maximum of 100 tabs can be opened in the editor area. Tabs are based on available resources of the host machine.
- A maximum of 64 characters (text only) are allowed for database object names (database, schema, function, procedure, table, sequence, constraint, index, view, and tablespace). There is no limit to the number of characters that can be used in expressions and descriptions in Data Studio.

- A maximum of 300 result tabs can be opened on a logged instance of Data Studio.
- If there are large objects loaded in Object Browser and Search Object window, expanding of objects in Object Browser may be slow and Data Studio may become unresponsive.
- Adjusting the cell width may cause Data Studio to fail to respond.
- When the data in a table cell is more than 1000 characters, it will be trimmed to 1000 characters with "..." at the end.
  - If the user copies the data from a cell in a table or the **Result** tab and pastes it on any editor (such as SQL terminal/PLSQL source editor, notepad or any other external editor application), the entire data is pasted.
  - If the user copies the data from a cell in a table or the **Result** tab and pastes it on an editable cell (same or different), the cell shows only the first 1000 characters with "..." in the end.
  - When the table/**Result** tab data is exported, the exported file contains the whole data.

Security

Data Studio validates SSL connection parameters only for the first time of connection. If **Enable SSL** is selected, the same SSL connection parameters are used when a new connection is opened.

NOTE

- If **Enable SSL** is not selected during connecting to Data Studio, data is not encrypted by default.
- If the security file is damaged during the SSL connection, Data Studio cannot perform any database operations. To resolve this problem, delete the security folder in the corresponding configuration folder and restart Data Studio.

SQL Terminal

- Opening an SQL file containing a large number of queries may result in an 'Insufficient Memory' error. For details, see **Troubleshooting**.
- Data Studio does not disable the auto-suggest and hyperlink features in the commented text in the **SQL Terminal**.
- The Hyperlink feature is not supported if the schema or table name have either spaces or dots (.) in them.
- Auto-suggest is not supported if the object name contains single or double quotes in it.
- DS supports basic formatting of simple SELECT statements only and may not work as expected for complex queries.

4.1.3 System Requirements

This section describes the minimum system requirements for using Data Studio.

Software Requirements

OS

The following table lists the OS requirements of Data Studio.

Table 4-3 Supported OSs and corresponding installation packages

Server	OS	Supported Version
General-purpose x86 servers	Windows	Windows 7 (64 bit)
		Windows 10 (64 bit)
		Windows 2012 (64 bit)
		Windows 2016 (64 bit)

Browser

The following table lists the browser requirement of Data Studio.

OS	Version
Windows	Internet Explorer 11 or later

Other software requirements

The following table lists the software requirement of Data Studio.

Table 4-4 Data Studio software requirement

Software	Specifications
Java	Open JDK 1.8 or later corresponding to the OS bit is recommended.
GNU libc	DDL can be displayed, imported, exported; and data operations can be performed only in libc 2.17 and later in GN.

Table 4-5 Supported database versions

Database	Version
GaussDB(DWS)	1.2.*x* 1.5.*x* 8.0.*x* 8.1.*x* 8.2.x

NOTE

The recommended minimum screen resolution is 1080 x 768. If the resolution is lower than this value, the page display will be abnormal.

4.1.4 Customizing Data Studio

In the navigation pane on the left, choose **Settings** > **Preferences**. You can customize Data Studio based on preferences.

Table 4-6 Functions of preferences

Preferences	Options	Description
General	Shortcut Keys	Modify or cancel the shortcut key. You cannot add shortcut keys. For details about the default shortcut keys, see **Customizing Data Studio**.
Editor	Syntax Coloring	You can modify the highlight colors of comments, default values, non-reserved keywords, reserved keywords, data types, symbols, constants, and character strings.
	SQL History	You can set the number of SQL history records and the number of SQL query characters. • Number of SQL history records Sets the number of queries that can be saved in historical records. After this parameter is modified, the modification takes effect for new queries. The value ranges from 1 to 1000. The default value is **50**. • Number of SQL query characters Sets the number of characters of the queries saved in the SQL historical records. After this parameter is modified, the modification takes effect for new queries. The value ranges from 1 to 1000. The default value is **1000**. If the characters are not limited, set the parameter to **0**.
	Template	Create, edit, or delete the abbreviation template.

Preferences	Options	Description
	Format	Sets the tab width and whether to convert tabs to spaces while performing indenting and un-indenting operation.
	Transaction	Whether the automatic commit function is enabled. • **Enable**: Transactions are automatically submitted. • **Disable**: You need to manually submit or roll back the transaction.
	Folding	Whether the SQL folding function is enabled. Modification only takes effect in a newly opened editor. The editor which is already opened will remain with previous settings until restart. • **Enable**: SQL statements can be folded or unfolded. • **Disable**: SQL statements cannot be folded or unfolded.
	Font	Setting the font. The value ranges from 1 to 50. The default value is **10**.
	Auto Suggest	Set the recommended minimum number of characters. This field indicates the minimum number of characters required to trigger automatic advice when a user enters an SQL statement. The value ranges from 2 to 10. The default value is **2**.

Preferences	Options	Description
Security	Password	• Whether the **Permanently** option is displayed in the **Save Password** drop-down list. – **Yes**: The **Permanently** option is displayed in the **Save Password** drop-down list in the connection window. – **No**: The **Permanently** option is displayed in the **Save Password** drop-down list in the connection window, and the saved password will be deleted. • Whether login is allowed after the password expires. – **Yes**: After the password expires, the user can still log in to Data Studio. – **No**: After the password expires, the user cannot log in to Data Studio.
	Security Disclaimer	Whether the security disclaimer is enabled. • **Enable**: The security disclaimer is displayed each time you try to establish an insecure connection or perform a file operation. • **Disable**: You need to agree to the security implications that may arise due to insecure connection.

Preferences	Options	Description
Environment	Session Setting	• Setting the Data Studio and file encoding. The default encoding type is UTF-8 • Whether the SQL assistant is enabled. The SQL assistant can provide suggestions or references for the content entered in the SQL terminal and stored procedure/ function terminal. • Setting the interval for automatically saving the data in the SQL terminal and stored procedure/ function terminal. The interval ranges from 2 to 60. The default value is **5**. • Setting whether automatically save the result by encrypting the saved data • Setting the display of imported table data and restrictions on imported file data

Preferences	Options	Description
Result Management	Query Results	• Setting the number of results to be obtained: Obtain all results or a specified number of records • Setting the width of a column – Content Length: setting the width of a column based on the content length of the column – Custom Length: setting the minimum width of a column based on the value length in network mode and the character length in text mode. The unit is pixel. • Setting advanced copy result: copying the selected content and column title or row number. • Whether the query result contains the result data code. If the result data code is contained, the result data code option is available when you edit and view the table, and query the results. • Setting whether the query result contains the result data text mode. If the result data text mode is contained, the result data text mode is available when you query results (up to 30,000,000 characters are supported). • Setting whether the query result window overwrites the result set. – Overwrite Resultset: After an opened result set window is closed, a new result set window will be opened. – Retain Current: Opens a new result set window while the already opened result set window is retained.
	Edit Table Data	Setting the table data saving mode. • Saving valid data. Invalid data is not saved and is highlighted for correction. • No data is saved when error occurs. Invalid data is highlighted for correction.

Preferences	Options	Description
Import/Export	Export	• Whether tablespace information is contained when DDL data is exported. • Whether the data export function is enabled. • Setting the export timeout period. The default value is 86400 seconds.

Table 4-7 Default shortcut keys of Data Studio

Function	Shortcut Key
Sorting the result sets of view tables, editing tables, and queries in ascending, descending, or server receiving order	Alt+Click
Opening the **Help** menu	Alt+H
Saving the SQL script	Ctrl+S
Opening the **Edit** menu	Alt+E
Compiling/Executing the SQL terminal statements	Ctrl+Enter
Find and Replace	Ctrl+F
Finding the previous one	Shift+F3
Finding the next one	F3
Redoing	Ctrl+Y
Copying the information of **Execution Time** and **Status** in the **Edit Table Data** tab	Ctrl+Shift+K
Copying the database object from the automatic recommendation list	Alt+U
Opening the **Callstack**, **Breakpoints**, or **Variables** pane	Alt+V
Opening the SQL script	Ctrl+O
Skipping a single step	F8
Stepping into	F7
Stepping out	Shift+F7

Function	Shortcut Key
Commenting out or canceling the comment line	Ctrl+/
Locating the first element in **Object Browser**	Alt+Page Up or Alt+Home
Locating the last element in **Object Browser**	Alt+Page Down or Alt+End
Locating to row	Ctrl+G
Disconnecting the connection	Ctrl+Shift+D
Formatting (SQL and PL/SQL)	Ctrl+Shift+F
Changing to uppercase	Ctrl+Shift+U
Changing to lowercase	Ctrl+Shift+L
Updating the cells or columns in the **Edit Table Data**, **Properties**, or **Results** pane Click the cell or column header to enable this option.	F2
Closing the **PL/SQL Viewer**, **View Table Data**, **Execute Query**, or **Properties** tab	Shift+F4
Continuing the PL/SQL debugging	F9
Cutting content	Ctrl+X
Copying the name of the object modified in **Object Browser** or in the terminal. You can copy the selected data from the **Terminal**, **Result**, **View Table Data**, or **Edit Table Data** page.	Ctrl+C
Copying the data on the **Result**, **View Table Data**, or **Edit Table Data** page (The data may or may not include the column titles and row numbers.)	Ctrl+Shift+C
Copying queries in the **Edit Table Data** tab	Ctrl+Alt+C
Copying content of the **Variables** tab	Alt+K
Copying the content of the **Callstack** tab	Alt+J
Copying the content of the **Breakpoints** tab	Alt+Y
Visualized interpretation plan	Alt+Ctrl+X

Function	Shortcut Key
Displaying online help (user manual)	F1
Template	Alt+Ctrl+Space
Switching to the first **SQL Terminal** tab	Alt+S
Selecting All	Ctrl+A
Opening the **Setting** menu	Alt+G
Refreshing the **Object Browser** pane	F5
Searching for objects	Ctrl+Shift+S
Opening the **Debugging** menu	Alt+D
Debugging a template	F10
Debugging a database object	Ctrl+D
Highlighting **Object Browser**	Alt+X
Opening the **File** menu	Alt+F
Creating a connection	Ctrl+N
Opening the **Run** menu	Alt+R
Switching between the **SQL Terminal** tabs	Ctrl+Page Up or Ctrl+Page Down
Expanding or collapsing all objects	Ctrl+M
Pasting content	Ctrl+V
Collapsing objects to browse the navigation tree	Alt+Q
Performing execution	Ctrl+E
Displaying the execution plan and expense	Ctrl+Shift+X
Stopping a running query	Shift+Esc
Commenting/Canceling the comment line or the entire segment	Ctrl+Shift+/
Opening the list of automatically recommended database objects	Ctrl+Space

4.2 Downloading and Installing the Data Studio Client

Step 1 GaussDB(DWS) provides a Windows-based Data Studio client and the tool depends on the JDK. You need to install the JDK on the client host first.

NOTICE

Only JDK 1.8 is supported.

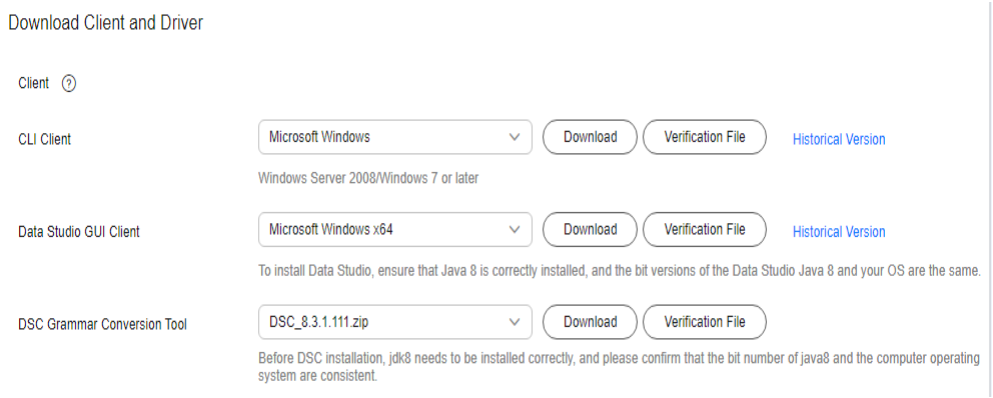
In the Windows operating system, you can download the required JDK version from the **official website of SDK**, and install it by following the installation guidance.

Step 2 Log in to the **GaussDB(DWS) console**. In the navigation pane, choose **Management** > **Client Connections**. The **Download Client and Driver** page is displayed.

Step 3 On the **Download Client and Driver** page, download **Data Studio GUI Client**.

- Select **Windows x86** or **Windows x64** based on the OS type and click **Download** to download a Data Studio version that matches the current cluster.
- Click **Historical Version** to download the corresponding Data Studio version. You are advised to download Data Studio based on the cluster version.

Figure 4-1 Downloading the DSC client



If you have clusters of different versions, select the gsql client that matches the version of your cluster. In the cluster list on the **Clusters** page, click the name of the specified cluster to go to the **Cluster Information** page and view the cluster version.

Step 4 Decompress the downloaded client software package (32-bit or 64-bit) to the installation directory.

Step 5 Open the installation directory and double-click **Data Studio.exe** to start the Data Studio client.

Figure 4-2 Starting the client

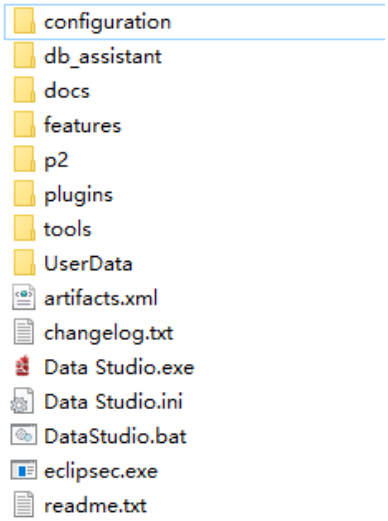


Table 4-8 Data Studio installation package structure

Folders/Files	Description
configuration	Contains information about the application startup and the required **Eclipse** plug-in path.
db_assistant	Contains files related to the SQL Assistant function.
docs	• Contains *Data Studio User Manual*. • Contains copyright notices, licenses, and written offer of the open source software used in Data Studio.
features	Contains Eclipse (rich client protocol-GUI) and Data Studio features.
p2	Contains files required for providing and managing Eclipse- and Equinox-based applications.
plugins	Contains the required Eclipse and Data Studio plug-ins.
tools	Contains tools that Data Studio depends on.

Folders/Files	Description
UserData/ • Autosave • *Logs/* • *Preferences/* • *Profile/* – *History/* • *Security/*	Contains folders of each OS user of Data Studio. **Autosave**: contains the information of queries and functions/procedures that are automatically saved. **Logs**: contains the **Data Studio.log** file that logs all the operations performed in Data Studio. **Preferences**: contains the **Preferences.prefs** file that stores the custom preferences. **Profile**: contains the **connection.properties** and **Profiles.txt** files, as well as SQL History, to manage connection information in Data Studio. **Security**: contains the files required for the management security of Data Studio. **NOTE** • The **UserData** folder is created after the first user opens the instance using Data Studio. • The **Logs** folder, language, memory settings and log level are common for all users. • After Data Studio is started, the **Logs**, **Preferences**, **Profile**, and **Security** folders, as well as the **Data Studio.log**, **Preferences.prefs**, **connection.properties**, and **Profiles.txt** files, are created. • If the **Logs** folder path is specified in the **Data Studio.ini** file, logs are created in the specified path. • When you cannot log in to Data Studio due to a damaged security key, perform the following steps to generate a new security key: 1. Delete the **Security** folder in **Data Studio** > **UserData**. 2. Restart Data Studio.
artifacts.xml	Contains the product build information.
changelog.txt	Contains the detailed log changes of the current release.
Data Studio.exe/DataStudio.sh	Allows you to connect to a server and perform operations, such as managing database objects, editing or executing PL/SQL programs.

Folders/Files	Description
Data Studio.ini	Contains the configuration information for running Data Studio.
readme.txt	Contains the features or fixed issues of the current release.

NOTE

If your computer blocks the running of the application, you can unlock the **Data Studio.exe** file to start the application.

Step 6 After the installation is complete, double-click the **StartDataStudio.bat** file in installation directory**/tools** to check the versions of the OS, Java, and Data Studio.

The batch file checks the version compatibility and will launch Data Studio or display appropriate message based on the OS, Java and Data Studio version installed.

If the Java version installed is earlier than 1.8, then an **error message** may be displayed.

The batch file determines the versions of the OS and Java for Data Studio based on the following scenarios:

DS Installation (32-bit/64-bit)	OS (Bit)	Java (bit)	Result
32	32	32	Data Studio is launched.
32	64	32	Data Studio is launched.
32	64	64	An **error message** will be displayed.
64	32	32	An **error message** will be displayed.
64	64	32	An **error message** will be displayed.
64	64	64	Data Studio is launched.

----End

4.3 Configuring Data Studio

This section describes the configuration steps to use Data Studio. It also explains the steps to configure servers for debugging PL/SQL Functions.

Configuring Data Studio

Steps to configure Data Studio using *Data Studio.ini* file:

NOTE

Restart Data Studio to view parameter changes. Invalid parameters added in the configuration file are ignored by Data Studio.

The following table lists the configuration parameters used in Data Studio.

Table 4-9 Configuration parameters

Parameter	Description	Value Range	Default Value
-startup	Defines the JAR files required to load Data Studio. This information varies based on the version used.	N/A	*plugins/org.eclipse.equinox.launcher_1.3.100.v20150511-1540.jar*
--launcher.library	Specifies the library required for loading Data Studio. The library varies depending on the Data Studio version.	N/A	**plugins/org.eclipse.equinox.launcher.win32.win32.x86_1.1.300.v20150602-1417** or **plugins/org.eclipse.equinox.launcher.win32.win32.x86_64_1.1.300.v20150602-1417** depending on the installation package used
-clearPersistedState	Removes any cached content on the GUI and reloads Data Studio.	N/A	N/A **NOTE** You are advised to add this parameter.
-consoleLineCount	Defines the maximum number of lines to be displayed in the **Messages** window.	1-5000	1000

Parameter	Description	Value Range	Default Value
-logfolder	Used to create the log folder. The user can specify the path to save logs. If the default value "." is used, then the folder is created in **Data Studio\UserData\<username>\logs**. For details, see **Setting the Location for Creating Log Files**.	N/A	-
-loginTimeout	Defines the connection wait time in seconds. Within the period specified by this parameter, Data Studio continuously attempts to connect to the database. If the connection times out, the system displays a message indicating that the connection times out or the connection fails.	N/A	180
-data	Defines the instance data location for the session.	N/A	@none
@user.home/MyAppWorkspace	Eclipse workspace is created in this location while Data Studio is being launched. **@user.home** refers to **C:/Users/<username>** Eclipse log files are available in **@user.home/MyAppWorkspace/.metadata**	N/A	N/A

Parameter	Description	Value Range	Default Value
-detailLogging	Defines the criteria with reference to logging error messages. Set to **True** to log all error messages. Set to **False** to log only error messages explicitly specified by Data Studio. Refer to **Controlling Exception and Error Logs** for more information. This parameter is not added by default and it can be set manually if logging is required.	True/False	False
-logginglevel	Creates the log files based on the value specified. If the value provided is arbitrary or empty, log files will be created using **WARN** value. For details, see **Different Types of Log Levels**. This parameter is not added by default and it can be set manually if logging is required.	FATAL, ERROR, WARN, INFO, DEBUG TRACE, ALL, and OFF	WARN
-focusOnFirstResult	Defines auto focus behavior for **Result** window. Set to **false** to automatically set focus to the last opened **Result** window. Set to **true** to disable the automatic set focus.	True/False	False
NOTE • All the above parameters must be added before **-vmargs**. • **-startup** and **--launcher.library** must be added as first and second parameter respectively.			

Parameter	Description	Value Range	Default Value
-vmargs	Specifies the start of virtual machine arguments. **NOTE** **-vmargs** must be the last parameter in the configuration file.	N/A	N/A
-vm <file name (javaw.exe) with relative path to Java executable>	Specifies the file name, for example, *javaw.exe*, and the relative path to Java.	N/A	N/A
-Dosgi.requiredJavaVersion	Defines the minimum java version required to run Data Studio. This value must not be modified.	N/A	1.5 **NOTE** **Note:** Recommended Java version is 1.8.0_141
-Xms	Defines the initial heap space that Data Studio consumes. This value must be in multiples of 1024 and greater than 40 MB and less than or equal to **-Xmx** size. Append the letter k or K to indicate kilobytes, m or M to indicate megabytes, g or G to indicate gigabytes. For example: -Xms40m -Xms120m Refer to Java documentation for more information.	N/A	-Xms40m

Parameter	Description	Value Range	Default Value
-Xmx	Defines the maximum heap space that Data Studio consumes. This value can be modified based on the available RAM space. Append the letter k or K to indicate kilobytes, m or M to indicate megabytes, g or G to indicate gigabytes. For example: -Xmx1200m -Xmx1000m Refer to Java documentation for more information.	N/A	-Xmx1200m
-OLTPVersionOldST	Used to configure the earlier OLTP versions. You can log in to gsql and run **SELECT VERSION()** to update the **OLTPVersionOldST** parameter in the .ini file using the obtained version number.	-	-
-OLTPVersionNewST	Used to configure the latest OLTP version. You can log in to gsql and run **SELECT VERSION()** to update the **OLTPVersionNewST** parameter in the .ini file using the obtained version number.	-	-

Parameter	Description	Value Range	Default Value
-testability	This parameter is used to enable testability features. For the current version after this function is enabled: • The user can copy content of last triggered auto-suggest operation using the Ctrl+Space shortcut key. • When **Include Analyze** is selected, **Execution Plan and Cost** is displayed in tree and graphical view. This parameter is available by default and needs to be added manually for testing.	True/False	False
-Duser.language	Defines the language settings for Data Studio. This parameter is added after the language setting is changed.	zh/en	N/A
-Duser.country	Specifies the country/region settings of Data Studio. This parameter is added after the language setting is changed.	CN/IN	N/A
-Dorg.osgi.framework.bundle.parent=ext	This parameter specifies which class loader is used for boot delegation.	boot/app/ext	boot
-Dosgi.framework.extensions=org.eclipse.fx.osgi	This parameter is used to specify a list of framework extension names. Framework extension bundles are fragments of the system bundle (org.eclipse.osgi). As a fragment, user can provide extra classes with the framework to use.	N/A	N/A

NOTE

- You are not allowed to modify the following settings:

  Dorg.osgi.framework.bundle.parent=ext

  Dosgi.framework.extensions=org.eclipse.fx.osgi
- If you receive the message **SocketException: Bad Address: Connect**:

  Check whether the client is connected to the server using the IPv6 or IPv4 protocol. You can also establish the connection by configuring the following parameters in the **.ini** file:

  -Djava.net.preferIPv4Stack=true

  -Djava.net.preferIPv6Stack=false

  Table 4-10 lists the supported communication scenarios.

  The first row and first column indicate the types of nodes that attempt to communicate with each other. **x** indicates that the nodes can communicate with each other.

Table 4-10 Communication scenarios

Node	V4 Only	V4/V6	V6 Only
V4 only	x	x	No communication possible
V4/V6	x	x	x
V6 only	No communication possible	x	x

Setting the Location for Creating Log Files

Step 1 Open the **Data Studio.ini** file.

Step 2 Provide the path for the **-logfolder** parameter.

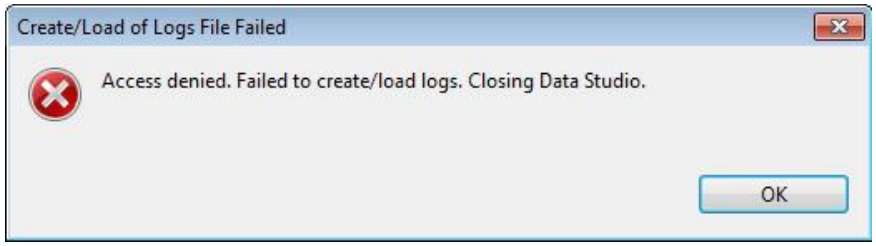
For example:

-logfolder=*c:|test1*

In this case, the **Data Studio.log** file is created in the **c:\test1\<user name>\logs** path.

NOTE

If any of the users does not have access to the path mentioned in the **Data Studio.ini** file, then Data Studio closes with the below pop-up message.



----End

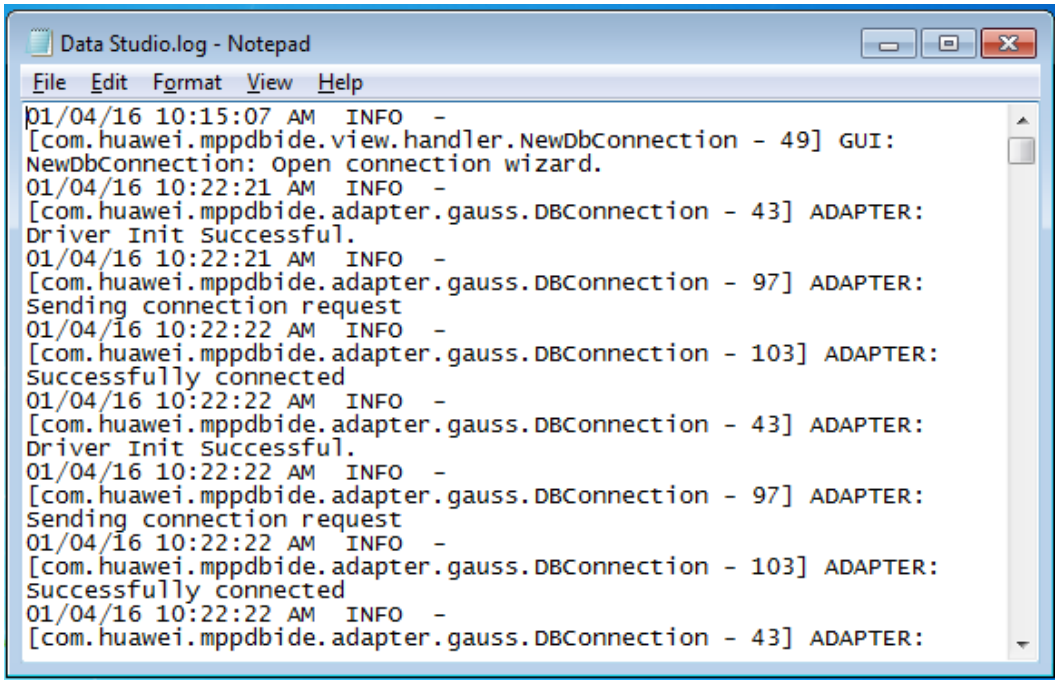
The **Data Studio.log** file will be created in the **Data Studio\UserData***Username*** \logs** path if:

- The path is not provided in the **Data Studio.ini** file.
  For example: **-logfolder=.**
- The path provided does not exist.

NOTE

Refer to the server manual for detailed information.

You can use any text editor to open and view the **Data Studio.log** file.



Controlling Exception and Error Logs

The stack running details of exception, error or throw-able are controlled based on the program argument parameter. This parameter is configured in the **Data Studio.ini** file.

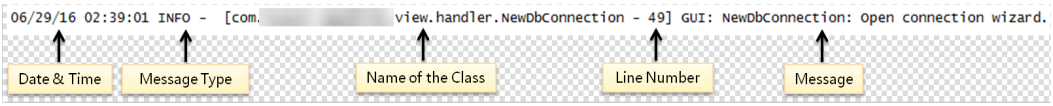
-detailLogging=false

If the value of **-detailLogging** is set to **True**, errors, exceptions, or stack running details of throwables will be logged.

If the value of **-detailLogging** is set to **False**, errors, exceptions, or stack running details of throwables will not be logged.

Description of the Log Message

The log message is described as follows:



When the size of the **Data Studio.log** file reaches 10,000 KB (the maximum value), the system automatically creates a file and saves it as **Data Studio.log.1**. Logs in **Data Studio.log** are stored in **Data Studio.log.1**. When the size of the **Data Studio.log** file reaches the maximum again, the system will automatically create a file and save it as **Data Studio.log.2**. Latest logs are always written in the **Data Studio.log** file. This process continues till **Data Studio.log.5** reaches the maximum file size and the cycle restarts. Data Studio deletes the earliest log file **Data Studio.log.1**. For example, the **Data Studio.log.5** renames to **Data Studio.log.4,** the **Data Studio.log.4** renames to **Data Studio.log.3** and so on.

NOTE

To enable performance logging in the **server log** file, the configuration parameter **log_min_messages** must be enabled and value must be set as **debug1** in the configuration file **data/postgresql.conf**, that is, **log_min_messages = debug1**.

Different Types of Log Levels

The different types of log levels that are displayed in the **Data Studio.log** file are as follows:

- **TRACE**: The TRACE level provides more detailed information than the DEBUG level.
- **DEBUG**: The DEBUG level indicates the granular information events that are most useful for debugging an application.
- **INFO**: The INFO level indicates the information messages that highlight the progress of the application.
- **WARN**: The WARN level indicates potentially harmful situations.
- **ERROR**: The ERROR level indicates error events.
- **FATAL**: The FATAL level indicates event(s) which cause the application to abort.
- **ALL**: The ALL level turns on all the log levels.
- **OFF**: The OFF level turns off all the log levels. This is opposite to ALL level.

  NOTE

  - If the user enters an invalid value to log level, then log level will be set to WARN.
  - If the user does not provide any log level, then log level will be set to WARN.

The logger outputs all messages greater than or equal to its log level.

The order of the standard log4j levels is as follows:

Table 4-11 Log levels

-	FATAL	ERROR	WARN	INFO	DEBUG	TRACE
OFF	x	x	x	x	x	x
FATAL	√	x	x	x	x	x
ERROR	√	√	x	x	x	x
WARN	√	√	√	x	x	x
INFO	√	√	√	√	x	x
DEBUG	√	√	√	√	√	x
TRACE	√	√	√	√	√	√
ALL	√	√	√	√	√	√
√- Creating a log file x - Not creating a log file						

4.4 Configuring SSL Connection

Data Studio can connect to the database using the Secure Sockets Layer [SSL] option. To use the SSL connection mode, you must configure related parameters on the client or in the application code. The GaussDB(DWS) console provides the SSL certificate required by the client. The SSL certificate contains the default certificate, private key, root certificate, and private key password encryption file required by the client.

Server Configuration

After a GaussDB(DWS) cluster is deployed, the SSL authentication mode is enabled by default. The server certificate, private key, and root certificate have been configured.

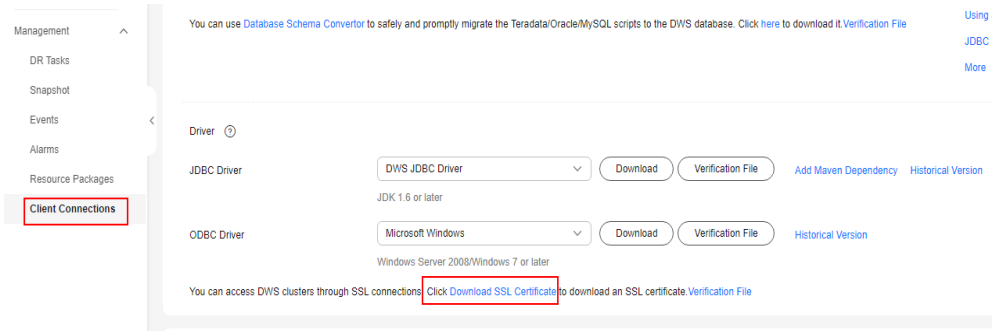
SSL Certificate and Client Configuration

You need to configure the client.

Step 1 You can download an SSL certificate from GaussDB(DWS).

Log in to the GaussDB(DWS) console. In the navigation pane, choose **Management** > **Client Connections**. In the **Driver** area, click **download an SSL certificate**.

Figure 4-3 Downloading SSL certificate



Step 2 Decompress the downloaded **dws_ssl_cert.zip** package to obtain the certificate file. Click the **SSL** tab on the Data Studio client and set the following parameters:

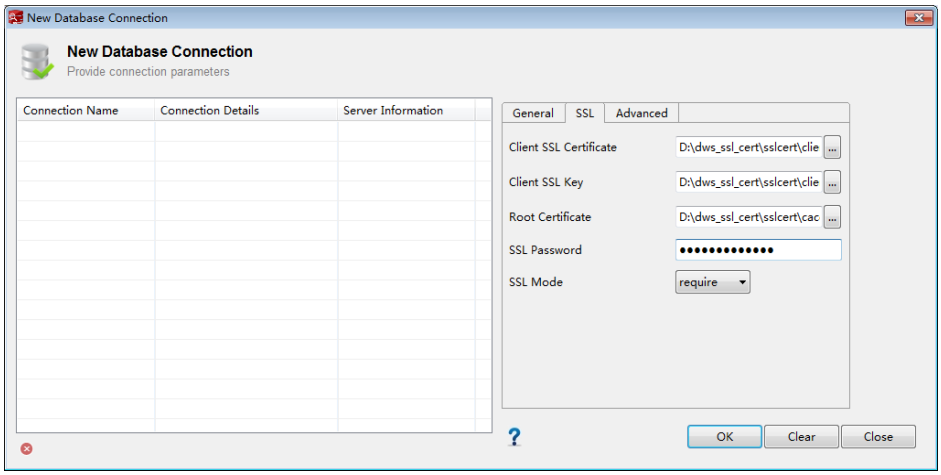
Table 4-12 Configuring SSL parameters

Parameter	Description
Client SSL Certificate	Select the **sslcert\client.crt** file in the decompressed SSL certificate directory.
Client SSL Key	Only the PK8 format is supported. Select the **sslcert\client.key.pk8** file in the directory where the SSL certificate is decompressed.
Root Certificate	When **SSL Mode** is set to **verify-ca** or **verify-full**, the root certificate must be configured. Select the **sslcert\cacert.pem** file in the decompressed SSL certificate directory.
SSL Password	Set the password for the client SSL key in PK8 format. The default password is **Gauss@MppDB**.
SSL Mode	Supported SSL modes include: • **require**: The SSL factory does not require verification, nor does it check the certificate validity. • **verify-ca**: The certificate authority (CA) will be verified using the corresponding SSL factory. • **verify-full**: The CA and database will be verified using the corresponding SSL factory. GaussDB(DWS) does not support the **verify-full** mode.

NOTE

- You can select a valid **Client SSL Certificate** and **Client SSL Key** to export DDL and data from Data Studio using secure connections.
- If the selected **Client SSL Certificate** and **Client SSL Key** are invalid, the export will fail. For details, see **Troubleshooting**.
- If you deselect **Enable SSL** and proceed, the **Connection Security Alert** dialog box is displayed. Refer to **Table 4-6** to determine whether to display this dialog box.
  - **Continue**: Continues to use insecure connections.
  - **Cancel**: Enables SSL.
  - **Do not show again**: If you select this option, the **Connection Security Alert** dialog box is not displayed for the subsequent connections of logged Data Studio instances.
- Data Studio prompts you to enter the client key upon the first access to the **gs_dump** feature.

Figure 4-4 SSL parameters



----End

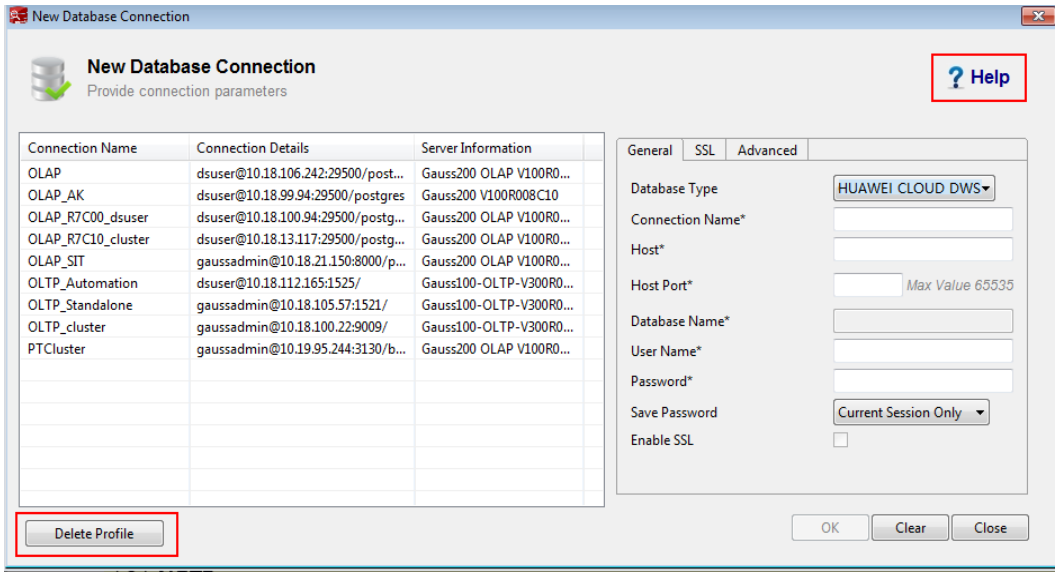
4.5 Connection Management

When Data Studio is started, the **New Database Connection** dialog box is displayed by default. To perform database operations, Data Studio must be connected to at least one database.

Enter the connection parameters to create a database connection between Data Studio and the database server. Hover the mouse cursor over the connection name to view the database information.

Adding a Connection

Step 1 Choose **File** > **New Connection** from the main menu, or click new connection icon on the toolbar. The **New Database Connection** dialog box is displayed.



Step 2 The table on the left lists the details of the existing connection profile(s) used to connect to the database along with the server information.

You can populate connection parameters, such as **Connection Name**, **Host**, and **Host Port**, by double-clicking the connection name.

NOTE

If the password or key for any of the existing connections is damaged, you need to enter the password for whichever connection you use.

Step 3 Configure the following parameters to create a database connection.

Table 4-13 Database connection information

Field	Description	Example
Database Type	Database type	-
Name	Connection name	My_Connection_DB

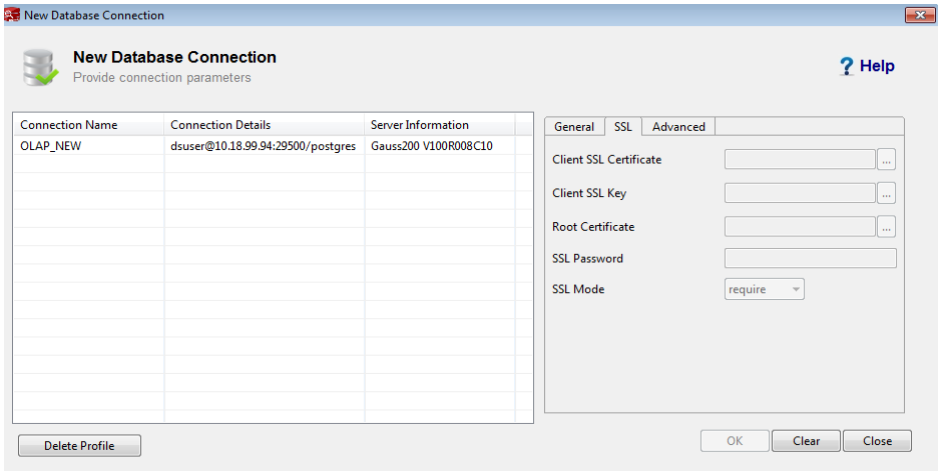
Field	Description	Example
Host	Specifies the host IP address (IPv4) or database domain name. **NOTE** • If domain name length is greater than 25 characters, the complete domain name will not be displayed. For example, *test1(db.dws… com:25xxx)* cannot be displayed completely. • Once the connection is created, hover the cursor over the connection name to see the server IP and version. • The value of this field will be identified as an IP address if it contains only digits with three periods (.). Otherwise, it will be identified as a domain name. • A domain name must meet the following requirements: – Start with a letter. – Contains only letters, digits, hyphens (-), and periods (.) and cannot contain any other special character. – Cannot contain spaces or tabs. – Contains 1 to 253 characters and a maximum of 63 characters between periods.	db.dws.mycloud.com 10.xx.xx.xx
Host Port	Provide the port address.	8000
Databases	Specifies the database name.	gaussdb
Username	Provide the username to connect to the selected database.	-
Password	Specifies the password for connecting to the selected database. The password is masked.	-
Save Password	• **Current Session Only**: The password is saved only for the current session. • **Do Not Save**: The password is not saved.	-

NOTE

- Click **Clear** to clear all fields in the **New Database Connection** dialog box.
- Press **Ctrl+V** to paste data in the **New Database Connection** dialog box. Right-click options are not available in the dialog boxes of Data Studio.

Step 4 Select the **Enable SSL** option and click the **SSL** tab. For details, see **Configuring SSL Connection**.

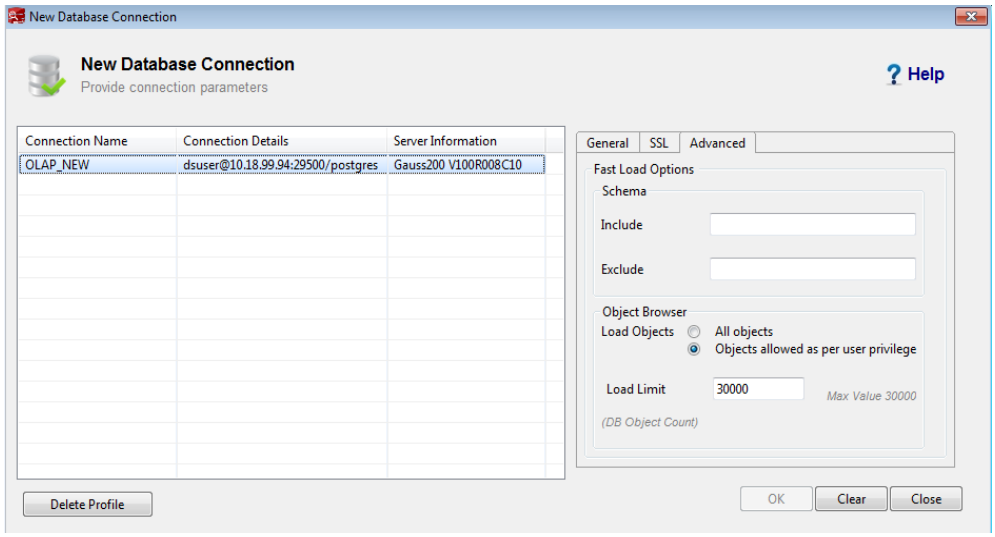
Figure 4-5 Configuring SSL parameters



Step 5 Perform the following steps to set **Fast Load Options**:

Click the **Advanced** tab.

Figure 4-6 Configuring fast load options



- Enter the schema names separated by commas in the **Include** field to load these schemas preferentially.
- Enter the schema names separated by commas in the **Exclude** field to avoid loading these schemas preferentially.
- Select either of the following options for **Load Objects**:

  - **All Objects**: Loads all objects.
  - **Objects allowed as per user privilege**: loads only objects that the user has permissions for accessing. For details about the minimum permissions for accessing objects listed in **Object Browser**, see **Table 4-2**.

    NOTE

    The default value is **Objects allowed as per user privilege**.

- Enter the number of database objects that can be loaded in **Load Limit**. The maximum number is 30,000.

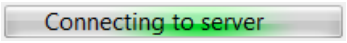
  NOTE

  - If the number of object types (such as tables and views) of the schema entered in **Include** is greater than the value of **Load Limit**, only the parent objects of the schema will be loaded. This indicates that child objects containing more than three parameters will not be loaded, such as columns, constraints, indexes, and functions.
  - Schema names provided in **Include** and **Exclude** are validated.
  - If you cannot access the schema specified in **Include**, an error message of the schema will be displayed during connection.
  - If you cannot access the schema specified in **Exclude**, the schema will not be loaded in **Object Browser** after the connection is created.

Step 6 Click **OK**.

The status bar displays the status of the completed operation.

When Data Studio is connecting to the database, the connection status is displayed as follows:



Once the connection is created, all schemas will be displayed in the **Object Browser** pane.

NOTE

- You can still log in to Data Studio even if the password has expired, but a message indicating that some operations may not be performed normally will be displayed. For details about how to change the configurations, see **Table 4-6**.
- PostgreSQL schema names are not displayed in the **Object Browser** pane.

----End

Renaming a Connection

Step 1 In the **Object Browser** pane, right-click the selected connection name and select **Rename Connection**.

A **Rename Connection** dialog box is displayed, prompting you to enter the new connection name.

Step 2 Enter the new connection name. Click **OK** to rename the connection.

The status bar displays the status of the completed operation.

NOTE

The new connection name must be unique. Otherwise, the rename operation will fail.

----End

Editing a Connection

Step 1 In the **Object Browser** pane, right-click the selected connection name and select **Edit Connection**. The **Edit Connection** dialog box is displayed.

To edit an active connection, you need to disable the connection and then open the connection with the new properties. A warning message about connection resetting is displayed.

Step 2 Edit the connection parameters. For details, see **Table 4-13**.

NOTE

The database type and name cannot be modified.

Step 3 Click **OK**.

NOTE

- You can click **Clear** to clear all fields in the **Edit Database Connection** dialog box.
- If you click **OK** without modifying any connection parameters, a dialog box is displayed, indicating that the modification is not saved. After the connection parameters are modified, a dialog box is displayed.

Step 4 If SSL is not enabled, a **Connection Security Alert** dialog box is displayed. Click **Continue** to proceed with insecure connections or click **Cancel** to return to the **Edit Connection** dialog box to enable SSL.

Do not show again option is used to hide the **Connection Security Alert** dialog box for subsequent connections.

A dialog box is displayed asking users to confirm whether the database whose connection has been edited is deleted.

Step 5 Click **Yes** to proceed to updating the connection information and reconnecting the connection with the updated parameters.

The status bar displays the status of the completed operation.

----End

Removing a Connection

Step 1 Right-click the selected connection name and select **Remove Connection**.

A confirmation dialog box is displayed.

Step 2 Click **Yes** to remove the server connection.

The status bar displays the status of the completed operation.

This operation will remove the connection from the **Object Browser**. Any unsaved data will be lost.

----End

Refreshing Connection Data

Step 1 In the **Object Browser** pane, right-click the selected connection name and select **Refresh** or press **F5** to update the connection with the latest content on the server.

The status bar displays the status of the completed operation.

----End

- The time taken to refresh a database depends on the number of objects in the database. Therefore, you are advised to perform this operation during off-peak hours in a large-scale database.
- If you refresh the entire database or connection, all child objects of schemas in **search_path**, as well as the schemas already expanded by the user, will be loaded again.
- If you reconnect to the database, only schema objects saved under **search_path** will be loaded. Objects that have been expanded will not be loaded.
- A database and multiple objects under it cannot be refreshed simultaneously.

Viewing Connection Properties

Step 1 Right-click the selected connection and select **Properties**.

The status bar displays the status of the completed operation.

Properties of the selected connection are displayed.

NOTE

If the property of a created connection is modified, then open the properties of the connection again to view the updated information.

----End

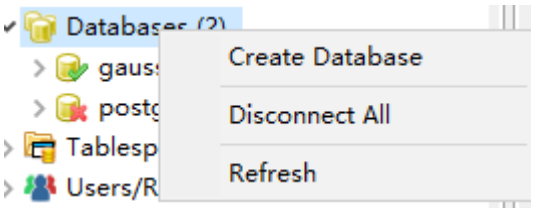
4.6 Database Management

4.6.1 Database Management

A relational database contains a set of tables that can be operated based on the relational model of data. A relational database contains a group of data objects for storing, managing, and accessing data objects, including tables, views, indexes, and functions.

Creating a Database

Step 1 In the **Object Browser** pane, right-click the **Databases** group and select **Create Database**.



Step 2 A **Create Database** dialog box is displayed, prompting you to provide the information required for creating a database.

- Enter a database name.
- Select the required type of encoding character set from the **Database Encoding** drop-down list. The database supports **UTF-8**, **GBK**, **SQL_ASCII**, and **LATIN1** encoding character sets. Creating the database with other encoding character sets may result in erroneous operations.

  NOTE

  This operation can be performed only when there is at least one active database.

Step 3 Select **Connect to the DB** and click **OK**.

The status bar displays the status of the completed operation.

You can view the created database in the **Object Browser**. The system-related schema in the server is automatically added to the new database.

----End

Connecting to the Database

In the **Object Browser** pane, right-click the database name and choose **Connect to DB**. The status bar displays the status of the completed operation.

NOTE

This operation can be performed on a disconnected database.

Renames a Database

Step 1 In the **Object Browser** pane, right-click the selected database and select **Rename Database**.

NOTE

This operation can be performed on a disconnected database.

The **Rename Database** dialog box is displayed, prompting you to provide the information required for renaming the database.

Step 2 Enter a new database name. Select the **Connect to the DB** check box and click **OK**.

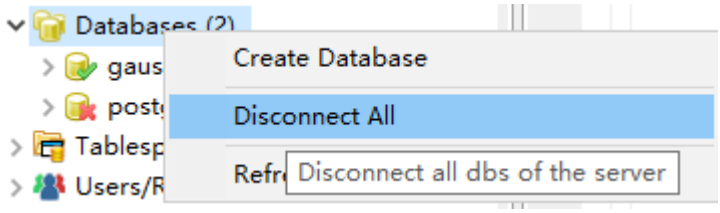
A confirmation dialog box is displayed, promoting you to confirm the renaming operation.

----End

Disconnecting from a Database

You can disconnect all databases or a specified database under a connection by using the disconnection function.

Step 1 In the **Object Browser** pane, right-click the selected the **Databases** group and select **Disconnect All**. This will disconnect all the databases under the connection.

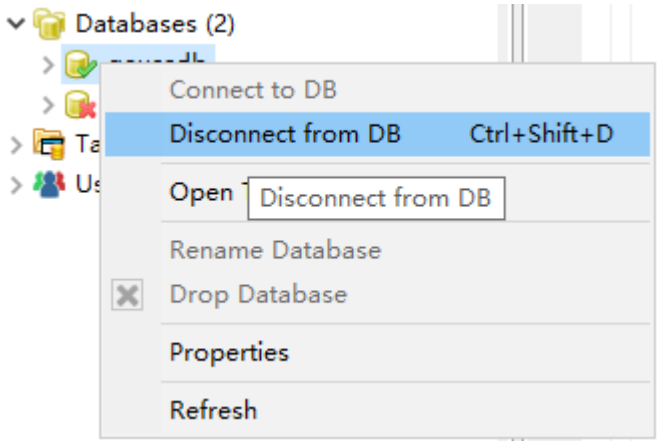


NOTE

This operation can be performed only when there is at least one active database.

A confirmation dialog box is displayed, prompting you to confirm the disconnection operation.

Step 2 In the **Object Browser** pane, right-click the selected database name and select **Disconnect from DB**.



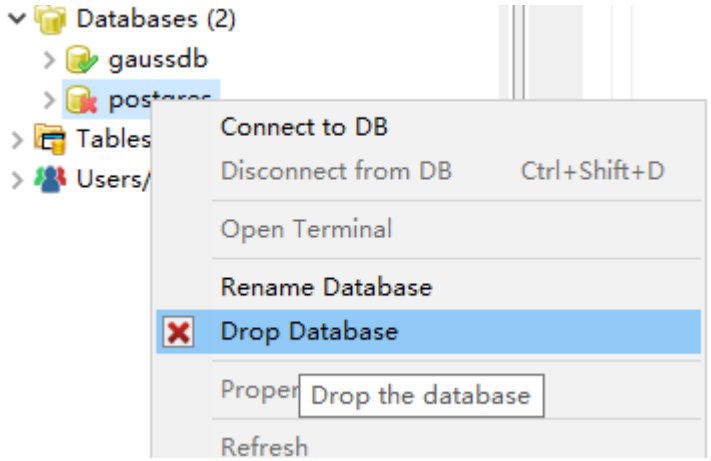
NOTE

This operation can be performed only on the primary database.

----End

Dropping a Database

In the **Object Browser** pane, right-click the selected database and select **Drop Database**. A confirmation dialog box is displayed, asking you whether to delete the database.

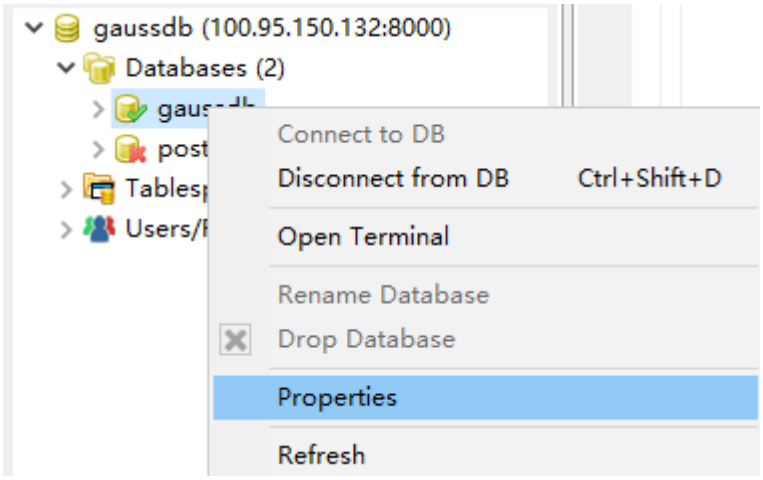


NOTE

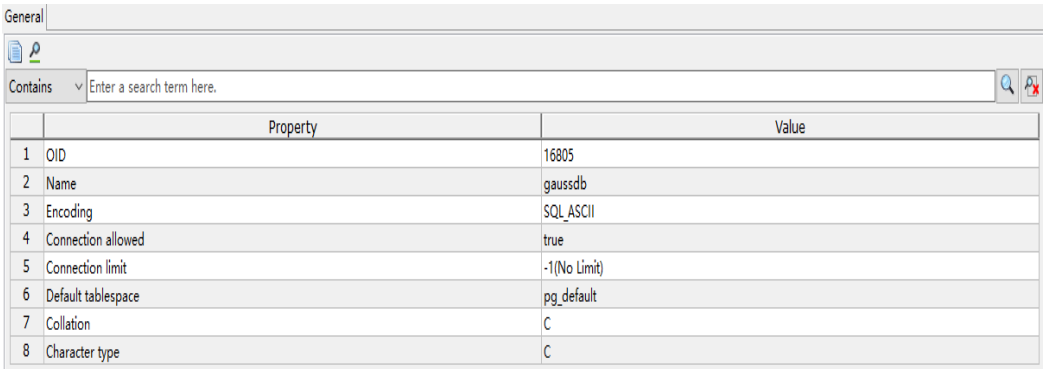
This operation can be performed on a disconnected database.

Viewing Data Properties

Right-click the selected database and select **Properties**. The status bar displays the status of the completed operation.



The properties of the selected database are displayed.



	Property	Value
1	OID	16805
2	Name	gaussdb
3	Encoding	SQL_ASCII
4	Connection allowed	true
5	Connection limit	-1(No Limit)
6	Default tablespace	pg_default
7	Collation	C
8	Character type	C

NOTE

- This operation can be performed only when there is at least one active database.
- If the property of an opened database is modified, then refresh and open the properties of the database again to view the updated information on the same opened window.

4.6.2 Schema Management

This section introduces how to use the database schema. All system schemas are grouped in **Catalogs**, and user schemas are grouped in **Schemas**.

Create Schema

Step 1 In the **Object Browser** pane, right-click the selected **Schemas** group and select **Create Schema**.

NOTE

Only refresh can be performed on **Catalogs** group.

Step 2 Enter a schema name and click **OK**. You can create the schema only if the database connection is active.

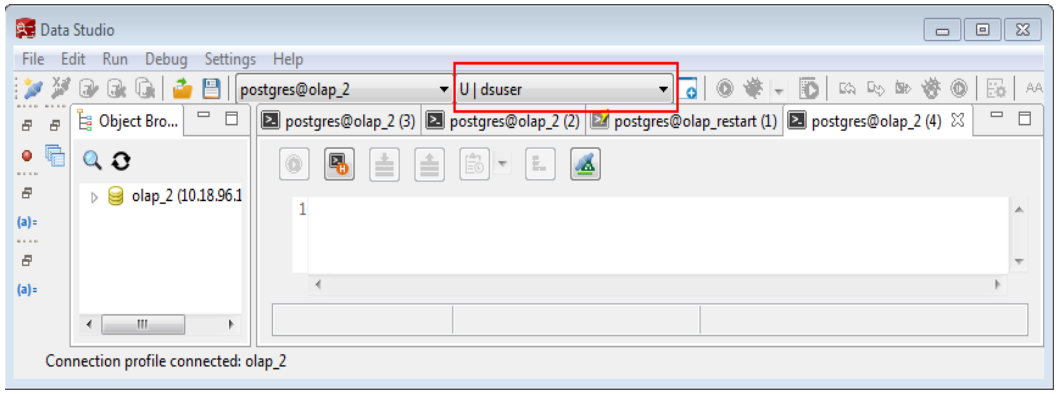
The status bar displays the status of the completed operation.

You can view the new schema in the **Object Browser** pane.

----End

NOTE

Data studio displays default schema of the user in the toolbar.



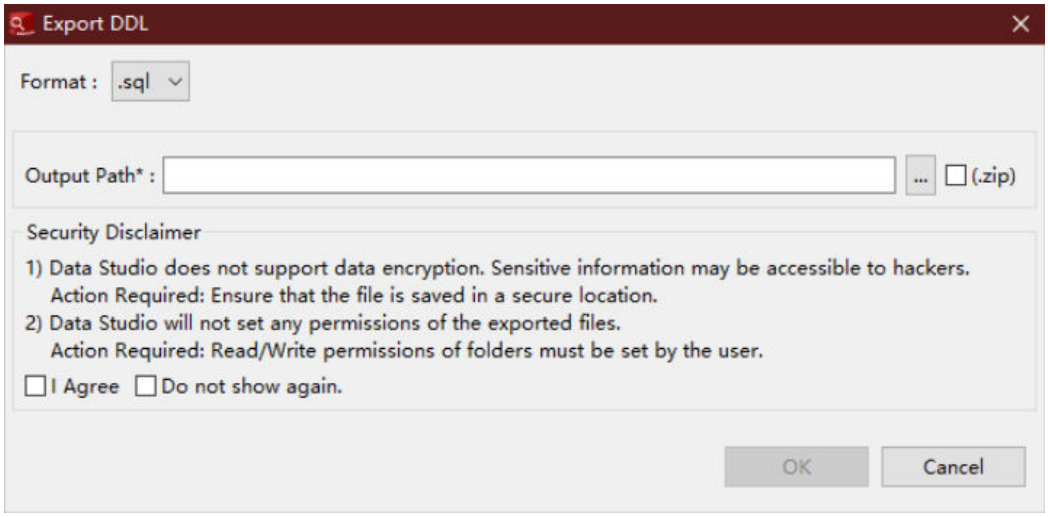
- When a CREATE query without mentioning the schema name is executed from SQL Terminal, the corresponding objects are created under the default schema of the user.
- When a SELECT query is executed in SQL terminal without mentioning the schema name, the objects are queried based on the default schema.
- When Data Studio starts, the default schemas are set to <username>. The public schemas have the same priority.
- If another schema is selected in the drop-down, the selected schema will be set as the default schema, overriding previous setting.
- The selected schema is set as the default schema for all active connections of the database (selected in database list drop-down).

Exporting Schema DDL and Data

You can right-click **Export DDL** to export the DDL of functions/procedures, tables, sequences, and views of the schema. To export data, choose **Export DDL and Data**.

Step 1 In the **Object Browser** pane, right-click the selected schema and select **Export DDL**.

You need to customize the export path. To compress data, select **.zip**.



You must select **I Agree** under the **Security Disclaimer**, then click **OK**. You can select **Do not show again**. After the disclaimer is disabled, it will not be displayed when you export DDL. For details, see **Table 4-6**.

Step 2 Click **OK**. The operation progress is displayed on the status bar in the lower right corner.

NOTE

- If the file name contains characters that are not supported by Windows, the file name will be different from the schema name.
- The Microsoft Visual C Runtime file (**msvcrt100.dll**) is required for this operation. For details, see **Troubleshooting**.
- You can select multiple objects and export their DDL. **Batch Export** lists the objects whose DDL cannot be exported.

The **Data Exported Successfully** dialog box and status bar display the status of the completed operation.

Table 4-14 The supported DDL encoding formats

Database Encoding	File Encoding	Support for Exporting a DDL
UTF-8	UTF-8	Yes
	GBK	Yes
	LATIN1	Yes

Database Encoding	File Encoding	Support for Exporting a DDL
GBK	GBK	Yes
	UTF-8	Yes
	LATIN1	No
LATIN1	LATIN1	Yes
	GBK	No
	UTF-8	Yes

----End

Renaming a Schema

Step 1 In the **Object Browser** pane, right-click the selected schema and select **Rename Schema**.

Step 2 Enter a schema name and click **OK**.

You can view the renamed schema in **Object Browser**.

The status bar displays the status of the completed operation.

----End

Granting/Revoking a Privilege

Step 1 Right-click the schema group and select **Grant/Revoke**.

The **Grant/Revoke** dialog box is displayed.

Step 2 Open the **Object Selection** tab to select the desired objects, and click **Next**.

Step 3 Select the role from the **Role** drop-down list in the **Privilege Selection** tab. Select the permissions to be granted or revoked.

Step 4 On the **SQL Preview** tab, you can check the automatically generated SQL query. If the result does not meet the expectation, return to the previous step until the result meets the expectation.

Step 5 Click **Finish**.

----End

Dropping a Schema

Step 1 In the **Object Browser** pane, right-click the selected schema and select **Drop Schema**. A confirmation dialog is displayed, prompting you to confirm the deletion.

Step 2 Click **OK**. This action will remove the schema from **Object Browser**.

A pop-up message and the status bar display the status of the completed operation.

----End

Refreshing a Schema

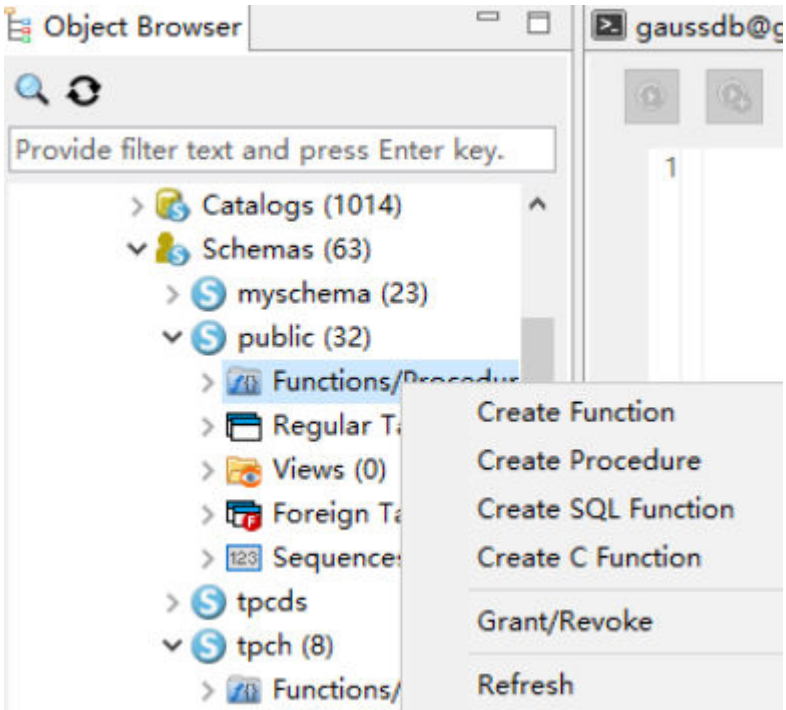
Right-click a schema name and choose **Refresh** to refresh all objects in the schema.

4.6.3 Function/Procedure Management

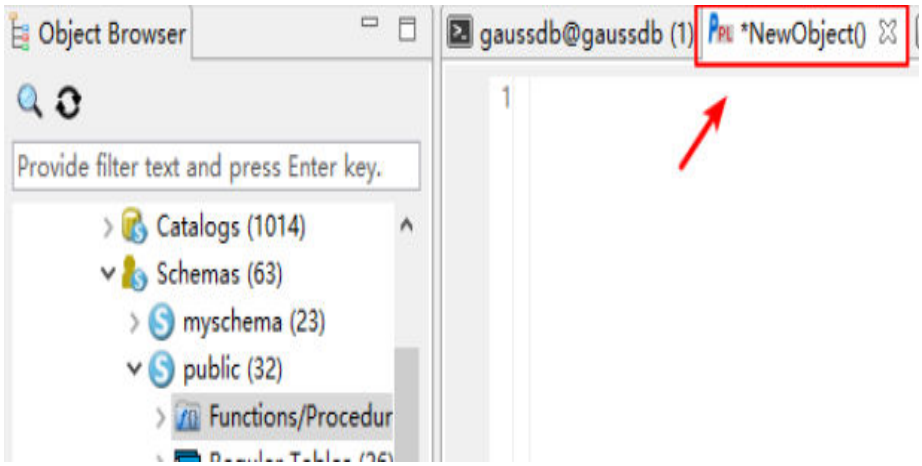
Creating a Function/Procedure

Step 1 In the **Object Browser** pane, right-click **Functions/Procedures** under the schema where you want to create the function/procedure. Then select **Create Function**, **Create SQL Function**, **Create Procedure**, or **Create C Function** as required.

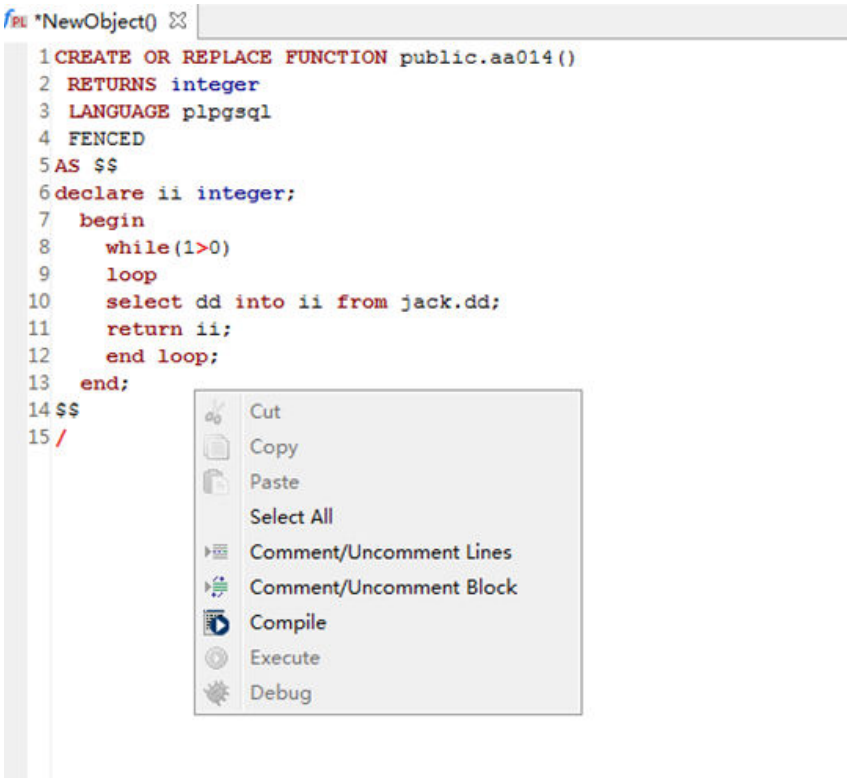
Right-click **Functions/Procedures** and a menu is displayed.



Step 2 The selected template is displayed in the new tab of Data Studio.



Step 3 After adding a function or procedure, you can choose **Run** > **Compile/Execute Statement** from the main menu or right-click the tab and select **Compile** to compile the function or procedure.



Step 4 The new function/procedure is displayed under **Object Browser**.

NOTE

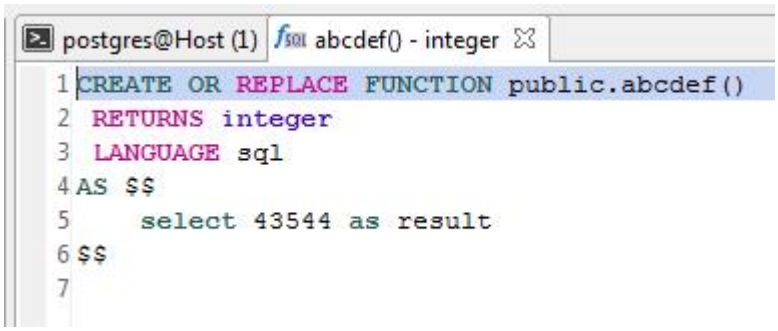
- The asterisk (*) next to the procedure name indicates that the procedure is not compiled or added to **Object Browser**.
- Refresh **Object Browser** by pressing **F5** to view the newly added debugging object.
- C functions do not support debugging operations.
- A pop-up message displays the status of the completed operation, which is not displayed in the status bar.

----End

Editing a Function/Procedure

Step 1 In the **Object Browser** pane, double-click or right-click the required function/procedure or SQL function and select **View Source**.

The selected function/procedure or SQL function is displayed in the **PL/SQL Viewer** tab page.



NOTE

- You must refresh **Object Browser** to view the latest DDL.
- If multiple functions/procedures or SQL functions have the same schema, name, and input parameters, only one of them can be opened at a time.

Step 2 After editing or updating, compile and execute the PL/SQL program or SQL function.

If you execute the function/procedure or SQL function before compilation, the **Source Code Change** dialog box is displayed.

Step 3 Click **Yes** to compile and execute the PL/SQL function/procedure. The status of the completed operation is displayed in the **Message** tab page.

Step 4 After compiling the function/procedure or SQL function, press **F5** to refresh **Object Browser** to view the updated code.

----End

Granting or Revoking a Permission

Step 1 Right-click **Function/Procedure Group** and select **Grant/Revoke**. The **Grant/Revoke** dialog box is displayed.

Step 2 Open the **Object Selection** tab to select the desired objects, and click **Next**.

Step 3 The **Privilege Selection** tab is displayed. Select the role from the **Role** drop-down list.

Step 4 On the **SQL Preview** tab, you can check the automatically generated SQL query. If the result does not meet the expectation, return to the previous step until the result meets the expectation.

Step 5 Click **Finish**.

----End

Debugging a Function/Procedure

A breakpoint is used to stop a PL/SQL program on the line where the breakpoint is set. You can use breakpoints to control the execution and debug the procedure. When a line with a breakpoint set is reached, the PL/SQL program on this line will be stopped, and you can perform other debugging operations.

- Setting or Adding a Breakpoint on a Line

  Data Studio allows you to set or add breakpoints on a line.

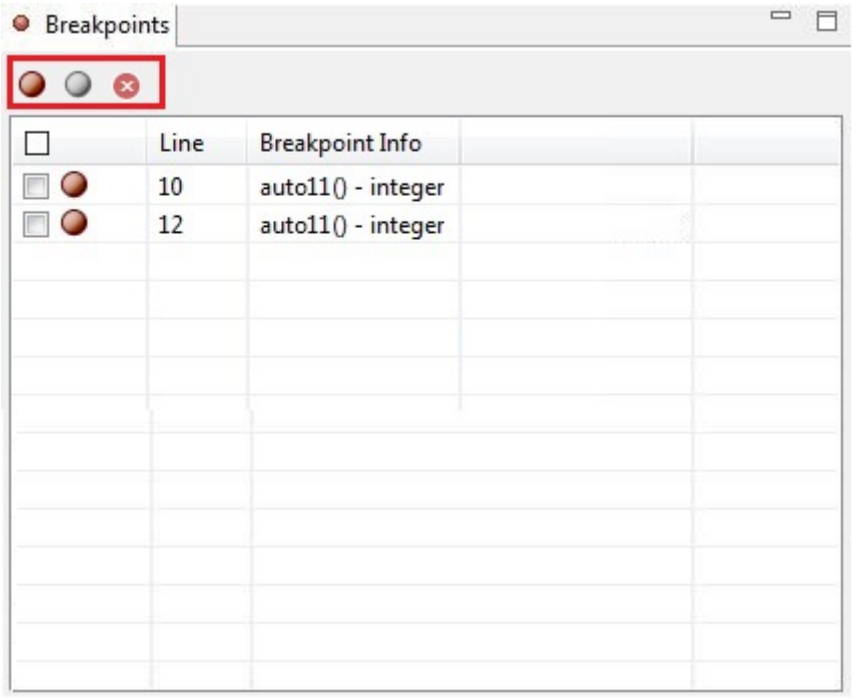
  Open the function or procedure for which you want to add a breakpoint, double-click the breakpoint ruler on the left of the line number field, and set a breakpoint. If the breakpoint is enabled, the operation is successful. If the function is not interrupted or stopped during debugging, the breakpoint set for the function is not validated.

- Using the Breakpoints Pane

  You can view and manage all breakpoints in the **Breakpoints** pane. Click the breakpoint option at the minimized pane to open the **Breakpoints** pane.

  You can enable, disable or remove a specific breakpoint by selecting the breakpoint check box and clicking the enabling, disabling, and removing icon in the **Breakpoints** pane. You can also double-click the breakpoint ruler on the left of the line number field to set a breakpoint or double-click the breakpoint icon to delete the breakpoint.

  In the **PL/SQL Viewer** pane, double-click the required breakpoint in the **Breakpoint Info** column to locate the breakpoint.



NOTE

- The **Breakpoints** pane lists each breakpoint with the row number and the debug object name.
- When a breakpoint is disabled, the program will not be stopped at the breakpoint. The breakpoint will be retained and can be enabled later.
- A removed breakpoint cannot be restored.
- Press **Alt+Y** to copy the content of the **Breakpoints** pane.

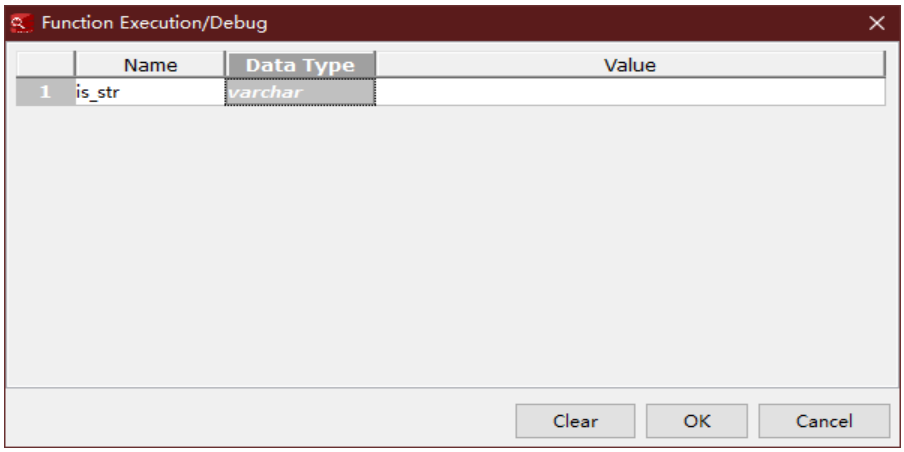
- Changing Source Code

  During debugging, if the source code is changed after it is fetched from the server and the debugging is continued, Data Studio displays an error. You are advised to refresh the object and perform the debug operation again.

  If you change the source code obtained from the server and execute or debug the source code without setting a breakpoint, the result of the source code obtained from the server will be displayed on Data Studio. You are advised to refresh the source code before executing or debugging it.

- Using Breakpoints to Debug Functions/Procedures

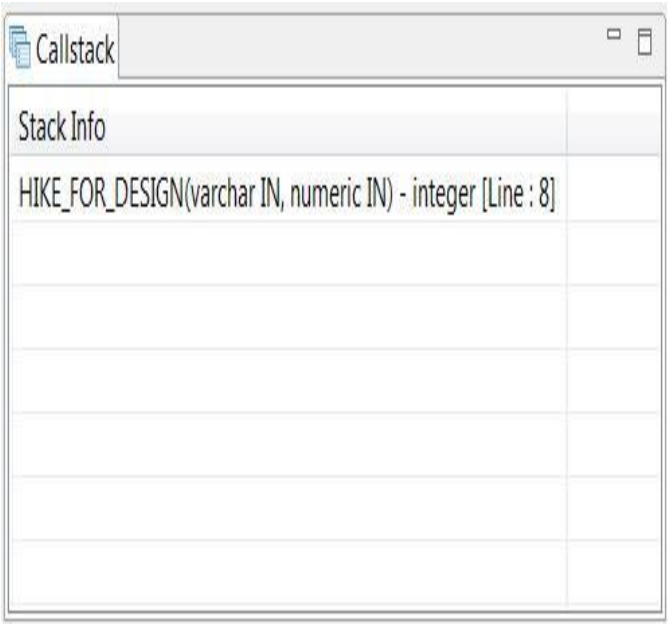
  After adding a breakpoint in the row you want to debug, click the debug icon or right-click the selected function in **Object Browser** and select **Debug**. In the **Debug Function/Procedure** dialog box, enter the parameter information.



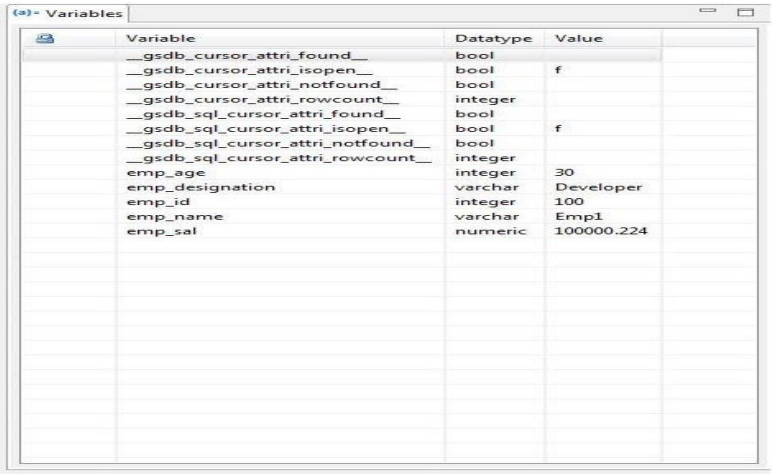
NOTE

- If no parameter is entered, the **Debug Function/Procedure** dialog box will not be displayed.
- Single quotation marks (') are mandatory for the parameters of **varchar** and **date** data types, but not mandatory for the parameters of **numeric** data type. To set the parameter to **NULL**, enter **NULL** or **null**.
- When a function or procedure is debugged or executed, the same parameter values are used for the next debugging or execution. The **Value** column is empty upon the first execution. Enter the values as required. Click **OK**. The parameter values will be cached. The cached parameter values will be displayed in the next execution or debugging. Once a specific connection is removed, all the cached parameter values are cleared.

– The **Callstack** pane is populated.



– The **Variables** pane shows the current value of variables. If you hover over the variable of a function/procedure, the current value is also displayed. System variables are displayed by default in the **Variables** pane. You can disable the display of system variables if necessary. The button is toggled on by default.



Setting/Displaying Variables	Description
Setting a variable to **NULL**	1. Double-click a variable value in the **Variables** pane. A dialog box is displayed. 2. Set the variable to an empty value.
Setting a string value	Some examples are as follows: ● To set the value to **abc**, enter **abc**. ● To set the value to **Master's Degree**, enter **Master''s Degree**. ● To set the value as text (**NULL**), set it to **NULL** in the **Variables** pane.
Setting a **BOOLEAN** value	Enclose the **BOOLEAN** value **t** or **f** within single quotation marks ('). For example, to set **t** to a **BOOLEAN** variable, enter **'t'** in the **Value** column.
Displaying a variable value	If the variable value is **NULL**, it will be displayed as **NULL**. If the variable value is **NULL**, it will be displayed as empty. If the variable value is set to a string, for example, **abc**, it will be displayed as **abc**.

  - During function/procedure debugging, right-click the variable to be added in the editor and choose **Add Variable To Monitor** in the menu displayed. If the variable is monitored, its value in the **Monitor** pane will always be the same as that in the **Variables** pane.
  - In Data Studio, variable information is displayed if the cursor is hovered over that variable during the debugging of PL/SQL functions.

- Terminating Debugging

Click the terminating debugging button on the toolbar or choose **Debug** > **Terminate Debugging**. After the debugging is complete, the function execution proceeds and will not be stopped at any breakpoint.

After the debugging is complete, the result tab page displays the function execution result, and the **Callstack** and **Variables** panes are cleared.

- Data Studio provides the option to commit/rollback the query execution result after debugging is finished. Right-click the **SQL Terminal** window where the function is executed. Select **Debug With Rollback** to enable the rollback function after the debugging is complete.
  - If **Debug With Rollback** option is enabled, the function execution result after debugging is not saved to the database.
  - If **Debug With Rollback** option is disabled, the function execution result after debugging is submitted to the database.

Controlling Execution

- Single Stepping a PL/SQL Function

  You can run the command for single stepping in the toolbar to debug a function. This allows you to debug the program line by line. When a breakpoint occurs during the operation of single stepping, the operation will be suspended and the program will be stopped.

  Single stepping means executing one statement at a time. Once a statement is executed, you can see the execution result in other debugging tabs.

  NOTE

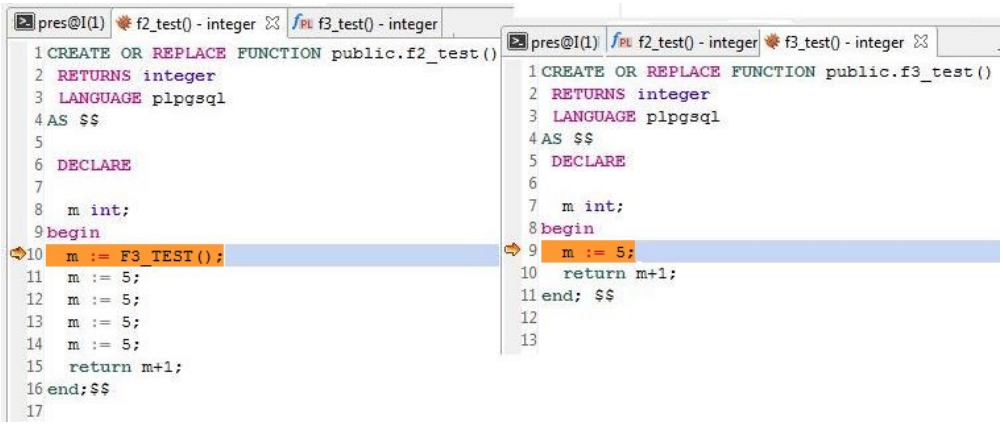
  A maximum of 100 **PL/SQL Viewer** tabs can be displayed at a time. If more than 100 tabs are opened, the tabs of function calls will be closed. For example, if 100 tabs are opened and a new debugging object is called, Data Studio will close the tabs of function calls and open the tab of the new debugging object.

- Step into

  To step through code one statement at a time, select **Step Into** from the **Debug** menu.

  When stepping into a function, Data Studio executes the current statement and then enters the break mode. The debug position will be indicated by an arrow on the left ruler pane. If the execution statement calls another function, Data Studio will step into that function. Once you have stepped through all the statements in that function, Data Studio jumps to the next statement of the function it was called from.

  Press **F7** to go to the next statement. If you click **Continue**, PL/SQL code execution will continue.



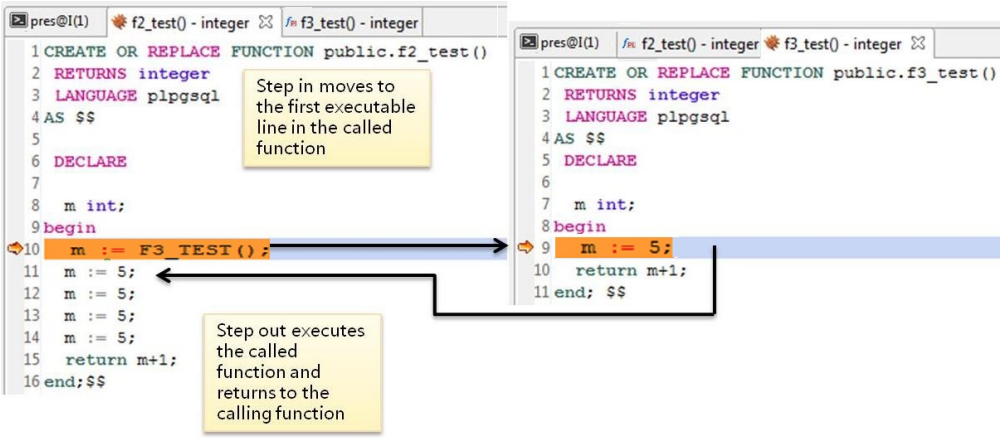
- Step Over

  Step Over is the same as **Step Into**. Unless another function is called, it will not step into the function. The function will run, and the next statement in the current function is executed. **F8** is the shortcut key for **Step Over**. However, if there is a breakpoint set inside the called function, **Step Over** will enter the function, and hit the set breakpoint.

- Step Out

  Stepping out of a sub-program continues the execution of the function and then suspends the execution after the function returns to its function call. You can step out this function when the rest of the function does not need to debug. However, if a breakpoint is set in the remaining part of the function, then that breakpoint will be hit before returning to the function call.

  A function will be executed when you step over or step out of it. **Shift+F7** is the shortcut key for **Step Out**.



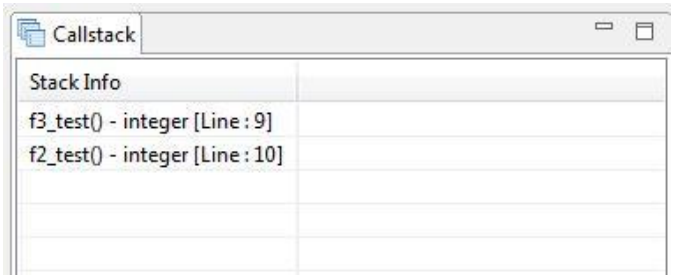
- Continuing the Debugging

  When the debugging process stops at a specific location, you can select **Continue** (or press **F9**) from the **Debug** menu to continue the PL/SQL function execution.

- Viewing Callstack

  The **Callstack** pane displays the chain of functions as they are called. The **Callstack** pane can be opened from the minimized window. The most recent functions are listed at the top, and the least recent at the bottom. At the end of each function name is the current line number in that function.

You can double-click the function names in the **Callstack** pane to open panes of different functions.

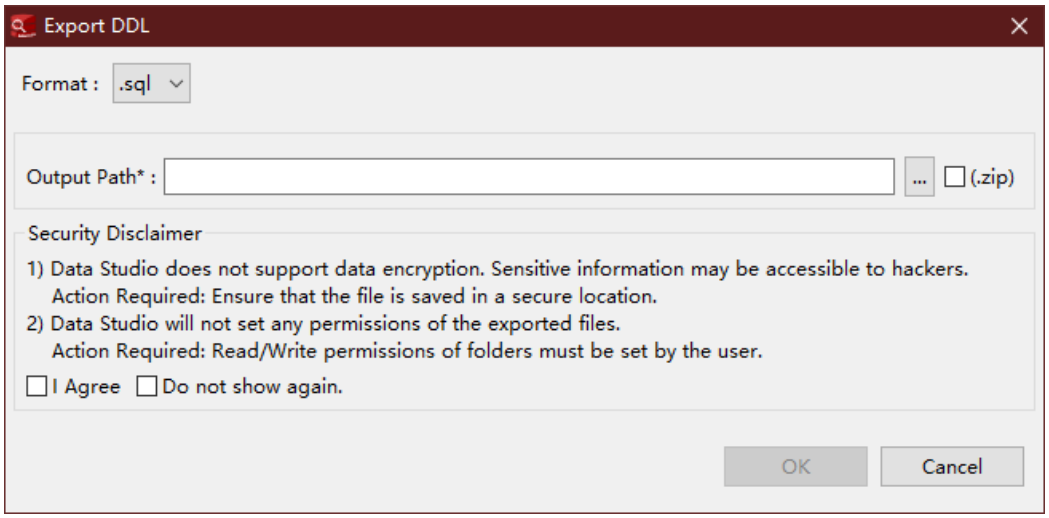


Exporting a Function or Procedure DDL

Follow the steps below to export the function/procedure DDL:

Step 1 In the **Object Browser** pane, right-click the selected function or procedure and select **Export DDL**.

You need to customize the export path. To compress data, select **.zip**.



You must select **I agree** under **Security Disclaimer**, then click **OK**. You can disable the security disclaimer. After the disclaimer is disabled, it will not be displayed when you export the DDL. For details, see **Table 4-6**.

Step 2 Click **OK**. The operation progress is displayed on the status bar in the lower right corner.

NOTE

- If the file name contains characters that are not supported by Windows, the file name will be different from the schema name.
- The Microsoft Visual C Runtime file (**msvcrt100.dll**) is required for this operation. For details, see **Troubleshooting**.

The **Data Exported Successfully** dialog box and status bar display the status of the completed operation.

Table 4-15 Supported DDL encoding formats

Database Encoding	File Encoding	Support for Exporting a DDL
UTF-8	UTF-8	Yes
	GBK	Yes
	LATIN1	Yes
GBK	GBK	Yes
	UTF-8	Yes
	LATIN1	No
LATIN1	LATIN1	Yes
	GBK	No
	UTF-8	Yes

----End

Deleting a Function/Procedure

You can delete functions or programs one by one or in batches.

Step 1 In the **Object Browser** pane, right-click the selected function/procedure object and select **Drop Object**.

Step 2 Select one or more function or procedure objects and choose **Delete Object**.

Step 3 In the dialog box that is displayed, click **Yes**.

The status bar displays the status of the completed operation.

----End

4.6.4 Table Management

4.6.4.1 Creating a Regular Table

4.6.4.1.1 Defining a Regular Table

A table is a logical structure maintained by a database administrator and consists of rows and columns. You can define a table as a part of your data definitions from the data perspective. Before defining a table, you need to define a database and a schema. This section describes how to use Data Studio to create a table. To define a table in the database, perform the following steps:

Step 1 In the **Object Browser** pane, right-click **Regular Tables**, and choose **Create Regular Table**.

Step 2 Define basic table information, such as the table name and table type. For details, see **Providing Basic Information**.

Step 3 Define column information, such as the column name, data type schema, data type, and column constraint. For details, see **Defining a Column**.

Step 4 For details about how to determine table data distribution settings, see **Selecting Data Distribution**.

Step 5 Define column constraints for different constraint types. Constraint types include **PRIMARY KEY**, **UNIQUE**, and **CHECK**. For details, see **Defining Table Constraints**.

Step 6 Define table index information, such as the index name and access mode. For details, see **Defining an Index**.

On the **SQL Preview** tab, you can check the automatically generated SQL query. For details, see **SQL Preview**.

----End

Providing Basic Information

If you create a table in a schema, the current schema will be used as the schema of the table. Perform the following steps:

Step 1 Enter a table name. It specifies the name of the table to be created.

NOTE

Select the **Case** check box to retain the capitalization of the value of the **Table Name** parameter. For example, if you enter the table name **Employee**, the table name will be created as **Employee**.

The name of the table schema is displayed in **Schema**.

Step 2 Select a table storage mode from the **Table Orientation** drop-down list.

- ROW: Create a row-store table.
- COLUMN: Create a column-store table.

Step 3 Select a table type. Its value can be:

- **Normal**: a normal table
- **Unlogged**: An unlogged table. When data is written to an unlogged table, the data is not recorded in logs. The speed of writing data to an unlogged table is much higher than that of writing data to a common table. However, an unlogged table is insecure. In the case of a conflict or abnormal shutdown, an unlogged table is automatically truncated. The content of unlogged tables is not backed up to the standby node, and no log is automatically recorded when an index is created for unlogged tables.

Step 4 Configure **Options**.

- **IF NOT EXISTS**: Create the table only if a table with same name does not exist.
- **WITH OIDS**: Create a table and assign OIDs.
- Configure **Filler Factor**. The value range is 10 to 100. The default value is 100 (filled to capacity).

If **Fill Factor** is set to a smaller value, the INSERT operation fills only the specified percentage of a table page. The free space of the page will be used to update rows on the page. In this way, the UPDATE operation can place the updated row content on the original page, which is more efficient than placing the update on a different page. Set it to 100 for a table that has never been updated. Set it to a smaller value for largely updated tables. TOAST tables do not support this parameter.

Step 5 Enter table description in the **Description of Table** text box.

Step 6 Click **Next** to define the column information of the table.

----End

You can configure the following parameters of a common table.

Table 4-16 Parameters

Parameter	Row-store Table	Column-store Table	ORC Table
Table Type	✓	✓	✗
If Not Exists	✓	✓	✓
With OIDS	✓	✗	✗
Fill Factor	✓	✗	✗

Defining a Column

A column defines a unit of information within a table's row. Each row is an entry in the table. Each column is a category of information that applies to all rows. When you add a table to a database, you can define the columns that compose it. Columns determine the type of data that the table can hold. After providing the general information about the table, click the **Columns** tab to define the list of table columns. Each column contains name, data type, and other optional properties.

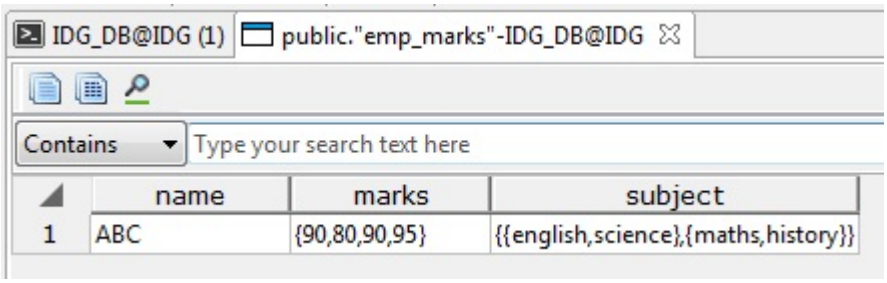
Step 1 Enter the column name in **Column Name** field. It specifies the name of a column to be created in the new table. This must be a unique name in the table.

NOTE

Select the **Case** check box to retain the capitalization of the value of the **Column Name** parameter. For example, if the column name entered is "Name", then the column name is created as "Name".

Step 2 Configure **Array Dimensions**. It specifies the array dimensions for the column.

Example: If array dimension for a column is defined as integer [], then it will add the column data as single dimension array.



The **marks** column in the above table was created as single dimension and subject column as two dimensions.

Step 3 Select the data type of the column from the **Data Type** drop-down list. For example, **bigint** for integer values.

For complex data types,

- Select the required schema from the **Data type Schema** drop-down list.
- Select the corresponding data type from the **Data Type** drop-down list. This list displays the tables and views for the selected schema.

  NOTE

  User-defined data types are not available for selection.

Step 4 Enter the precision/size value of the data type entered in the **Precision/Size** field. This parameter is valid only when the data type can be defined by precision or size.

Step 5 Select the scale of the data type entered in the **Scale** field.

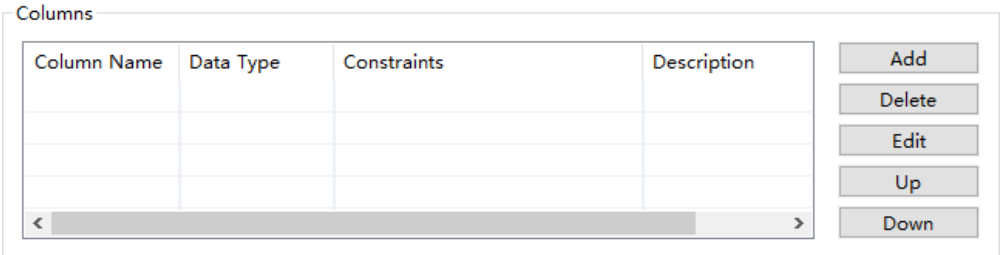
Step 6 Choose the following **Column Constraints** if required:

- **NOT NULL**: The column cannot contain null values.
- **UNIQUE**: The column may contain only unique values.
- **DEFAULT**: The default value used when no value is defined for the column.
- **Check**: An expression producing a Boolean result, which new or updated rows must satisfy for an INSERT or UPDATE operation to succeed.

Step 7 To add comments to **Column** in the **Create Regular Table** dialog box, add column information in **Description of Column (Max 5000 chars)** text box and click **Add**. You can also add comments in the column addition dialog box. You can check comments in general table properties.

Step 8 After you enter all information for new column, click **Add**. You can also delete a column from a list or change the order of columns. After defining all columns, click **Next**.

Step 9 You can add, delete, and edit columns, and adjust the sequence of columns.



----End

You can configure the following parameters of a column in a common table:

Table 4-17 Parameters

Parameter	Row-store Table	Column-store Table	ORC Table
Array Dimensions	√	x	x
Data Type Schema	√	x	x
NOT NULL	√	√	√
Default	√	√	√
UNIQUE	√	x	x
CHECK	√	x	x

Selecting Data Distribution

Data distribution specifies how the table is distributed or replicated among data nodes.

Select one of the following options for the distribution type:

Distribution Type	Description
DEFAULT DISTRIBUTION	The default distribution type will be assigned for this table.
REPLICATION	Each row of the table will be replicated in all the data nodes of the database cluster.
HASH	Each row of the table will be placed based on the hash value of the specified column.
RANGE	Each row of the table will be placed based on the range value.
LIST	Each row of the table will be placed based on the list value.

After selecting data distribution, click **Next**.

The following table lists the data distribution parameters that can be configured for common tables.

Table 4-18 Distribution types

Distribution Type	Row-store Table	Column-store Table	ORC Table
DEFAULT DISTRIBUTION	√	√	x
HASH	√	√	√
REPLICATION	√	√	x

Defining Table Constraints

Creating constraints is optional. A table can have one (and only one) primary key. Creating the primary key is a good practice.

The following table lists the table constraint parameters that can be configured for common tables.

Table 4-19 Constraint types

Constraint Type	Row-store Table	Column-store Table	ORC Table
CHECK	√	x	x
UNIQUE	√	x	x
PRIMARY KEY	√	x	x

You can select the following types of constraints from the **Constraint Type** drop-down list:

- **PRIMARY KEY**

  The primary key is the unique identity of a row and consists of one or more columns.

  Only one primary key can be specified for a table, either as a column constraint or as a table constraint. The primary key constraint must name a set of columns that is different from other sets of columns named by any unique constraint defined for the same table.

  Set the constraint type to **PRIMARY KEY** and enter the constraint name. Select a column from the **Available Columns** list and click **Add**. If you need a multi-column primary key, repeat this step for another column.

  Fill Factor for a table is in the range 10 and 100 (unit: %). The default value is 100 (filled to capacity). If **Fill Factor** is set to a smaller value, the INSERT operation fills only the specified percentage of a table page. The free space of the page will be used to update rows on the page. In this way, the UPDATE operation can place the updated row content on the original page, which is more efficient than placing the update on a different page. Set it to 100 for a

table that has never been updated. Set it to a smaller value for largely updated tables. TOAST tables do not support this parameter.

DEFERRABLE: Defer an option.

INITIALLY DEFERRED: Check the constraint at the specified time point.

In the **Constraints** area, click **Add**.

You can click **Delete** to delete a primary key from the list.

Mandatory parameters are marked with asterisks (*).

- **UNIQUE**

  Set the constraint type to **UNIQUE** and enter the constraint name.

  Select a column from the **Available Columns** list and click **Add**. To configure unique for multiple columns, repeat this step for another column. After adding the first column, the UNIQUE constraint name will be automatically filled. The name can be modified.

  You can click **Delete** to delete UNIQUE from the list.

- **CHECK**

  Set the constraint type to **CHECK** and enter the constraint name.

  When the INSERT or UPDATE operation is performed, and if the check expression fails, then table data is not altered.

  If you double-click column in **Available Columns** list, it is inserted to **Check Expression** edit line to current cursor position.

  In the **Constraints** area, click **Add**. You can click **Delete** to delete CHECK from the list. Mandatory parameters are marked with asterisks (*).

Defining an Index

Indexes are optional. They are used to enhance database performance. This operation constructs an index on the specified column(s) of the specified table. Select the **Unique Index** check box to enable this option.

Choose the name of the index method from the **Access Method** list. The default method is B-tree.

The **Fill factor** for an index is a percentage that determines how full the index method will try to pack index pages. For B-trees, leaf pages are filled to this percentage during initial index build, and also when extending the index at the right (adding new largest key values). If pages subsequently become completely full, they will be split, leading to gradual degradation in the index's efficiency. B-trees use a default fill factor of 90, but any integer value from 10 to 100 can be selected. If the table is static, then a fill factor of 100 can minimize the index's physical size. For heavily updated tables, an explain plan smaller fill factor is better to minimize the need for page splits. Other indexing methods use different fill factors but work in similar ways. The default fill factor varies between methods.

You can either enter a user-defined expression for the index or you can create the index using the **Available Columns** list. Select the column in the **Available Columns** list and click **Add**. If you need a multi-column index, repeat this step for other columns.

After entering the required information for the new index, click **Add**.

You can also delete an index from the list using the **Delete** button. After defining all indexes, click **Next**.

You can configure the following parameters of an index in a common table.

Table 4-20 Parameters

Parameter	Row-store Table	Column-store Table	ORC Table
Unique Indexes	√	x	x
btree	√	√	x
gin	√	√	x
gist	√	√	x
hash	√	√	x
psort	√	√	x
spgist	√	√	x
Fill Factor	√	x	x
User Defined Expression	√	x	x
Partial Index	√	x	x

SQL Preview

Data Studio generates a DDL statement based on the inputs provided in **Create New table** wizard.

You can only view, select, and copy the query. You cannot edit the query.

- To select all queries, press **Ctrl+A** or right-click and select **Select All**.
- To copy the selected query, press **Ctrl+C** or right-click and select **Copy**.

Click **Finish** to create the table. On clicking the **Finish** button, the generated query will be sent to the server. Any errors are displayed in the dialog box and status bar.

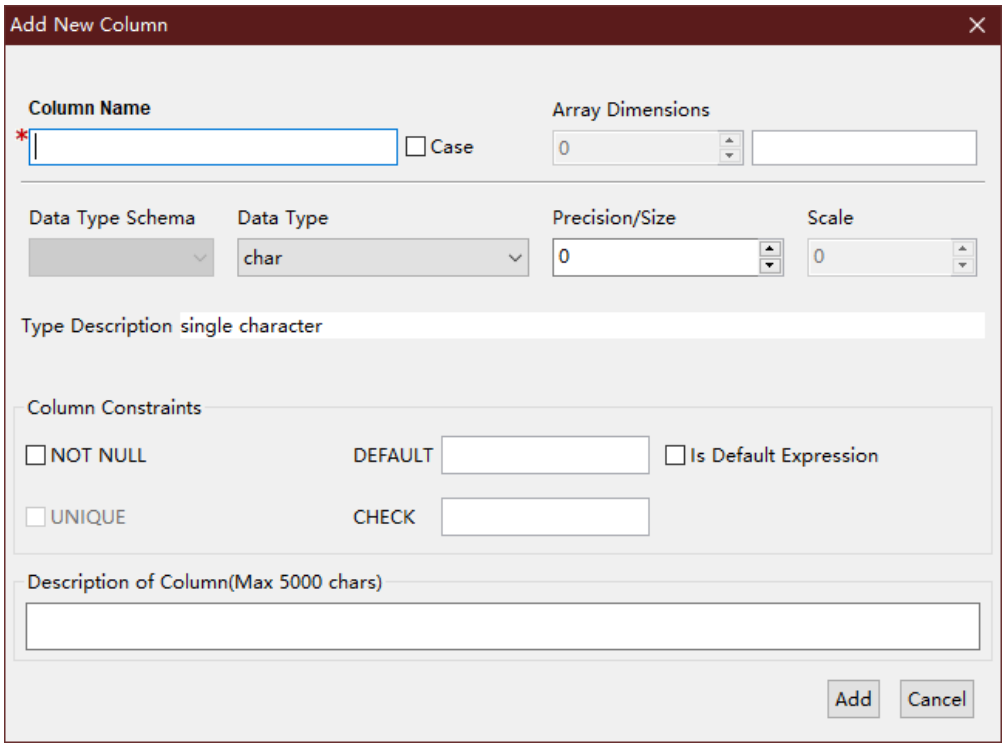
4.6.4.1.2 Managing Columns

After creating a table, you can create a new column, rename a column, and modify attributes in the table.

Creating a New Column

Step 1 Right-click **Columns** and select **Create column**.

The **Add New Column** dialog box is displayed prompting you to add information about the new column. Enter the details and click **Add**.



Step 2 You can view the added column in the corresponding table.

Data Studio displays the status of the operation in the status bar.

----End

Toggle Not Null

Follow the steps below to set or reset the **Not Null** option:

Step 1 Right-click the selected column and select **Toggle Not Null**.

Data Studio displays the **Toggle Not-null Property** dialog box.

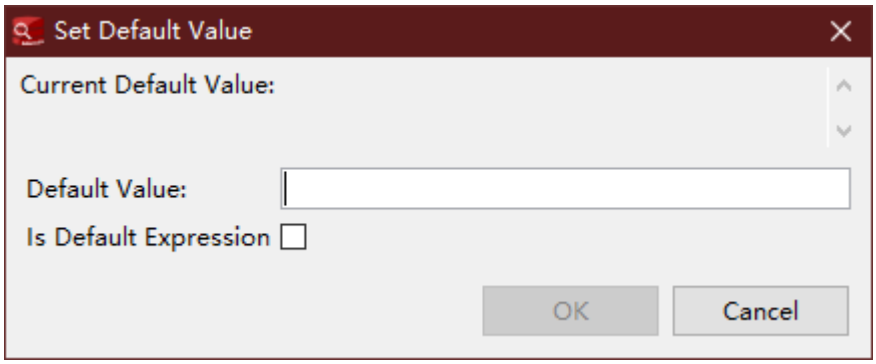
Step 2 Click **OK** to complete the operation successfully. Data Studio displays the status of the operation in the status bar.

----End

Setting the Default Value of a Column

Step 1 Right-click the selected column and select **Set Column Default Value**.

A dialog box with the current default value (if it is set) is displayed, prompting you to provide the default value.



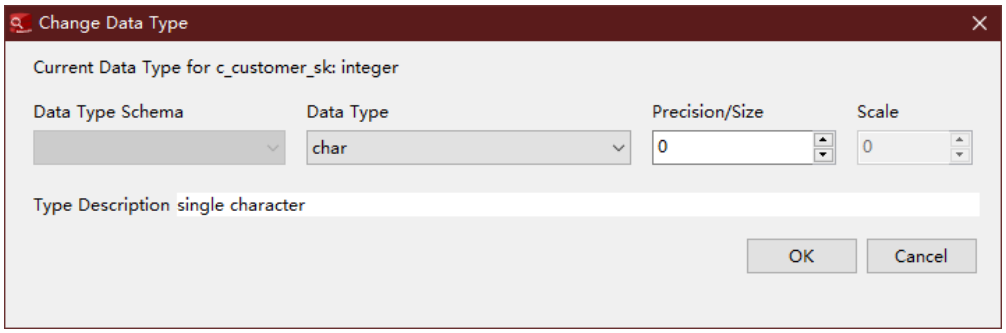
Step 2 Enter the value and click **OK**. Data Studio displays the status of the operation in the status bar.

----End

Changing the Data Type

Step 1 Right-click the selected column and select **Change Data Type**.

Change Data Type dialog box is displayed.



NOTE

The existing data type will be displayed as **Unknown** while you modify complex data types.

Step 2 Select the **Data type Schema** and **Data Type**. If the **Precision/Size** spin box is enabled, enter the required details and click **OK**. Data Studio displays the status of the operation in the status bar.

----End

Renaming a Column

Step 1 Right-click the selected column and select **Rename Column**.

The **Rename Column** dialog box is displayed prompting you to provide the new name.

Step 2 Enter the name and click **OK**. Data Studio displays the status of the operation in the status bar.

----End

Dropping a Column

Step 1 Right-click the selected column and select **Drop Column**. This operation deletes the column from the table.

A **Drop Column** dialog box is displayed.

Step 2 Click **OK** to complete the operation successfully. Data Studio displays the status of the operation in the status bar.

----End

4.6.4.1.3 Managing Constraints

Creating, dropping, and renaming constraints for a created table.

NOTE

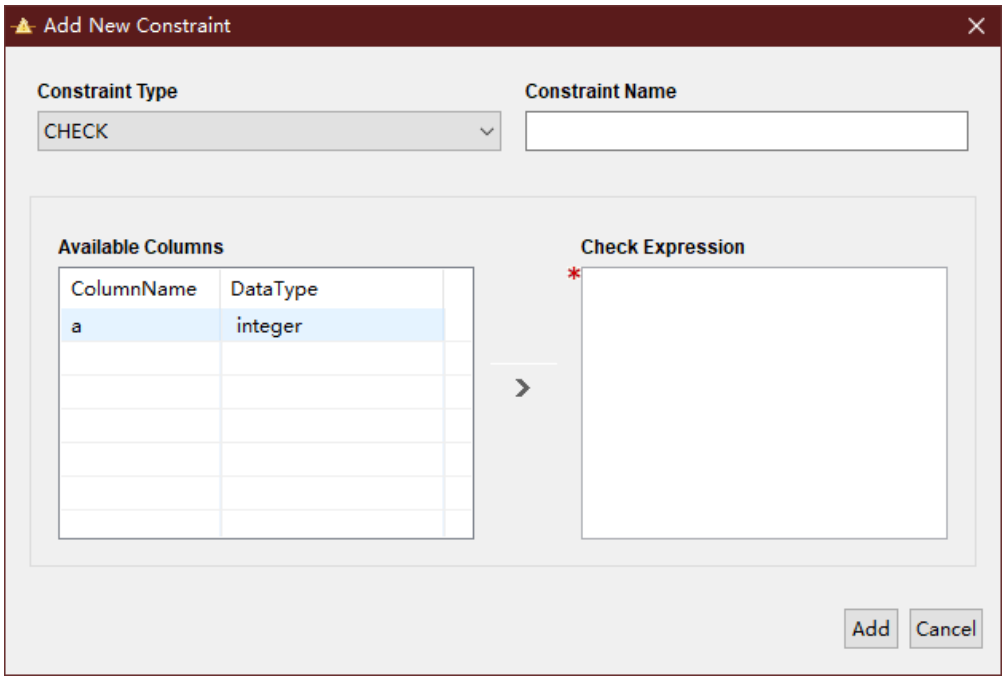
Constraints cannot be added to column-store tables.

Creating a Constraint

Step 1 Right-click the selected constraint of the table and select **Create constraint**.

The **Add New Constraint** dialog box is displayed prompting you to add information about the new constraint.

There are three options of constraint type: CHECK, PRIMARY KEY, and UNIQUE. For details, see **Defining Table Constraints**.



Step 2 Enter the **Constraint Name**, **Check Expression**, and click **Add**. You can view the added constraint in the corresponding table.

Data Studio displays the status of the operation in the status bar.

NOTE

The status bar will show the name of the constraint if it has been provided in the **Constraint Name** field, or else the constraint name will not be displayed as it is created by database server.

----End

Renaming a Constraint

Follow the steps below to rename a constraint:

Step 1 Right-click the selected constraint and select **Rename Constraint**.

The **Rename Constraint** dialog box is displayed prompting you to provide the new name.

Step 2 Enter the constraint name and click **OK**. Data Studio displays the status of the operation in the status bar.

----End

Dropping a Constraint

Follow the steps below to drop a constraint:

Step 1 Right-click the selected constraint and select **Drop Constraint**.

The **Drop Constraint** dialog box is displayed.

Step 2 Click **OK** to complete the operation successfully. Data Studio displays the status of the operation in the status bar.

----End

4.6.4.1.4 Managing Indexes

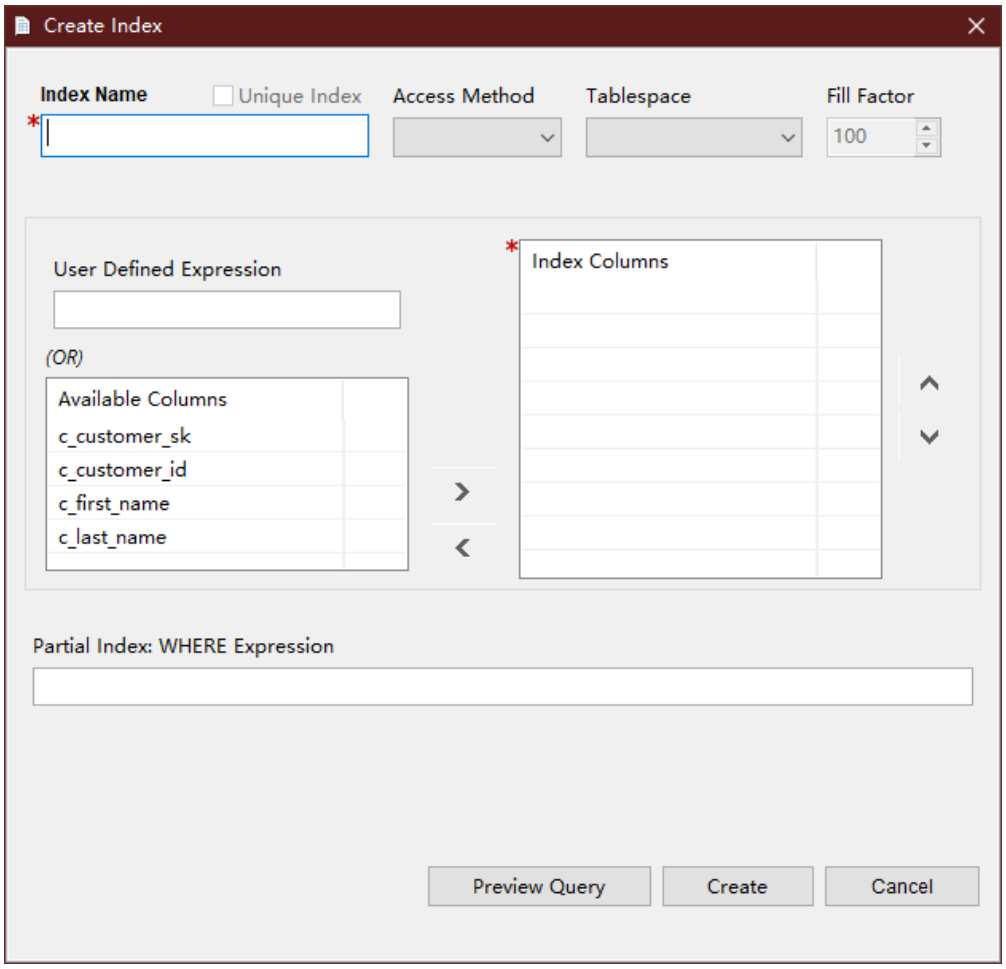
You can create indexes in a table to search for data efficiently.

After a table is created, you can add indexes to it.

Creating an Index

Step 1 Right-click **Indexes** and choose **Create Index** from the shortcut menu.

The **Create Index** dialog box is displayed.



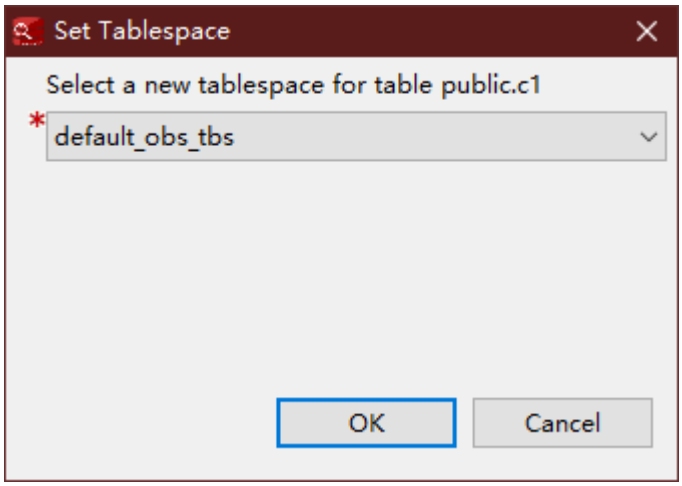
Step 2 Enter the details and click **Create**. You can also view the SQL statement by clicking the **Preview Query** button. Items in **Available Columns** are not sorted. Items moved back from **Index Columns** to **Available Columns** are unsorted, and is not related to the column order in the table. You can set the order of the **Index Columns** using the arrow buttons. Data Studio displays the operation status in the status bar.

----End

Setting a Tablespace

Step 1 Right-click an index and choose **Set Tablespace** from the shortcut menu.

The **Set Tablespace** dialog box is displayed.



Step 2 Select a tablespace and click **OK**. Data Studio displays the operation status in the status bar.

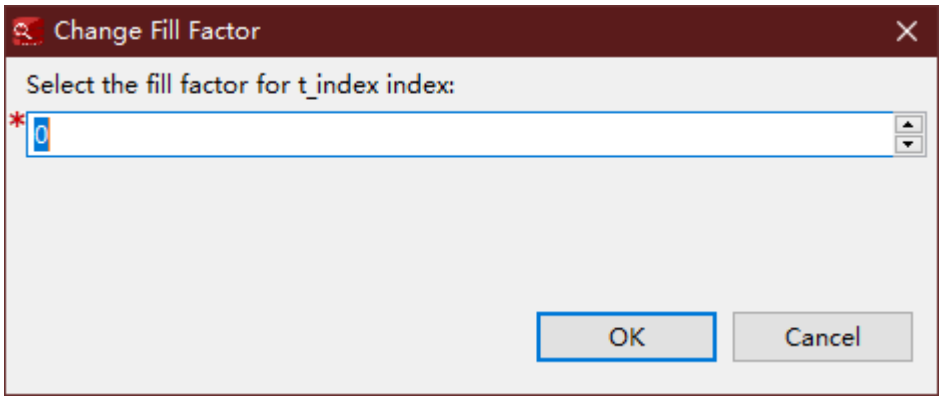
----End

Changing a Fill Factor

To change a fill factor, perform the following steps:

Step 1 Right-click an index and choose **Change Fill Factor** from the shortcut menu.

The **Change Fill Factor** dialog box is displayed.



Step 2 Select a fill factor and click **OK**. Data Studio displays the operation status in the status bar.

----End

Renaming an Index

Follow the steps below to rename an index:

Step 1 Right-click the selected index and select **Rename Index**.

The **Rename Index** dialog box is displayed.

Step 2 Enter a new name and click **OK**. Data Studio displays the operation status in the status bar.

----End

Deleting an Index

Perform the following steps to delete an index:

Step 1 Right-click an index and choose **Drop Index** from the shortcut menu.

The **Drop Index** dialog box is displayed.

Step 2 In the confirmation dialog box, click **OK**. Data Studio displays the operation status in the status bar. The index will be deleted from the table.

NOTE

When the last index of a table is deleted, the value of the **Has Index** parameter may still be **TRUE**. After a vacuum operation is performed on the table, this parameter will change to **FALSE**.

----End

4.6.4.2 Creating a Partitioned Table

Partitioning refers to splitting what is logically one large table into smaller physical pieces based on specific schemes. The table based on the logic is called a partitioned table, and a physical piece is called a partition. Data is stored on these smaller physical pieces, namely, partitions, instead of the larger logical partitioned table.

Follow the steps below to define a table in your database:

Step 1 In the **Object Browser** pane, right-click **Regular Tables**, and choose **Create Partition Table**.

Step 2 Define basic table information, such as the table name and table type. For details, see **Providing Basic Information**.

Step 3 Define column information, such as the column name, data type schema, data type, and column constraint. For details, see **Defining a Column**.

Step 4 Select the data distribution information for the table. For details, see **Configuring Data Distribution**.

Step 5 Define column constraints for different constraint types. Constraint types include **PRIMARY KEY**, **UNIQUE**, and **CHECK**. For details, see **Defining Table Constraints**.

Step 6 Define table index information, such as the index name and access mode. For details, see **Defining an Index**.

Step 7 Define the partition information for the table such as partition name, partition column, partition value and so on. For details, see **Defining a Partition**.

On the **SQL Preview** tab, you can check the automatically generated SQL query. For details, see **SQL Preview**.

Step 8 To add comments to **Column** in the **Create Partition Table** dialog box, add column information in **Description of Column (Max 5000 chars)** text box and click **Add**.

----End

Providing Basic Information

If you create a table in a schema, the current schema will be used as the schema of the table. Perform the following steps to create a partitioned table:

Step 1 Select a table storage mode from the **Table Orientation** drop-down list.

NOTE

If table orientation is selected as ORC, then an HDFS Partitioned table is created. Enter the ORC version number in the **ORC Version** field.

Step 2 After providing the general information about the table, click **Next** to define the columns information for the table.

The following table describes the parameters of partitioned tables.

Table 4-21 Parameters

Parameter	Row Partition	Column Partition	ORC Partition
Table Type	x	x	x
If Not Exists	√	√	√
With OIDS	x	x	x
Fill Factor	√	x	x

----End

Defining a Column

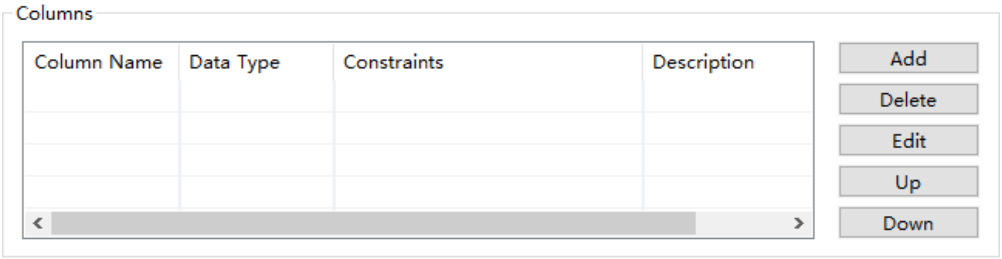
The following table describes the parameters of partitioned tables.

Table 4-22 Parameters

Parameter	Row Partition	Column Partition	ORC Partition
Array Dimensions	√	x	x
Data Type	√	x	x
NOT NULL	√	√	√
Default	√	√	√
UNIQUE	√	x	x
CHECK	√	x	x

You can add, delete, and edit columns, and adjust the sequence of columns.

You can change the order of partitions in the table as required. To change the order, select the required partition and click **Up** or **Down**.



SQL Preview

Data Studio generates a DDL statement based on the inputs provided in **Create New table** wizard.

You can only view, select, and copy the query. You cannot edit the query.

- To select all queries, press **Ctrl+A** or right-click and select **Select All**.
- To copy the selected query, press **Ctrl+C** or right-click and select **Copy**.

Click **Finish** to create the table. On clicking the **Finish** button, the generated query will be sent to the server. Any errors are displayed in the dialog box and status bar.

Defining a Partition

The following table describes the parameters of partitioned tables.

Table 4-23 Parameters

Parameter	Row Partition	Column Partition	ORC Partition
Partition Type	By Range	By Range	By Value
Partition Name	√	√	x
Partition Value	√	√	x

Step 1 If **Row** or **Column** is selected for **Table Orientation** on the **General** tab, **By Range** will be displayed in the **Partition Type** area. If **ORC** is selected for **Table Orientation** on the **General** tab, **By Value** will be displayed in the **Partition Type** area.

Step 2 In the **Available Column** area, select a column and click the Right Arrow button. The column will be moved to the **Partition Column** area.

NOTE

- If **Table Orientation** is set to **Row** or **Column**, only one column can be selected for partitioning.
- If **Table Orientation** is set to **ORC**, up to four columns can be selected for partitioning.
- A maximum of four columns can be selected to define partitions.

Step 3 Enter a partition name.

Step 4 Click the **Enter Partition Value** button next to **Partition Value**. Enter the value by which you want to partition the table in **Value** column. Click **OK**.

Step 5 After you enter all information for partition, click **Add**.

You can add, delete, edit and move a column.

Change the partition sequence according to the requirements in the table. To change the order, select the required partition and click **Up** or **Down**.

Step 6 After defining all partitions, click **Next**.

----End

Defining an Index

For details about index definitions, see **Defining an Index**.

Table 4-24 Parameters

Parameter	Row-store Table	Column-store Table	ORC Table
Unique Indexes	√	x	x
btree	√	√	x
gin	√	√	x
gist	√	√	x
hash	√	√	x
psort	√	√	x
spgist	√	√	x
Fill Factor	√	x	x

Parameter	Row-store Table	Column-store Table	ORC Table
User Defined Expression	√	x	x
Partial Index	√	x	x

Defining Table Constraints

For details about how to define table constraints, see **Defining Table Constraints**.

Table 4-25 Parameters

Parameter	Row Partition	Column Partition	ORC Table
Check	√	x	x
Unique	√	x	x
Primary Key	√	x	x

Configuring Data Distribution

For details about how to configure a distribution type, see **Selecting Data Distribution**.

Table 4-26 Parameters

Parameter	Row Partition	Column Partition	ORC Partition
DEFAULT DISTRIBUTION	√	√	x
Hash	√	√	√
Replication	√	√	x

Dropping a Partition

Step 1 Right-click the selected index and select **Drop Partition**.

Drop Partition Table dialog box is displayed.

Step 2 Click **OK**.

The partition is deleted from the table. Data Studio displays the status of the operation in the status bar.

----End

Renaming a partition

Step 1 Right-click the selected partition and select **Rename Partition**.

Rename Partition Table dialog box is displayed prompting you to provide the new name for the partition.

Step 2 Enter new name and click **OK**.

Data Studio displays the status of the operation in the status bar.

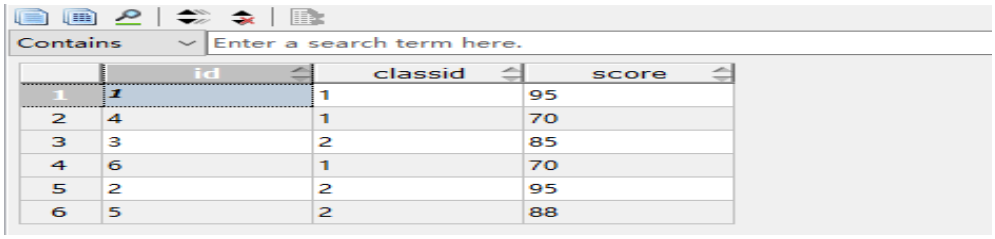
----End

4.6.4.3 Managing Table Data

After creating a table, you can query, edit, and analyze the table and table data.

Viewing Table Data

Right-click the selected table and select **View Table Data**. The **View Table Data** tab is displayed where you can view the table data information.



Toolbar menu in the **View Table Data** window:

Toolbar Name	Description
Copy	Click the icon to copy selected content from **View Table Data** window to clipboard. Shortcut key - **Ctrl+C**.
Advanced Copy	Click the icon to copy content from result window to the clipboard. Results can be copied to include the row number and/or column header. Refer to **Table 4-6** to set this preference. The shortcut key is **Ctrl+Shift+C**.
Show/Hide Search Bar	Click the icon to display/hide the search text field. This is a toggle button.
Encoding	Refer to **Execute SQL Queries** for information on encoding selection.

Icons in the Search field:

Icon Name	Description
Search	Click the icon to search the table data displayed based on the criteria defined. The text is case-insensitive.
Clear Search Text	Click the icon to clear the search texts entered in the search field.

Refer to **Execute SQL Queries** for column reordering and sorting options.

- **Query Submit Time**: Provides the query submitted time.
- Number of rows fetched with execution time is displayed. The default number of rows is displayed. If there are additional rows to be fetched, then it will be denoted with the word "more". You can scroll to the bottom of the table to fetch and display all rows.

NOTICE

- When you view table data, Data Studio automatically adjusts the column widths for an optimal table view. You can resize the columns as needed. If the text content of a cell exceeds the total available display area, then resizing the cell column may cause DS to become unresponsive.
- When the data in a table cell is more than 1000 characters, it will be trimmed to 1000 characters with "..." at the end.

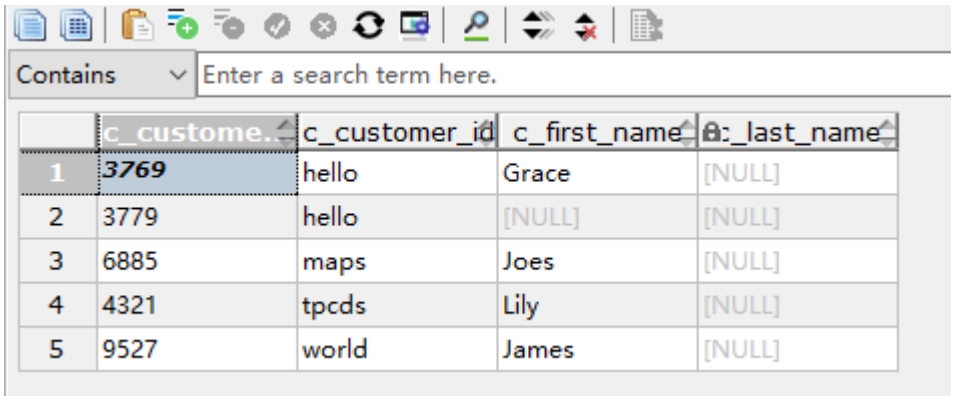
NOTE

- Individual table data window is displayed for each table.
- If the data of the table that is already opened is modified, then refresh and open the table data again to view the updated information on the same opened window.
- While the data is loading a message displays at the bottom stating "fetching".
- If the text of a column contains spaces, word wrapping is applied to fit the column width. Word wrapping is not applied to columns without spaces.
- Select part of cell content and press **Ctrl**+**C**.
- The size of a column is determined by the maximum content length in the column.

Editing Table Data

Right-click the selected table and select **Edit Table Data**. The **Edit Table** data tab is displayed.

Refer to **Viewing Table Data** for description on copy and search toolbar options.



Data Studio validates only the following data types entered into cells: Bigint, bit, boolean, char, date, decimal, double, float, integer, numeric, real, smallint, tinyint, and varchar.

Editing of array data, time with time zone, and time stamp with time zone are not supported.

Any related errors during this operation reported by database will be displayed in Data Studio.

Reindexing a Table

Index facilitates lookup of records. You need to reindex tables in the following scenarios:

- The index is corrupted and no longer contains valid data. Although in theory this will never happen, in practice, indexes can become corrupted due to software bugs or hardware failures. Reindexing provides a recovery method.
- An index contains many empty or almost empty pages. The B-tree index of the GaussDB(DWS) database may encounter this problem in some scenarios. Using REINDEX to create a new index without empty pages can reduce the space usage.
- You have altered a storage parameter (such as the fill factor) for an index, and wish to ensure that the change has taken full effect.

Follow the steps below to reindex a table:

Right-click the selected table and select **Reindex Table**. A pop-up message and the status bar display the status of the completed operation.

NOTE

This operation is not supported for Partition ORC tables.

Analyzing a Table

The analyzing table operation collects statistics about tables and table indexes and stores the collected information in internal tables of the database where the query optimizer can access the information and use it to help make better query planning choices.

Right-click the selected table and select **Analyze Table**. The **Analyze Table** message and status bar displays the status of the completed operation.

Truncating a Table

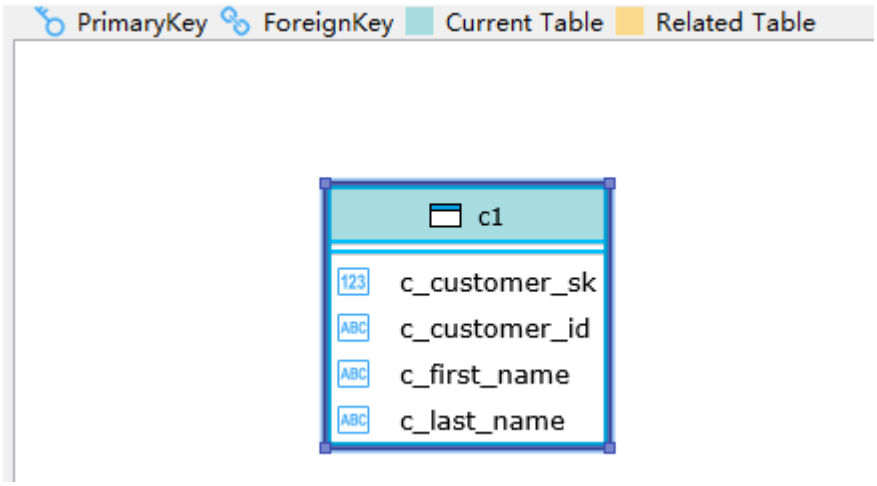
This operation deletes the all data from an existing table. Exercise caution when performing this operation.

Right-click the selected table and select **Truncate Table**. Data Studio prompts you to confirm this operation. In the confirmation dialog box, click **OK** to complete the operation successfully.

A pop-up message and the status bar display the status of the completed operation.

ER Diagrams

View table relationships, including primary keys, foreign keys, and associated tables with ER diagram.



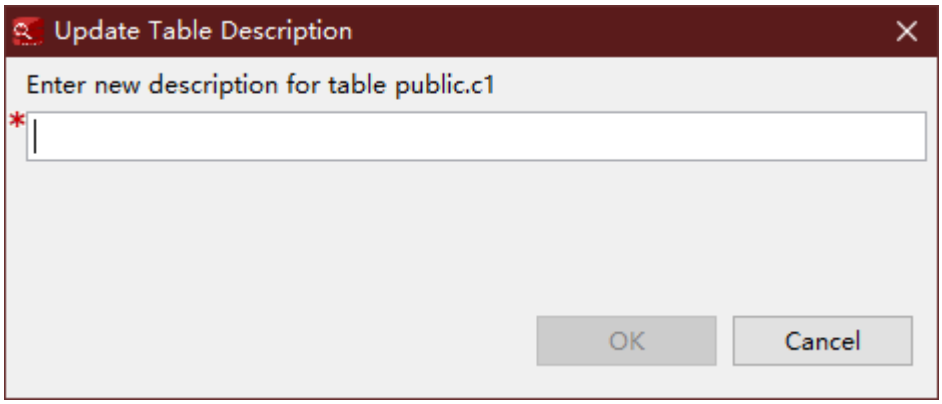
Vacuuming a Table

Vacuuming table operation reclaims space and makes it available for re-use.

Right-click the selected table and select **Vacuum Table**. The **Vacuum Table** message and status bar display the status of the completed operation.

Setting the Table Description

Right-click the selected table and select **Set Table Description**. The **Update Table Description** dialog box is displayed. It prompts you to set the table description.

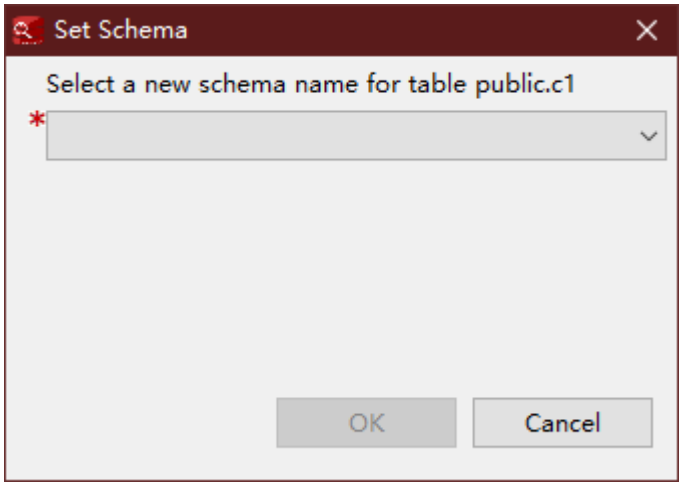


Enter the description and click **OK**. The status bar displays the status of the completed operation.

Setting a Schema

Right-click the selected table and select **Set Schema**. The **Set Schema** dialog box is displayed, prompting you to select the new schema for the selected table.

Select the schema name from the drop-down list and click **OK**. The selected table will be moved to the new schema.



The status bar displays the status of the completed operation.

NOTE

- This operation is not supported for partitioned ORC tables.
- If the required schema contains a table with the same name as the current table, then Data Studio does not allow setting the schema for the table.

Exporting Table Data

Step 1 Right-click the selected table and select **Export Table Data**.

The **Export Table Data** dialog box is displayed with the following options:

- **Format**: Table data can be exported in Excel (xlsx/xls), CSV, TXT, or binary format. The default format is Excel (xlsx/xls).
- **Include Header**: This option is available for CSV and TXT files. If this option is selected, the exported data will include the column headers. By default, this option is selected when a CSV or TXT file is exported, but it is not mandatory. This option will be disabled for Excel (xlsx/xls) and binary formats.
- **Quotes**: Use this option to define quotes. You can enter only a single-byte character for this option, and the value of **Quotes** should be different from that of **Delimiter**. By default, this option is selected when a CSV or TXT file is exported, but it is not mandatory. This option will be disabled for Excel (xlsx/xls) and binary formats.
  - If table data includes delimiters, the character specified in this option will be used.

  - If the value includes a quote, it will not be escaped by the same quote.
  - If the result contains the values of multiple rows, it will be quoted by quotes.
- **Escape**: Use this option to define escape values. You can enter only a single-byte character for this option, and the value of **Escape** must be different from that of **Quotes**. By default, this option is selected when a CSV or TXT file is exported, but it is not mandatory. This option will be disabled for Excel (xlsx/xls) and binary formats.
- **Replace NULL with**: Use this option to replace the null value in the table with a specified string. This option contains a maximum of 100 characters, and cannot contain newline characters or carriage return characters. The value of this option must be different from the values of **Delimiter** and **Quotes**. By default, this option is selected when a CSV or TXT file is exported, but it is not mandatory. This option will be disabled for Excel (xlsx/xls) and binary formats.
- **Encoding** (optional): This option will be pre-populated with the encoding options made in the **Preferences > Session Setting** tab.
- **Delimiter**: Use this option to define delimiters. You can select the provided delimiters or customize delimiters in **Delimiter** > **Other**. The default delimiter for CSV and TXT formats is commas (,). The **Other** field can contain a maximum of 10 bytes. By default, this option is selected when a CSV or TXT file is exported, but it is not mandatory. This option will be disabled for Excel (xlsx/xls) and binary formats. It is mandatory when the **Other** field is selected.
- **All Columns**: Use this option to quickly select all columns. This option is selected by default. To manually select columns, deselect this option and select the columns to export from the **Available Columns** list.
  - **Available Columns**: Use this option to select the columns to export.
  - **Selected Columns**: This option displays the selected columns to export. The column sequence can be adjusted. By default, all columns are displayed in this option.

    NOTE

    The .xlsx format supports a maximum of 1 million rows and 16,384 columns. The .xls format supports a maximum of 64,000 rows and 256 columns.
- **File Name**: Use this option to specify the name of the exported file. By default, the table name is displayed in this option.

  NOTE

  The file name follows the Windows file naming convention.
- **Output Path**: Use this option to select the location where the exported file is saved. The **Output Path** field is auto-populated with the selected path.
- **Security Disclaimer**: This option displays the security disclaimer. To continue the export operation, you need to read and agree to the disclaimer.
  - **I Agree**: By default this option is selected. You cannot proceed if this option is deselected.
  - **Do not show again**: You can select this option to hide the **Security Disclaimer** for the subsequent table data export from the current Data Studio instance.

NOTE

- String, double, date, calendar, and Boolean data will be stored in the Excel format. All other data types will be converted into strings and stored in the Excel format.
- To export an Excel file, if a cell contains more than 32767 characters, the data exported to the cell will be truncated.

Step 2 Complete the required fields and click **OK**.

The **Save As** dialog box is displayed.

Step 3 Click **Save** to save the exported data in the selected format. The status bar displays the progress of the operation.

The **Data Exported Successfully** dialog box and status bar displays the status of the completed operation.

NOTE

- If the disk is full during table export, Data Studio displays an I/O error. Perform the following operations to rectify this error:
  1. Click **OK** to close the connection profile.
  2. Clean the disk.
  3. Re-establish the connection and export the table data.
- The exported file name will not be the same as table name, if the table name contains characters which are not supported by Windows.

----End

Importing Table Data

Prerequisites:

- If the definition of the source file does not match that of the destination table, modify the properties of the destination table in the **Import Table Data** dialog box. Additional columns of the destination table will be inserted with default values.
- You should know the export properties of the file to be imported, such as **Delimiter**, **Quotes**, and **Escape**. Export properties saved during data export cannot be changed when a file is being imported.

Perform the following steps to import table data:

Step 1 Right-click the selected table and select **Import Table Data**.

The **Import Table Data** dialog box is displayed with the following options:

- **Import Data File**: This option displays the path of the imported file. Click **Browse** to select other files.
- **Format**: Table data can be imported in CSV, TXT, or binary format. CSV is the default format.
- **Include Header**: Select this option if the imported file contains a column header. By default, this option is selected when a CSV or TXT file is exported, but it is not mandatory. This option will be disabled for the binary format.
- **Quotes**: You can enter only a single-byte character for this option, and the value of **Quotes** should be different from the null value of **Delimiter**. By

default, this option is selected when a CSV or TXT file is exported, but it is not mandatory. This option will be disabled for the binary format.

- **Escape**: You can enter only a single-byte character for this option. If the value of **Escape** is the same as that of **Quotes**, the value of **Escape** will be replaced with **\0**. This option defaults to double quotation marks (") when a CSV or TXT file is exported, but it is not mandatory. This option will be disabled for the binary format.
- **Replace with Null**: You can configure this option to replace the null value in the table with a string. The null string used for exporting data should be used for importing data, and the null string needs to be specified. By default, this option is selected when a CSV or TXT file is exported, but it is not mandatory. This option will be disabled for the binary format.
- **Encoding** (optional): This option will be pre-populated with the encoding options made in the **Preferences > Session Setting** tab.
- **Delimiter**: You can select the provided delimiters or customize delimiters in **Delimiter** > **Other**. The default delimiter for CSV and TXT formats is commas (,). The value of this option should be different from those of **Quotes** and **Replace with Null**. By default, this option is selected when a CSV or TXT file is exported, but it is not mandatory. This option will be disabled for the binary format. It is mandatory when the **Other** field is selected.
- **All Columns**: Use this option to quickly select all columns. This option is selected by default. To manually select columns, deselect this option and select the columns to export from the **Available Columns** list.
  - **Available Columns**: Use this option to select the columns to export.
  - **Selected Columns**: This option displays the selected columns to export. The column sequence can be adjusted. By default, all columns are displayed in this option.

Step 2 Click **Browse** next to the **Import Data File** field.

The **Open** dialog box is displayed.

Step 3 In the **Open** dialog box, select the file to import and click **Open**.

Step 4 Complete the required fields and click **OK**.

The status bar displays the operation progress. The imported data will be added to the existing table data.

The **Data Imported Successfully** dialog box and status bar display the status of the completed operation.

----End

Show DDL

Right-click the table, and then select **Show DDL**. Data Studio displays the DDL of the selected table.

NOTE

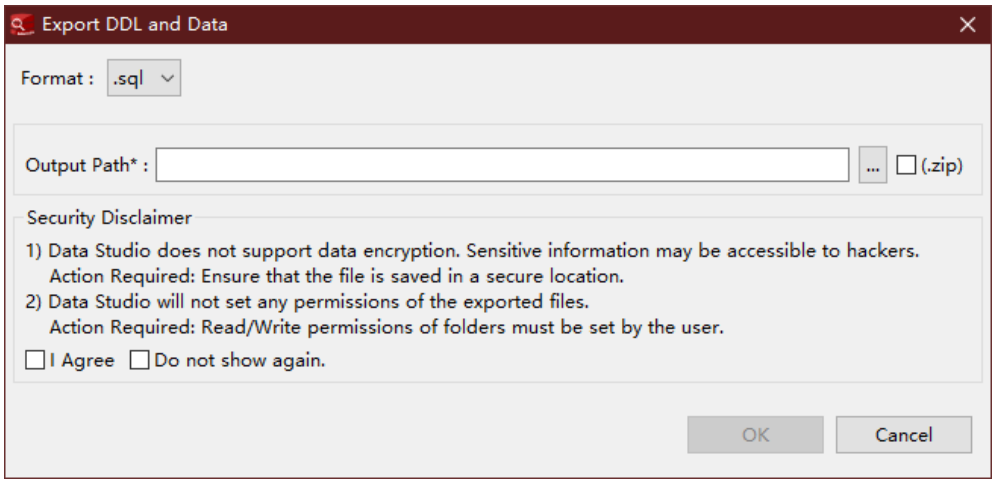
- A new terminal window is opened each time you select to show DDL.
- MS Visual C runtime file (msvcrt100.dll) is required to complete this operation. For details, see **Troubleshooting**.

Exporting Table DDL and Data

Follow the steps below to export the table DDL:

Step 1 In the **Object Browser** pane, right-click the selected table and select **Export DDL and Data**.

You need to customize the export path. To compress data, select **.zip**



You must click **I agree** under **Security Disclaimer**, then click **OK**. You can disable the security disclaimer. After the disclaimer is disabled, it will not be displayed when you export the DDL. For details, see **Table 4-6**.

Step 2 Click **OK**. The operation progress is displayed on the status bar in the lower right corner.

NOTE

- If the table name contains characters which are not supported by Windows, the exported file name will not be the same as table name.
- MS Visual C runtime file (msvcrt100.dll) is required to complete this operation. For details, see **Troubleshooting**.

The **Export** message and status bar display the status of the completed operation.

Table 4-27 Supported DDL encoding formats

Database Encoding	File Encoding	Support for Exporting a DDL
UTF-8	UTF-8	Yes
	GBK	Yes
	LATIN1	Yes
GBK	GBK	Yes
	UTF-8	Yes
	LATIN1	No

Database Encoding	File Encoding	Support for Exporting a DDL
LATIN1	LATIN1	Yes
	GBK	No
	UTF-8	Yes

NOTE

You can select multiple objects from regular and partitioned tables to export DDL and data, including columns, rows, indexes, constraints, and partitions.

----End

Renaming a table

Right-click the selected table and select **Rename Table**. The **Rename Table** dialog box is displayed prompting you to provide the new name.

Enter the table name and click **OK**. You can view the updated table name in **Object Browser**.

NOTE

This operation is not supported for partitioned ORC tables.

Dropping a Table

Right-click the selected table and select **Drop Table**. Press **Ctrl+left-click** (select objects one by one) or **Shift+left-click** (select objects in batches) to select the objects to be dropped.

This operation removes the complete table structure (including the table definition and index information) from the database and you have to re-create this table once again to store data.

Viewing Table Properties

Right-click the selected table and select **Properties**.

Data Studio displays the properties (**General**, **Columns**, **Constraints**, and **Index**) of the selected table in different tabs. The following table lists the operations that can be performed on each tab along with data editing and refreshing operation. Edit operation is performed by double-clicking the cell.

NOTICE

When viewing table data, Data Studio automatically adjusts the column widths for table view. You can resize the columns as needed. If the text content of a cell exceeds the total available display area, then resizing the cell column may cause Data Studio to become unresponsive.

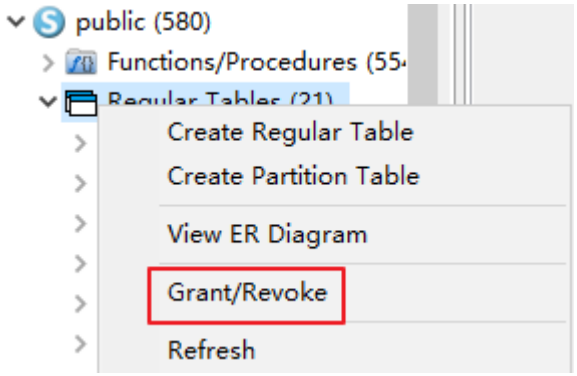
NOTE

- Individual property window is displayed for each table.
- If the property of a table is modified for a table that is already opened, refresh and open the properties of the table again to view the updated information on the same opened window.
- If the content of the column has spaces between the words, then word wrapping is applied to fit the column within the display area. Word wrapping is not applied if the content does not have any spaces between the words.
- The size of the column is determined by the maximum content length column.
- Any change made to the table properties from **Object Browser** will be reflected after refreshing the **Properties** tab.
- Pasting operation is not allowed in the **Data Type** column.

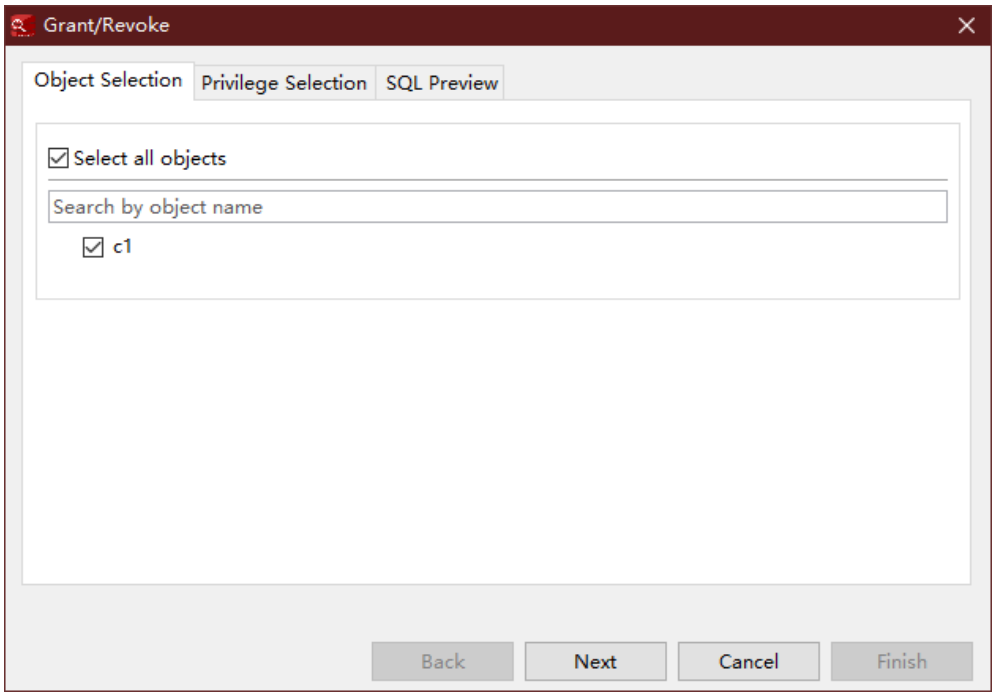
Grant/Revoke Privilege

Step 1 Right-click the selected regular/partitioned table and select **Grant/Revoke**.

The **Grant/Revoke** dialog box is displayed.



Step 2 Select the objects to grant/revoke privilege from the **Object Selection** tab and click **Next**.



Step 3 Select the **role** from the Role drop-down list in the **Privilege Selection** tab. Select **Grant/Revoke** in the **Privilege Selection** tab.

Step 4 On the **SQL Preview** tab, you can check the automatically generated SQL query.

Step 5 Click **Finish**.

----End

4.6.4.4 Editing Temporary Tables

Data Studio allows you to edit temporary tables. Temporary tables are deleted automatically when you close the connection that was used to create the table.

NOTICE

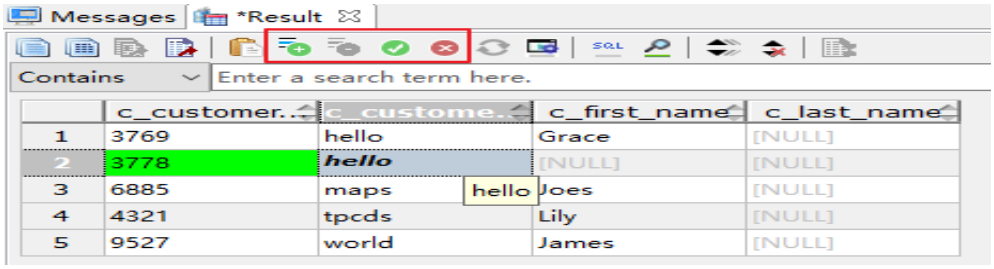
Ensure that connection reuse is enabled when you use edit temporary tables in **SQL Terminal**. For details, see **Managing SQL Terminal Connections**.

Follow the steps to edit a temporary table:

Step 1 Execute a query on the temporary table.

The **Result** tab displays the results of the SQL query along with the query statement executed.

Step 2 Edit the temporary table on the **Result** tab. You can click add, delete, modification, or cancel modification to edit the temporary table.



----End

4.6.4.5 Creating a Foreign Table

NOTICE

Currently, Data Studio does not support foreign table creation. You can only view foreign tables.

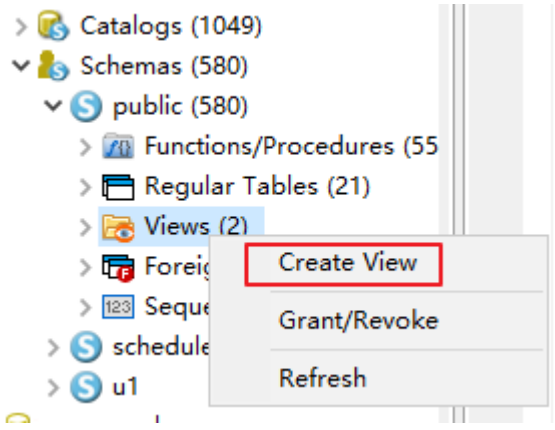
Foreign tables created using query execution in **SQL Terminal** or any other tool can be viewed in the **Object** browser after refresh.

Right-click the database, schema, or foreign table group and choose **Refresh** from the shortcut menu to view the created foreign table.

4.6.5 View Management

Creating a View

Right-click the **Views** and select **Create View**. The DDL template for the view is displayed in the **SQL Terminal** tab.



Granting/Revoking a Privilege

Step 1 Right-click the views group and select **Grant/Revoke**. The **Grant/Revoke** dialog box is displayed.

Step 2 Select the objects to grant/revoke a privilege from the **Object Selection** tab and click **Next**.

Step 3 Select a role from the **Role** drop-down list in the **Privilege Selection** tab. Select **Grant/Revoke** in the **Privilege Selection** tab.

Step 4 On the **SQL Preview** tab, you can check the automatically generated for the inputs provided.

----End

Viewing the DDL

Right-click the selected view and select **Show DDL**.

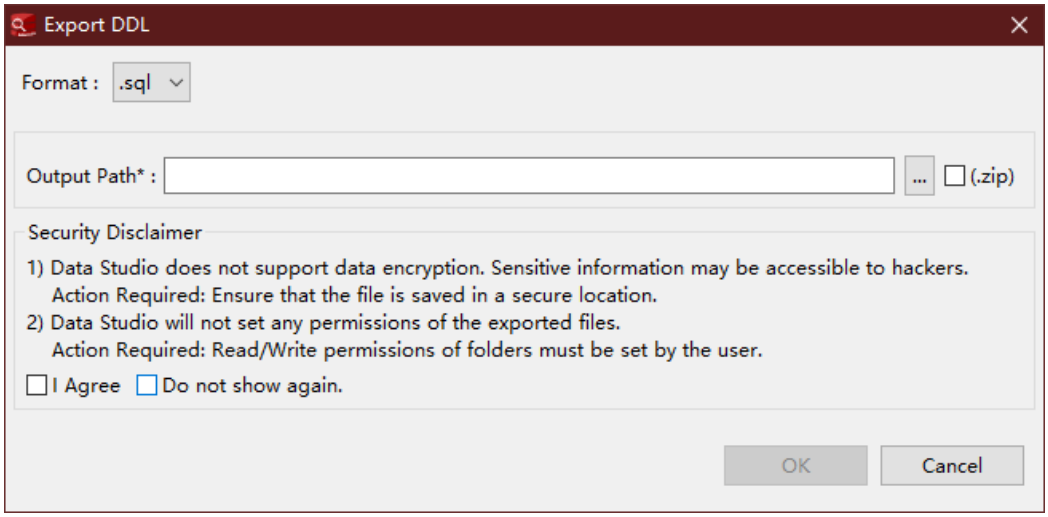
The DDL is displayed in a new **SQL Terminal** tab. You must refresh the **Object Browser** to view the latest DDL.

Exporting the View DDL

Right-click the selected view and select **Export DDL**.

Step 1 In the **Object Browser** pane, right-click the selected schema and select **Export DDL**.

You need to customize the export path. To compress data, select **.zip**



You must click **I agree** under **Security Disclaimer**, then click **OK**. You can disable the security disclaimer. After the disclaimer is disabled, it will not be displayed when you export the DDL. For details, see **Table 4-6**.

Step 2 Click **OK**. The operation progress is displayed on the status bar in the lower right corner.

The **Export** message and status bar display the status of the completed operation.

NOTE

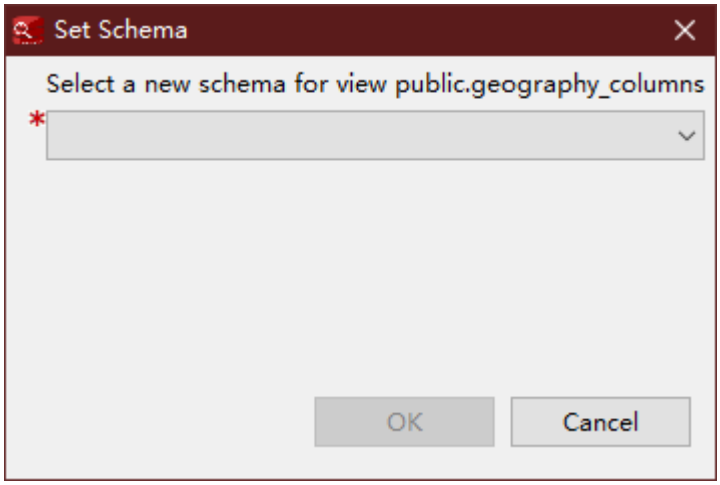
- To cancel the export operation, double-click the status to open the **Progress View** tab and click .
- If the view name contains characters that are not supported by Windows, the name of the exported file is different from the view name.

Database Encoding	File Encoding	Support for Exporting a DDL
UTF-8	UTF-8	Yes
	GBK	Yes
	LATIN1	Yes
GBK	GBK	Yes
	UTF-8	Yes
	LATIN1	No
LATIN1	LATIN1	Yes
	GBK	No
	UTF-8	Yes

----End

Setting the Schema for a View

Step 1 Right-click the selected view and select **Set Schema**. The **Set Schema** dialog box is displayed.



Step 2 Select the required schema from the drop-down list and click **OK**.

The status bar displays the status of the completed operation.

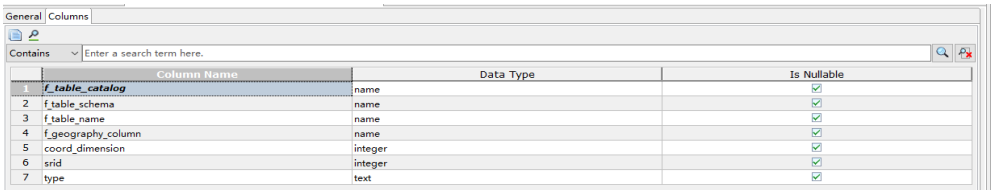
If the required schema contains a view with the same name as the current view, then Data Studio does not allow setting the schema for the view.

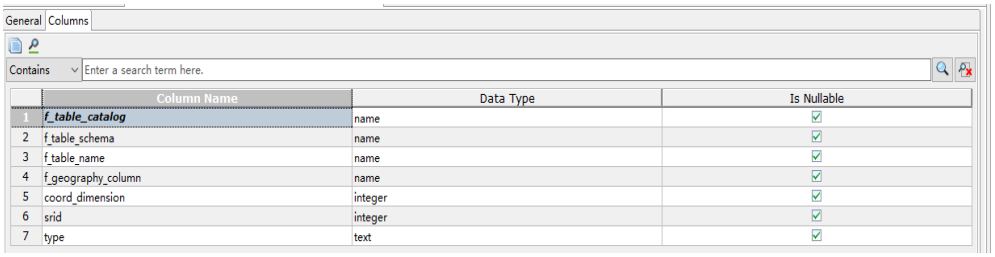
----End

Viewing the Properties of a View

Right-click the selected View and select **Properties**.

The properties (**General** and **Columns**) of the selected view is displayed in different tabs.



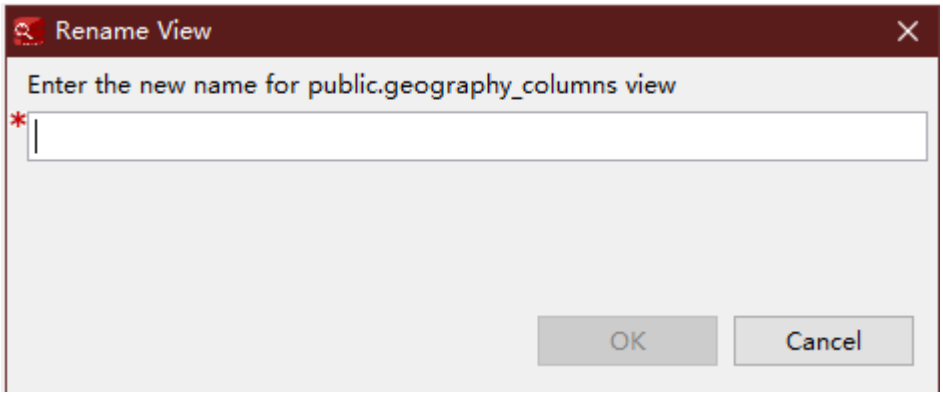


NOTE

If you modify the properties of a view that is already opened, then refresh and open the properties of the view again to check the updated information on the same opened window.

Renaming a View

Step 1 Right-click the selected view and select **Rename View**. The **Rename View** dialog box is displayed.



Step 2 Enter the required name for the view and click **OK**. You can view the renamed view in the **Object Browser**.

The status bar displays the status of the completed operation.

----End

Dropping a View

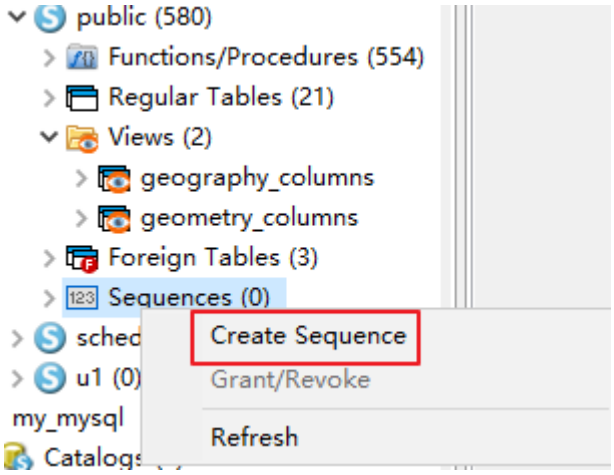
Right-click the selected view and select **Drop View**. The **Drop View** dialog box is displayed. Click **Yes** to delete the view.

Right-click the selected view and select **Drop View Cascade**.

4.6.6 Sequence Management

Creating a Sequence

Step 1 In the **Object Browser** pane, right-click **Sequences** under the particular schema where you want to create the sequence and select **Create Sequence**. The **Create New Sequence** dialog box is displayed.



Step 2 Provide information to create a sequence:

1. Enter a name in the **Sequence Name** field.

   NOTE

   Select the **Case** check box to retain the capitalization of the text entered in **Sequence Name** field. For example, if the entered sequence name is **Employee**, then the sequence name is created as "Employee".

2. Enter the minimum value in the **Minimum Value** field.
3. Enter the increase step value in the **Increment By** field.
4. Enter maximum value in the **Maximum Value** field.

   NOTE

   The maximum and minimum value should be between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807.

5. Enter the start value of the sequence in **Start Value** field.
6. Enter the cache information in **Cache** field. The cache value denotes the number of sequences stored in the memory for quick access.
7. Select the **Cycle** field to recycle sequences after the number of sequences reaches either the maximum or minimum value.

   NOTE

   The schema name auto-populates in the **Schema** field.

8. Select the table from the **Table** drop-down list.
9. Select the column from the **Column** drop-down list.

Step 3 Click **Finish**.

The status bar displays the status of the completed operation.

NOTE

On the **SQL Preview** tab, you can view the SQL query automatically generated for the inputs provided.

----End

Support for Sequence DDL

Data Studio allows you to display sequence DDL and export sequence DDL, including **Show DDL**, **Export DDL**, and **Export DDL and Data**.

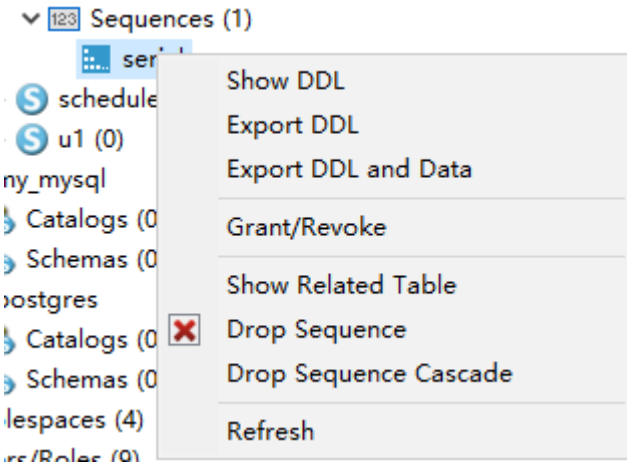
In the **Object Browser** pane, right-click the selected table.

To check DDL statement, choose **Show DDL**.

To export DDL statements, choose **Export DDL**.

To export DDL and SELECT statements, choose **Export DDL and Data**

The procedure is shown in the following figure.



NOTE

Only the sequence owner, system administrator, or a user who has the select permission on the sequence can perform the preceding operations.

Granting/Revoking a Privilege

Step 1 Right-click selected sequence and select **Grant/Revoke**. The **Grant/Revoke** dialog box is displayed.

Step 2 Select the objects to grant/revoke privilege from the **Object Selection** tab and click **Next**.

Step 3 Select the **role** from the Role drop-down list in the **Privilege Selection** tab. Select **Grant/Revoke** in the **Privilege Selection** tab.

Step 4 On the **SQL Preview** tab, you can check the automatically generated SQL query.

Step 5 Click **Finish**.

----End

Dropping a Sequence

Right-click the selected sequence and select **Drop Sequence**. The **Drop Sequence** dialog box is displayed. Click **Yes** to drop the sequence.

Right-click the selected sequence and select **Drop Sequence Cascade**.

4.6.7 Users/Roles

Creating a User/Role

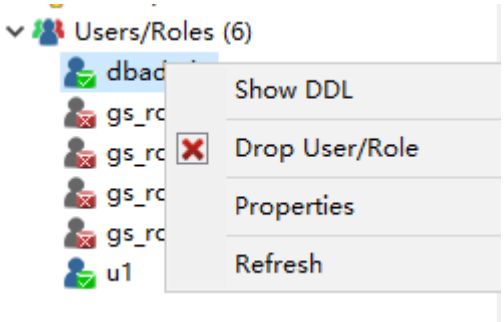
A database is used by many users, and the users are grouped for management convenience. A database role represents a database user or a group of database users.

Users and roles have similar concepts in databases. In practice, you are advised to use a role to manage permissions rather than to access databases.

- **Users**: They are set of database users. These users are different from operating system users. These users can assign permissions to other users to access database objects.
- **Role**: This can be considered as a user or group based on the usage. Roles are at cluster level, and hence applicable to all databases in the cluster. A role is a cluster-level definition and applies to all databases in a cluster.

Dropping a User/Role

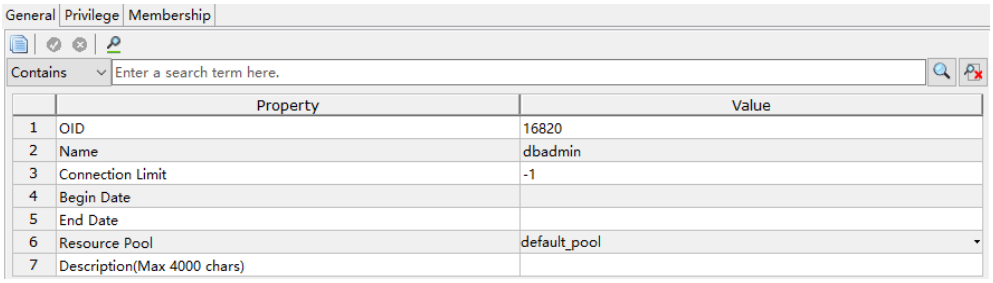
Right-click the selected user/role and select **Drop User/Role**.



Viewing/Editing User/Role Properties

Right-click the selected user/role and select **Properties**.

Data Studio displays the properties (General, Privilege, and Membership) of the selected user/role in different tabs. Editing of properties can be performed. OID is a non-editable field.



Viewing the User/Role DDL

Right-click the selected user/role and select **Show DDL**.

The user/role DDL is displayed in a new **SQL Terminal** tab. You must refresh the **Object Browser** to view the latest DDL.

4.7 SQL Terminal Management

4.7.1 Opening Multiple SQL Terminal Tabs

You can open multiple **SQL Terminal** tabs in Data Studio to execute multiple SQL statements for query in the current **SQL Terminal** tab. Perform the following steps to open a new **SQL Terminal** tab.

You can also open multiple **SQL Terminal** tabs on different connection templates.

Step 1 In the **Object Browser** pane, right-click the desired database and choose **Open Terminal** from the shortcut menu. Alternatively, click **Open Terminal** in the toolbar or press **Ctrl+T** to open a new **SQL Terminal** tab.

The **SQL Terminal** tab is displayed.

The name format of the new **SQL Terminal** tab is as follows:

Database name@*Connection information*(*Tab number*), for example, *postgres*@*IDG_1*(*2*). The number of each **SQL Terminal** tab in the same connection information is unique.

----End

NOTE

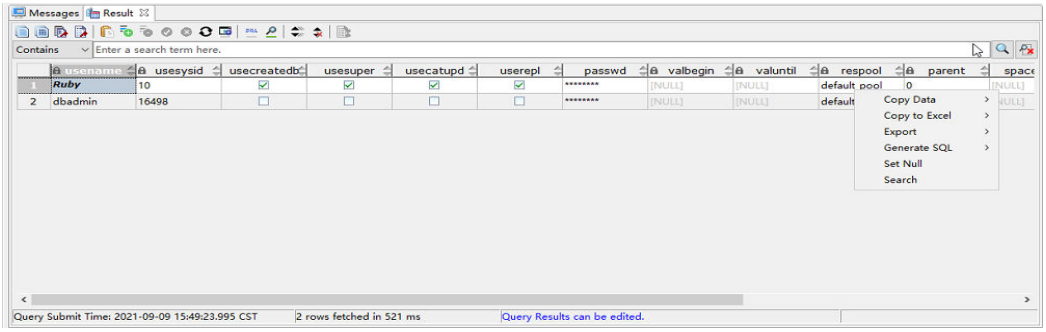
- In Data Studio, a maximum of 100 **SQL Terminal** tabs can be opened for each connected database. Based on the number of query times, each **SQL Terminal** tab contains multiple **Result** tabs and one **Message** tab. If the database connection is lost, the corresponding **SQL Terminal** tab is still available.
- The restoration operation applies to all minimized **SQL Terminal** tabs. You cannot restore a single tab.
- After all terminals are shut down, Data Studio resets the counter of the SQL terminal.
- After all **Result Set** tabs are closed, Data Studio resets the counter of the result set.
- Data Studio allows you to reset counters in the following pages: **Display DDL Tablespace**, **Display DDL User/Role**, **Batch Delete**, **Result Set**, and **Execution Plan**.

Data Studio displays an error message indicating that no result is found in the status bar. The **Result** tab displays the successful execution results.

Right-Click Menus in the Result Tab

You can copy or export cell data to an Excel file and generate a SQL query file.

After the SQL query result is displayed in the **Result** tab, right-click the result. The following menu is displayed:



Perform the following steps to add a row number and column header to the result set:

Step 1 On the menu bar of Data Studio, click **Settings**.

Step 2 Choose **Preferences**. Expand the **Result Management** tab and choose **Query Results**.

Step 3 In the **Result Advanced Copy** area, select **Include column header** and **Include row number**.

----End

The following table describes the right-click options.

Table 4-28 Right-click options

Option	Sub-Item	Description
Copy Data	Copy	Copies data in the selected cell.
	Advanced Copy	Copies data in the selected cell, row number, and column header based on the preference settings.
Copy to Excel	Copy as xls	Exports data of selected cells to an xls file, which contains a maximum of 64,000 rows and 256 columns.
	Copy as xlsx	Exports data of selected cells to an xlsx file, which contains a maximum of 1 million rows.

Option	Sub-Item	Description
Export	Current Page	Exports the table data on the current page.
	All Pages	Exports all tables.
Generate SQL	Selected Line	Selects data from the target table of the statement for inserting data to generate a SQL file.
	Current Page	Selects data of the current page from the target table of the statement for inserting data to generate a SQL file.
	All Pages	Selects all table data from the target table of the statement for inserting data to generate a SQL file.
Set Null	-	Sets the cell data to **null**.
Search	-	Searches for data in the selected cell and displays all data that meets the search criteria.

NOTE

The preceding SQL files do not take effect for the result sets generated by queries that use **JOIN**, expressions, views, **SET** operators, aggregate functions, **GROUP BY** clauses, or column aliases.

Result Tab (Text View)

You can view data in text mode in the **Result** tab.

In addition to the grid view, you can also copy and search for data in the text view.

Click  to obtain the result in text mode.

NOTE

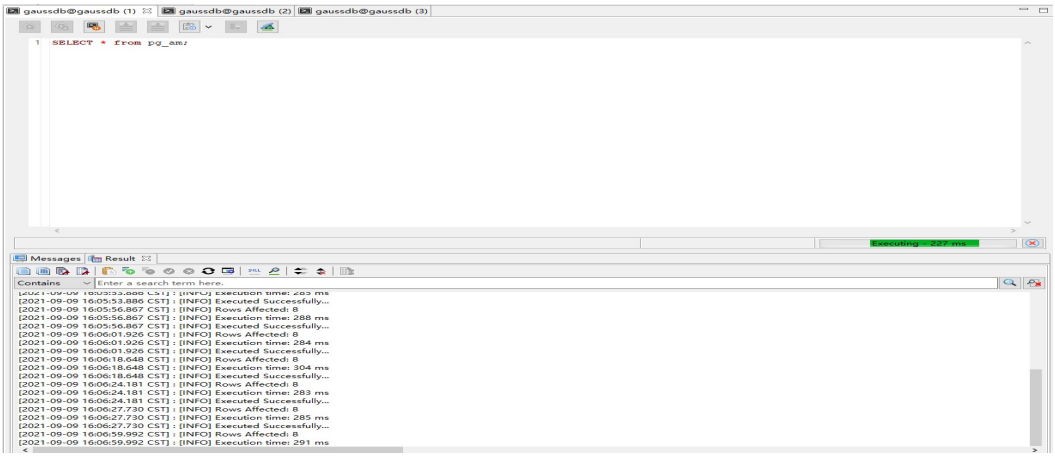
Searching for data in multiple cells may cause the system to display incorrect text results, because all information needs to be copied to the **Search** pane in plain text.

Displaying Execution Progress Bar

When a query is executed in the **SQL Terminal** pane, a progress bar is displayed to dynamically display the execution duration. After the query is complete, the time bar disappears. The total execution duration is displayed next to the time bar.

If you want to cancel the query, click **Cancel** next to the time bar.

The procedure is shown in the following figure.



NOTE

- The **Cancel** button is deleted from the toolbar.
- The progress bar is also displayed when you compile or debug an object in the PL/SQL editor.
- The time format displayed in the progress bar is *w* **hour** *x* **minute** *y* **second** *z* **millisecond**.
- When queries are performed in batches in **SQL Terminal**, the progress bar displays the total time consumed by the queries.

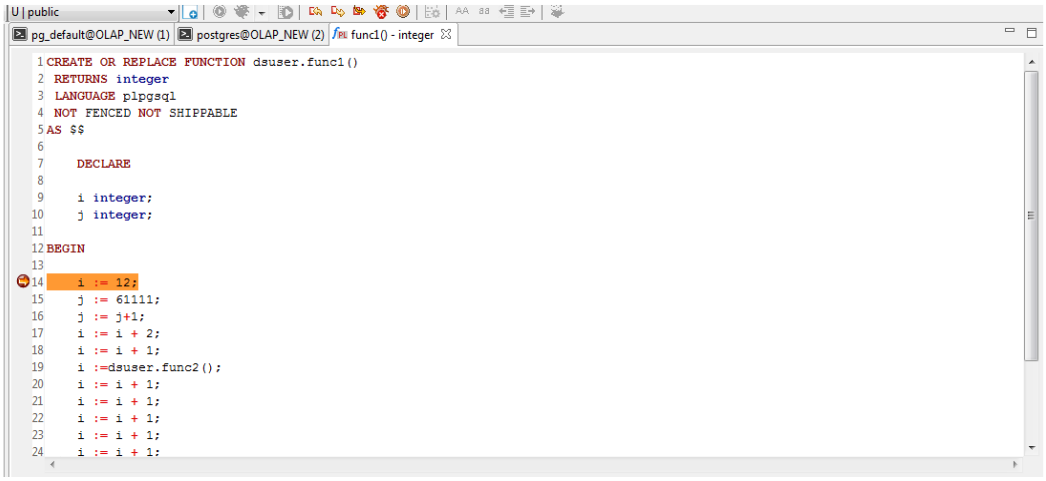
Debugging Duration

During debugging, Data Studio displays the status bar, showing the last operation time and total debugging time of each debugging statement.

During the debugging, the last operation time and total debugging time of the terminal are updated continuously. The value of the total debugging time is the sum of the values of the last operation time.

This simplifies the search for time-consuming statements in the debugged object.

The procedure is shown in the following figure.



NOTE

Functions and procedures can be debugged.

4.7.2 Managing the SQL Query Execution History

Data Studio allows viewing and managing frequently used SQL queries. The history of executed SQL queries is maintained only for the **SQL Terminal**.

Step 1 On the **SQL Terminal** tab page, click **SQL History.** The **SQL History** dialog box is displayed.

----End

NOTE

SQL history scripts are not encrypted.

The number of queries saved in the **SQL History** dialog box is based on the value defined in **Preferences** > **Editor** > **SQL History** pane. Refer to the **Table 4-6** section to modify the SQL History count. Data Studio overwrites the older queries into the SQL history after the list is full. The executed query is automatically stored in the list.

The **SQL History** dialog box has the following columns:

- **Pin Status** - Displays the pinned status of the queries. Pinned queries will always show on the top and it will not be deleted from the history even when the list is full.
- **SQL Statement** - Displays the SQL query. The number of characters for an SQL query displayed in the **SQL Statement** column is based on the number defined in **Preferences** > **Editor** > **SQL History** pane. Refer to the **Table 4-6** section to modify the number of characters for a query.

- **Number of Records** - Displays the number of records fetched by the SQL query.
- **Start Time** - Displays the time the query execution was started.
- **Execution Time** - Displays the time taken to execute the query.
- **Database Name** - Displays the name of the database.
- **Execution Status** - Displays the execution status of the query as **Success** or **Failure**.

Deleting the connection profile deletes the history. If the **SQL History** dialog box is closed, the query is not removed from the list.

You can perform the following operations in the **SQL History** dialog box:

- **Loading an SQL Query Into the SQL Terminal**
- **Loading Multiple SQL Queries Into the SQL Terminal**
- **Deleting an SQL Query**
- **Deleting all SQL Queries**
- **Pinning an SQL Query**
- **Unpinning an SQL Query**

Loading an SQL Query Into the SQL Terminal

Follow the steps to load the SQL query into the SQL terminal:

Step 1 Select the required query and click  .

The query is appended to the cursor position in the **SQL Terminal**.

----End

Loading Multiple SQL Queries Into the SQL Terminal

The **Load in SQL Terminal and close History** button loads selected queries into the **SQL Terminal** and closes the **SQL History** dialog box.

Follow the steps to load selected SQL queries into the SQL terminal:

Step 1 Select the required queries.

Step 2 Click  .

The queries are appended to the cursor position in the **SQL Terminal**.

----End

NOTE

If you continue the execution on error, then each statement in the terminal will be running as a scheduled job and runs one after the other. The execution status is updated on the console and job progress is displayed. When the time difference between Job Execution, Progress Bar Update and Console Update is very minimal, you will not be able to open the progress bar and stop the execution. In such scenarios you have to close the SQL terminal to the terminate execution.

Loading More Records

Regarding to load more data of result tab, you have to scroll down to bottom in order to load more data, which is inconvenient in some use cases. Currently, DS supports a loading more record button which makes it easier to trigger the loading more data action.

Follow the steps to load more records

Step 1 Select the required queries and click  .

List all the required records.

----End

Deleting an SQL Query

Follow the steps to delete a SQL query from the SQL history list:

Step 1 Select the required query and click  .

A confirmation pop up window is displayed.

Step 2 Click **OK**.

----End

Deleting all SQL Queries

Follow the steps to delete all SQL queries from the SQL History list:

Step 1 Click  .

A confirmation pop up window is displayed.

Step 2 Click **OK**.

----End

Pinning an SQL Query

You can pin queries that you do not want Data Studio to delete automatically from the **SQL History**. You can pin a maximum of 50 queries. Pinned queries are displayed at the top of the list. The value set in SQL history count does not affect the pinned queries. Refer to **Table 4-6** for additional information on SQL history count.

NOTE

The pinned queries appear on top once the **SQL History** window is closed and re-opened.

Follow the steps to pin a SQL query:

Step 1 Select the required SQL query and click  .

The **Pin Status** column displays the pinned status of the query.

----End

Unpinning an SQL Query

Follow the steps to unpin a SQL query:

Step 1 Select the required SQL query and click  .

The **Pin Status** column displays the unpinned status of the query.

----End

4.7.3 Opening and Saving SQL Scripts

Opening an SQL Script

Follow the steps to open an SQL script:

Step 1 Choose **File** > **Open** from the main menu. Alternatively, click **Open** on the toolbar or right-click the **SQL Terminal** and select **Open**.

If the SQL Terminal has existing content, then there will be an option to overwrite the existing content or append content to it.

Step 2 The Open dialog box is displayed.

Step 3 In **Open** dialog box, select the SQL file to import and click **Open**.

The selected SQL script is opened as a **File Terminal**.

The icons on the file terminal tab are different from those in the SQL terminal. When you move the mouse cursor over the source file, corresponding database connection will be displayed on **File Terminal**.

----End

NOTE

- The encoding type of the SQL file must match the encoding type specified in **Table 4-6**.
- Label of the file terminal will start with asterisk(*) if any of its content is edited. Dirty flag is removed once the file terminal is saved.
- File Terminals cannot be renamed. One terminal is always mapped to one Source Script File, but one script can be opened in multiple terminals.
- You can open SQL scripts only on SQL Terminals.

Data Studio allows you to save and open SQL scripts in the SQL Terminal. After saving the changes, SQL Terminal will be changed to a File Terminal.

Saving an SQL Script

The **Save** option saves the File Terminal content to the associated file.

Follow the steps to save an SQL script:

Step 1 Perform any of the following operations:

- Choose File > Save from the main menu.
- Press "Ctrl + S" to save the SQL terminal content. (This operation can be performed to save the file terminal content.)
- Click **Save** on the toolbar or right-click the SQL Terminal and select **Save**.

The **Data Studio Security Disclaimer** dialog box is displayed.

Step 2 Click **OK**.

Data Studio displays the status of the operation in the status bar.

NOTE

- The script is saved as an SQL file. Data Studio sets the read/write permission for the saved SQL file. To ensure security, you must set the read/write permissions for folders.
- When a change is made in a file and if that associated file is unavailable, it will trigger the **Save As** option.
  - In any case, if saving of the source file is failed due to some reasons, then user is prompted with the **Save As** option to save the content as a new source file. If you choose not to save (that is cancelling **Save As**), then File Terminal gets converted into an SQL Terminal.
  - The changes made to File Terminals are not **Auto Saved**.

----End

Saving an SQL Script in New File

The **Save As** option saves the terminal content to a new file.

Follow the steps to save an SQL script:

Step 1 Perform any of the following operations:

- Choose **File** > **Save as**.
- Press Ctrl+Alt+S. (This operation supports saving SQL terminal contents and file terminal contents.)

The **Data Studio Security Disclaimer** dialog box is displayed.

Step 2 Click **OK**.

The **Save As** dialog box is displayed.

Step 3 Select the location to save the script and click **Save**.

----End

NOTE

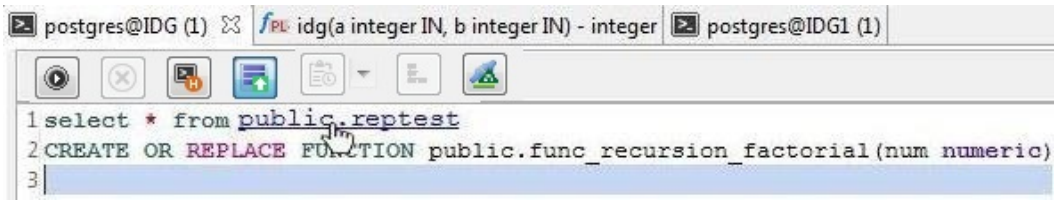
When there are unsaved changes in File Terminals, user will be given an option to save or cancel on graceful exit of data studio.

4.7.4 Viewing Table Properties, PL/SQL Functions/Procedures on the SQL Terminal Page

You can view table properties and functions/procedures in Data Studio.

Perform the following steps to view table properties:

Step 1 Press **Ctrl** and move the cursor over the table name.



Step 2 Click the highlighted table name. The attributes of the selected table are displayed in Data Studio.

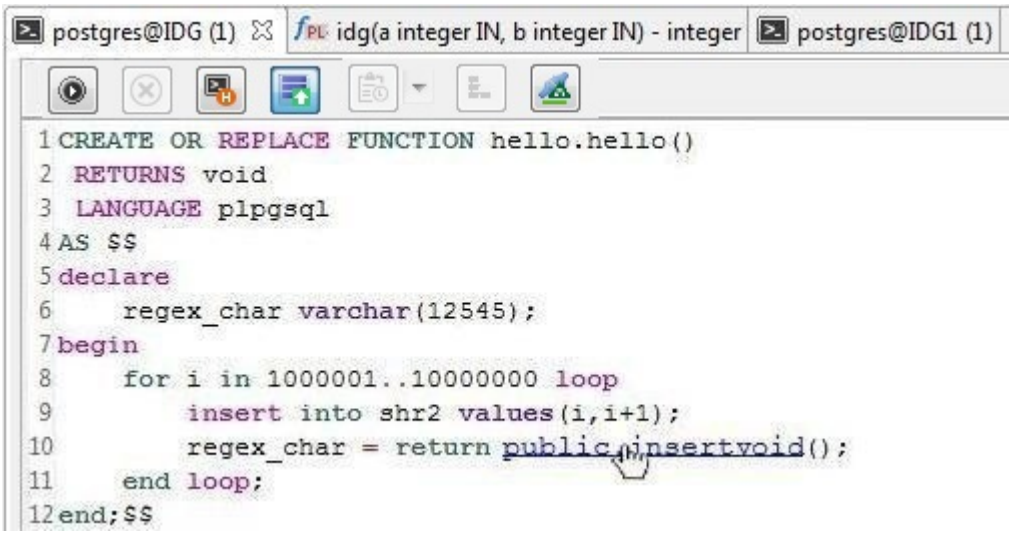
NOTE

All table properties are read-only.

----End

To view a function or procedure, perform the following steps:

Step 1 Press **Ctrl** and move the cursor over the function or procedure name.



Step 2 Click the highlighted function or procedure name. Data Studio displays the selected function or procedure on the **PL/SQL Viewer** page.

----End

Perform the following steps to check view properties:

Step 1 Press **Ctrl** and move the cursor over the view name.

Step 2 Click the highlighted view name. Data Studio displays the properties of the selected view.

----End

Saving the File Before Closing Data Studio

Before you exit, Data Studio prompts you to save the editing on the terminal.

Perform the following steps:

Step 1 Click **Close** and the **Exit Application** dialog box is displayed.

Step 2 Click **Graceful Exit**.

In the **Saving File Terminal** dialog box, select the file terminal to be saved.

Step 3 Select the file terminal to be saved.

Step 4 Click **OK**.

----End

NOTE

If you select **Force Exit**, the **Saving File Terminal** dialog box will not be displayed.

4.7.5 Terminating an Ongoing SQL Query

You can terminate an ongoing SQL query on the **SQL Terminal** tab page of Data Studio.

To terminate an ongoing SQL query, perform the following steps:

Step 1 Execute a SQL query on the **SQL Terminal** tab page.

Step 2 Click  or press **Shift**+**Esc**.

You can also choose **Run** > **Cancel**, or right-click in the code area and choose **Cancel**, or click **Cancel** on the **Progress View** tab page.

----End

After you terminate an ongoing query, the SQL statement being executed is stopped.

Database changes made by the canceled query are rolled back and the following queries will not be executed.

A query cannot be canceled and the **Result** tab displays the result when:

1. The query has been executed and the result is being generated.
2. The result of the executed query is being transferred from the server to the Data Studio client.

A query cannot be canceled when you are viewing the execution plan of a query. For details, see **Viewing the Execution Plan and Costs**.

The **Messages** tab displays a message indicating that the query has been cancelled.

NOTE

The **Cancel** option is enabled only when a query is being executed.

4.7.6 Formatting of SQL Queries

Data Studio supports formatting and highlighting of SQL queries and PL/SQL statements.

PL/SQL Formatting

Follow the steps to format PL/SQL statements:

Step 1 Select the PL/SQL statements to be formatted.

Step 2 Click  on the toolbar to format the query.

Alternatively, use the key combination **Ctrl+Shift+F** or choose **Edit** > **Format** from the main menu.

The PL/SQL statements are formatted.

----End

SQL Formatting

Data Studio supports formatting of simple SQL SELECT, INSERT, UPDATE, DELETE statements which are syntactically correct. The following are some of the statements for which formatting is supported:

1. The SELECT statement must be made of the following clauses:
   - Target list
   - FROM clause (including JOIN)
   - WHERE clause
   - GROUP BY clause
   - HAVING clause
   - ORDER BY clause
   - Common table expression

   SELECT statement without SET operations like UNION, UNION ALL, MINUS, INTERSECT and so on

   SELECT statements without sub-queries
2. The INSERT statement is made of the following clauses only:
   - Insert Into Table name
   - Values
   - Values Column List
   - RETURNING
3. The UPDATE statement is made of the following clauses only:
   - Update Table name
   - SET
   - FROM clause (including JOIN)
   - WHERE clause
   - RETURNING
4. The DELETE statement is made of the following clauses only:
   - Delete From Table name
   - USING clause (including JOIN)
   - WHERE clause

- RETURNING

Follow the steps below to format SQL queries:

Step 1 Select the SQL query statements to be formatted.

Step 2 Click  on the toolbar to format the query.

Alternatively, use the key combination **Ctrl+Shift+F** or choose **Edit** > **Format** from the main menu.

The query is formatted.

The following table describes the query formatting rules.

Table 4-29 Query formatting rules

Statement	Clause	Formatting Rule
SELECT	SELECT list	Line break before first column
		Indent column list
	FROM	Line break before FROM
		Line break after FROM
		Indent FROM list
		Stack FROM list
	Line break before JOIN	Line break after JOIN
		Line break after JOIN
		Line break before ON
		Line break after ON
		Indent table after JOIN
		Indent ON condition
	WHERE	Line break before WHERE
		Line break after WHERE
		Indent WHERE condition
		Place WHERE condition on single line
	GROUP BY	Line break before GROUP
		Line break before GROUP BY expression
		Indent column list
		Stack column list
	HAVING	Line break before HAVING

Statement	Clause	Formatting Rule
		Line break after HAVING
		Indent HAVING condition
	ORDER BY	Line break before ORDER
		Line break after BY
		Indent column list
		Stack column list
	CTE	Indent subquery braces
		Each CTE in a new line
INSERT	INSERT INFO	Line break before opening brace
		Line break after opening brace
		Line break before closing brace
		Indent column list braces
		Indent column list
		Line break before VALUES
		Stack column list
		Line break before VALUES
		Line break before opening brace
		Line break after opening brace
		Line break before closing brace
		Indent VALUES expressions list braces
		Indent VALUES expressions list
		Stack VALUES expressions list
	DEFAULT	Line break before DEFAULT
		Indent DEFAULT keyword
	CTE	Each CTE in a new line
	RETURNING	Line break before RETURNING
		Line break after RETURNING
		Indent RETURNING column list
		Place RETURNING column List on single line

Statement	Clause	Formatting Rule
UPDATE	UPDATE Table	Line break before table
		Indent table
	SET Clause	Line break before SET
		Indent column assignments list
		Indent column assignments list
	FROM CLAUSE	Line break before FROM
		Line break after FROM
		Indent FROM list
		Stack FROM list
	JOIN CLAUSE(FROM CLAUSE)	Line break before JOIN
		Line break after JOIN
		Line break before ON
		Line break after ON
		Indent table after JOIN
		Indent ON condition
	WHERE CLAUSE	Line break before WHERE
		Line break after WHERE
		Indent WHERE condition
		Place WHERE condition on single line
	CTE	Each CTE in a new line
	RETURNING	Line break before RETURNING
		Line break after RETURNING
DELETE	USING CLAUSE	Indent RETURNING column list
		Line break before FROM
		Line break after FROM
		Indent USING list
		Stack FROM list
	JOIN CLAUSE	Line break before JOIN
		Line break after JOIN

Statement	Clause	Formatting Rule
		Line break before ON
		Line break after ON
		Indent table after JOIN
		Indent ON condition List
	WHERE CLAUSE	Line break before WHERE
		Line break after WHERE
		Indent WHERE condition
		Stack WHERE condition list
	CTE	Each CTE in a new line
	RETURNING	Line break before RETURNING
		Line break after RETURNING
		Indent RETURNING column list

----End

Data Studio supports automatic highlighting of the following punctuation mark's pair when cursor is placed before or after the punctuation mark or the punctuation mark is selected.

- Brackets - ( )
- Square brackets - [ ]
- Braces - { }
- Single-quoted string literals - ' '
- Double-quoted string literals - " "

Follow the steps below to change case for SQL queries and PL/SQL statements:

Method 1

Step 1 Select the text, and choose **Edit** > **Upper Case/Lower Case**.

The text changes to the case selected.

----End

Method 2:

Step 1 Select the text, and choose AA or aa from the toolbar.

The text changes to the case selected.

----End

Method 3:

Step 1 Select the text, and press Ctrl+Shift+U to change to the upper case or Ctrl+Shift+L to change to the lower case.

The text changes to the case selected.

----End

SQL Highlighting

Keywords are highlighted automatically when you enter them (according to the default color scheme) as shown below:

The following figure shows the default color scheme for the specified type of syntax:

Syntax Category	Color Value	HEX Value	Shade
DEFAULT	RGB(0,0,0)	000000	
UNRESERVED_KEYWORD	RGB(198,0,134)	C60086	
TYPE	RGB(64,0,200)	4000C8	
PREDICATES	RGB(224,5,11)	E0050B	
RESERVED	RGB(35,90,80)	235A50	
CONSTANTS	RGB(35,90,80)	235A50	
SINGLE_LINE_COMMENT	RGB(64,128,128)	408080	
STRING	RGB(0,0,255)	0000FF	

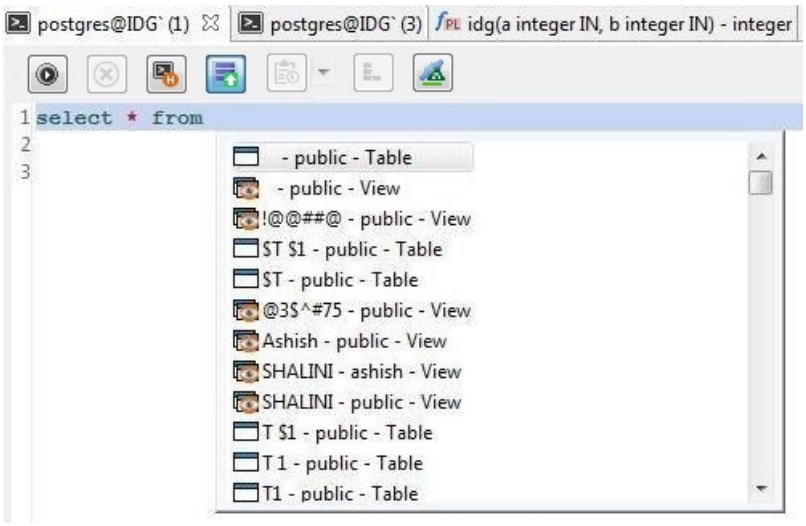
You can also customize SQL highlighting schemes for specific types of syntax. For details, see **Table 4-6**.

4.7.7 Selecting a DB Object in the SQL Terminal

Data Studio suggests a list of possible schema names, table names and column names, and views in the SQL Terminal.

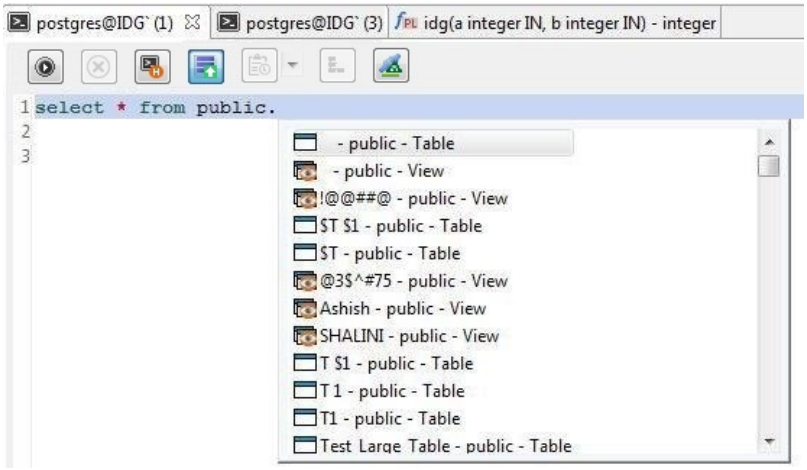
Follow the steps below to select a DB object:

Step 1 Press **Ctrl** and **Space** and enter the required parent DB object name. The DB objects list is refined as you continue typing the DB object name. The DB objects list displays all DB objects of the database connected to the SQL Terminal.



Step 2 To select the parent DB object, use the **Up** or **Down** arrow keys and press **Enter** on the keyboard, or double-click the parent DB object.

Step 3 Press **.** to list all child DB objects.



Step 4 To select the child DB object, use the **Up** or **Down** arrow keys and press **Enter** on the keyboard, or double-click the child DB object.

On selection, the child DB object will be appended to the parent DB object (with a period '.').

NOTE

- Auto-suggest also works on keywords, data types, schema names, table names, views, and table name aliases in the same way as shown above for all schema objects that you have access.

  Following is a sample query with alias objects:

  ```
  SELECT
    table_alias.<auto-suggest>
  FROM test.t1 AS table_alias
    WHERE
      table_alias.<auto-suggest> = 5
  GROUP BY table_alias.<auto-suggest>
  HAVING table_alias.<auto-suggest> = 5
  ORDER BY table alias.<auto-suggest>
  ```

- A loading message may be displayed on the **SQL Terminal** in the following scenarios:
  - The object is not loaded due to the value mentioned in the **Load Limit** field. For details, see **Table 4-6**.
  - The objects that are already added to **Exclude** will not be loaded. For details, see **Table 4-6**.
  - There is a delay in fetching the object from the server.
- If there are objects with the same name in different case, then auto-suggest will display child objects of both parent objects.

  Example: If there are two schemas with the name **public** and **PUBLIC**, then all child objects for both these schemas will be displayed.

----End

4.7.8 Viewing the Execution Plan and Costs

The execution plan shows how the table referenced by the SQL statement will be scanned (sequential scan or index scan).

The SQL statement execution cost indicates the duration for executing a statement (measured in cost units that are arbitrary, but conventionally mean disk page fetches).

Follow the steps below to view the plan and cost for a required SQL query:

Step 1 Enter the query or use an existing query in the **SQL Terminal** and click  on the SQL Terminal toolbar to view explain plan.

To view explain plan with analyze, click the drop-down from  , select **Include Analyze**, and click  .

The **Execution Plan** opens in tree view format as a new tab at the bottom by default. The display mode has a tree shape and text style.

NOTE

The data shown in tree explain plan and visual explain may vary, since the execution parameters considered by both are not the same.

Following are the parameters selected for explain plan with/without analyze and the columns displayed:

Table 4-30 Explain plan options

Explain Plan Type	Parameter	Column
Include Analyze deselected (default setting)	Verbose, Costs	Node type, startup cost, total cost, rows, width, additional Info
Include Analyze selected	Analyze, Verbose, Costs, Buffers, Timing	Node type, startup cost, total cost, rows, width, Actual startup time, Actual total time, Actual Rows, Actual loops, Additional Info

The **Additional Info** column contains predicate information (filter predicate and hash condition), distribution key, output information, and node type.

In **Tree Format**, nodes are categorized into 16 types. Each node is indicated with an icon of the corresponding type. The following table lists the node types and icons.

Table 4-31 Node types and icons

Node Category	Icon
Aggregate	
Group Aggregate	
Function	
Hash	
Hash Join	
Nested Loop	
Nested Loop Join	
Modify Table	
Partition Iterator	
Row Adapter	
Seq Scan on	

Node Category	Icon
Set Operator	
Sort	
Stream	
Union	
Unknown	

You can hover over highlighted cells to identify the heaviest, costliest, and slowest nodes. Cells can only be highlighted in **Tree Format**.

If multiple queries are selected, the explain plan with/without **Include Analyze** selected is displayed only for the last query.

Each execution plan generated from executing a query is opened on a new tab page.

If the connection is lost during execution but the database remains connected in **Object Browser**, the **Connection Error** dialog box is displayed with the following options:

- **OK**: Connects to the database again to obtain the execution plan and cost.
- **Cancel**: Disconnects the database from **Object Browser**.

Toolbar menu in the **Execution Plan** window:

Toolbar Name	Icon	Description
Tree Format		This icon is used view explain plan in tree format.
Text Format		This icon is used to display explain plan in text format.
Copy		This icon is used to copy selected data from the **Result** tab to clipboard. The shortcut key is **Ctrl** +**C**.
Commit		This icon is used to save the explain plan in text format.

For information about refresh, SQL preview, and search bar, see **Execute SQL Queries**.

After you click **Refresh**, the **EXPLAIN ANALYZE** query is executed again, and the execution plan is updated.

The result is displayed on the **Messages** tab.

----End

4.7.9 Viewing the Query Execution Plan and Cost Graphically

Visual Explain plan displays a graphical representation of the SQL query using information from the extended JSON format. This function improves the query and server performance by refining the query. It also analyzes the query path of the database and finds the heaviest, costliest and slowest node.

The graphical execution plan shows how the table(s) referenced by the SQL statement will be scanned (plain sequential scan and index scan).

The SQL statement execution cost is the estimate at how long it will take to run the statement (measured in cost units that are arbitrary, but conventionally mean disk page fetches).

Costliest: Highest **Self Cost** plan node.

Heaviest: Maximum number of rows output by a plan node is considered heaviest node.

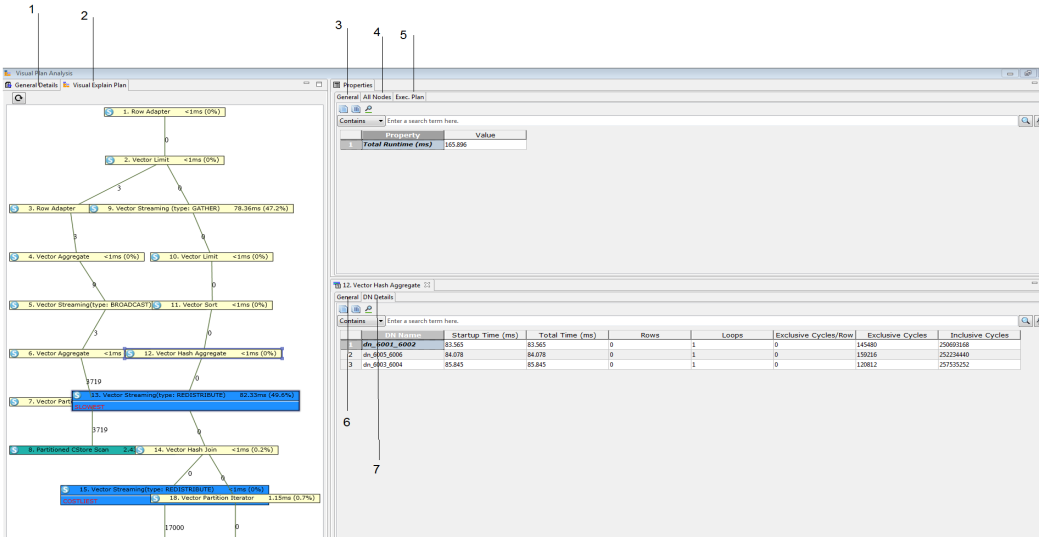
Slowest: Highest execution time by a plan node.

Follow the steps to view the graphical representation of plan and cost for a required SQL query:

Step 1 Enter the query or use an existing query in the SQL Terminal and click  on the SQL Terminal toolbar.

Visual Plan Analysis window is displayed.

Refer to **Viewing the Execution Plan and Costs** for information on reconnect option in case connection is lost while retrieving the execution plan and cost.



- 1 - **General Detail tab**: This tab displays the query.
- 2 - **Visual Explain Plan tab**: This tab displays a graphical representation of all nodes like execution time, costliest, heaviest, and slowest node. Click each node to view the node details.
- 3 - **Properties - General** tab: Provides the execution time of the query in ms.
- 4 - **Properties - All Nodes** tab: Provides all node information.

Column Name	Description
Node Name	Name of the node
Analysis	Node analysis information
RowsOutput	Number of rows output by the plan node
RowsOutput Deviation (%)	Deviation % between estimated rows output and actual rows output by the plan node
Execution Time (ms)	Execution time taken by the plan node
Contribution (%)	Percentage of the execution time taken by plan node against the overall query execution time.
Self Cost	**Total Cost** of the plan node - Total Cost of all child nodes
Total Cost	Total cost of the plan node

- 5 - **Properties - Exec. Plan** tab - Provides the execution information of all nodes.

Column Name	Description
Node name	Node
Entity Name	Name of the object
Cost	Execution time taken by the plan node
Rows	Number of rows output by the plan node
Loops	Number of loops of execution performed by each node
Width	The estimated average width of rows output by the plan node in bytes
Actual Rows	Number of estimated rows output by the plan node

Column Name	Description
Actual Time	Actual execution time taken by the plan node

- 6 - **Plan Node - General** tab - Provides the node information for each node.

Row Name	Description
Output	Provides the column information returned by the plan node.
Analysis	Provides analysis of the plan node like costliest, slowest, and heaviest.
RowsOutput Deviation (%)	Deviation % between estimated rows output and actual rows output by the plan node
Row Width (bytes)	The estimated average width of rows output by the plan node in bytes
Plan Output Rows	Number of rows output by the plan node
Actual Output Rows	Number of estimated rows output by the plan node
Actual Startup Time	The actual execution time taken by the plan node to output the first record
Actual Total Time	Actual execution time taken by the plan node
Actual Loops	Number of iterations performed for the node
Startup Cost	The execution time taken by the plan node to output the first record
Total Cost	Execution time taken by the plan node
Is Column Store	This field represents the orientation of the table (column or row store)
Shared Hit Blocks	Number of shared blocks hit in buffer
Shared Read Blocks	Number of shared blocks read from buffer
Shared Dirtied Blocks	Number of shared blocks dirtied in buffer

Row Name	Description
Shared Written Blocks	Number of shared blocks written in buffer
Local Hit Blocks	Number of local blocks hit in buffer
Local Read Blocks	Number of local blocks read from buffer
Local Dirtied Blocks	Number of local blocks dirtied in buffer
Local Written Blocks	Number of local blocks written in buffer
Temp Read Blocks	Number of temporary blocks read in buffer
Temp Written Blocks	Number of temporary blocks written in buffer
I/O Read Time (ms)	Time taken for making any I/O read operation for the node
I/O Write Time (ms)	Time taken for making any I/O write operation for the node
Node Type	Represents the type of node
Parent Relationship	Represents the relationship with the parent node
Inner Node Name	Child node name
Node/s	No description needed for this field, this will be removed from properties

Based on the plan node type additional information may display. Few examples:

Plan Node	Additional Information
Partitioned CStore Scan	Table Name, Table Alias, Schema Name
Vector Sort	Sort keys
Vector Hash Aggregate	Group By Key
Vector Has Join	Join Type, Hash Condition
Vector Streaming	Distribution key, Spawn On

- 7 - **Plan Node - DN Details** tab - Provides detailed data node information for each node. **DN Details** are available only if data is being collected from data node.

  Refer to **Viewing Table Data** for description on copy and search toolbar options.

----End

4.7.10 Working with SQL Terminals

In **SQL Terminal**, you can

- **Automatically Commit a Transaction**
- **Execute SQL Queries**
- **Multi-Column Sort**
- **Backuping Unsaved Queries/Functions/Procedures**
- **Error Locator**
- **Search in PL/SQL Viewer or SQL Terminal**
- **Go to Line in PL/SQL Viewer or SQL Terminal**
- **Comment/Uncomment**
- **Indent/Un-indent Lines**
- **Insert Space**
- **Execute Multiple Functions/Procedures or Queries**
- **Rename SQL Terminal**
- **Using Templates**

Auto Commit

The **Auto Commit** option can be switched on or off based on the **Preferences** settings. Refer to **Table 4-6** for more details.

- If **Auto Commit** option is enabled, **Commit** and **Rollback** buttons are disabled. Transactions are committed automatically.
- If **Auto Commit** option is disabled, **Commit** and **Rollback** buttons are enabled. You can use the buttons manually to commit or revert the changes.

  NOTE

  - For OLAP, the server will open a transaction for all SQL statements, such as **select**, **explain select**, and **set parameter**.
  - For OLTP, the server will open a transaction for only DML statements, such as **INSERT**, **UPDATE**, and **DELETE**.

Reuse Connection

It enables the user to choose the same SQL terminal connection or new connection for the result set. The selection affects the record visibility due to the isolation levels defined in the database server.

- When **Reuse Connection** is **ON**, terminal connection will be used for data manipulation and refresh of the result window.

For some data base temp tables that are created or used by the terminal can be edited in the result window.

- When **Reuse Connection** is **OFF**, new connection will be used for data manipulation and refresh of the result window.

For some databases, the temporary tables can be edited in the **Result** tab.

 : displayed when **Reuse Connection** is set to **ON**

 : displayed when **Reuse Connection** is set to **OFF**

 : displayed when **Reuse Connection** is disabled

Perform the following steps to set **Reuse Connection** to **OFF**:

Step 1 Click  on the **SQL Terminal** toolbar.

Reuse Connection is disabled for the terminal. 

NOTE

- The **Reuse Connection** function is enabled by default. You can disable it as required. If you enable **Auto Commit**, the system automatically enables the **Reuse Connection** function.
- If you disable **Auto Commit**, the system automatically disables the **Reuse Connection** function. However, this function is still displayed as **Enabled** on the GUI, and the status cannot be modified.

----End

Refer to **Table 4-6** for more details about **Auto Commit** and **Reuse Connection**.

Execute SQL Queries

Perform the following steps to execute a function/procedure or SQL query.

Enter a function/procedure(s) or SQL query(s) in the **SQL Terminal** tab and click  in the **SQL Terminal** tab, or press **Ctrl+Enter**, or choose **Run > Compile/ Execute Statement** from the main menu.

Alternatively, you can right-click in the **SQL Terminal** tab and select **Execute Statement**.

NOTE

You can check the status bar to view the status of a query being executed.

The **Result** tab displays the results after executing the function/procedure(s) or SQL queries along with the query statement executed.

If the connection is lost during execution and the database is still connected in Object Browser, then **Connection Error** dialog box is displayed:

- **Reconnect** - The connection is reestablished.

- **Reconnect and Execute** - With Auto commit on, execution will continue from failure statement. With Auto commit off, execution will continue from position of cursor.
- **Cancel** - Disconnects database in Object Browser.

Failure to reconnect after three attempts will disconnect the database in Object Browser. Connect to the database in Object Browser and retry execution.

NOTE

- For long running queries, result set can be edited only after the complete results are fetched.
- Editing of query results are only allowed in following scenarios:
  - Selected targets are from a single table
  - Either select all columns or subset of columns [No aliases, aggregate functions, expressions on columns]
  - All WHERE condition
  - All ORDER BY clause
  - On regular, partition, and temporary tables.
- Committing an empty row assigns Null to all columns.
- Only result set of queries executed on tables available in Object Browser is editable.
- Editing of query results is allowed only for queries executed in SQL Terminal.

The column width definition can be set using **Settings** > **Preferences** option. Refer to **Table 4-6** to set this parameter.

Column Reorder

Column reordering can be performed by clicking and dragging the selected column header to the desired position.

Multi-Column Sort

This feature allows the user to sort table data of some pages by multiple columns. In addition, you can set the priority of columns for sorting.

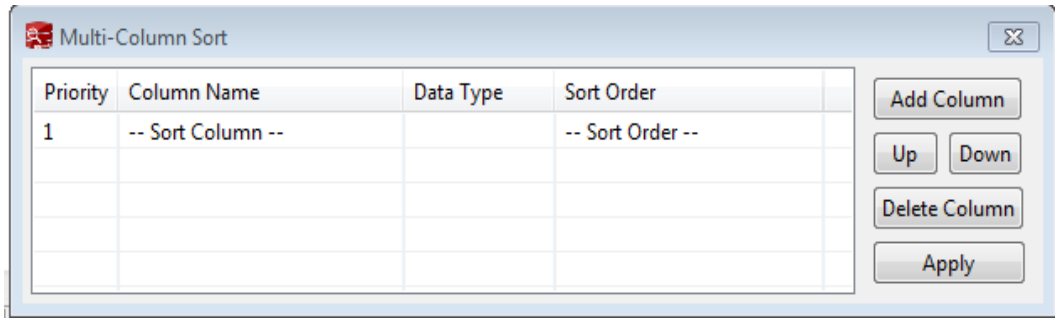
The feature is available for the following pages:

- Result Set Tab
- Edit Table Data Window
- View Table Data Window
- Batch Drop Result Window

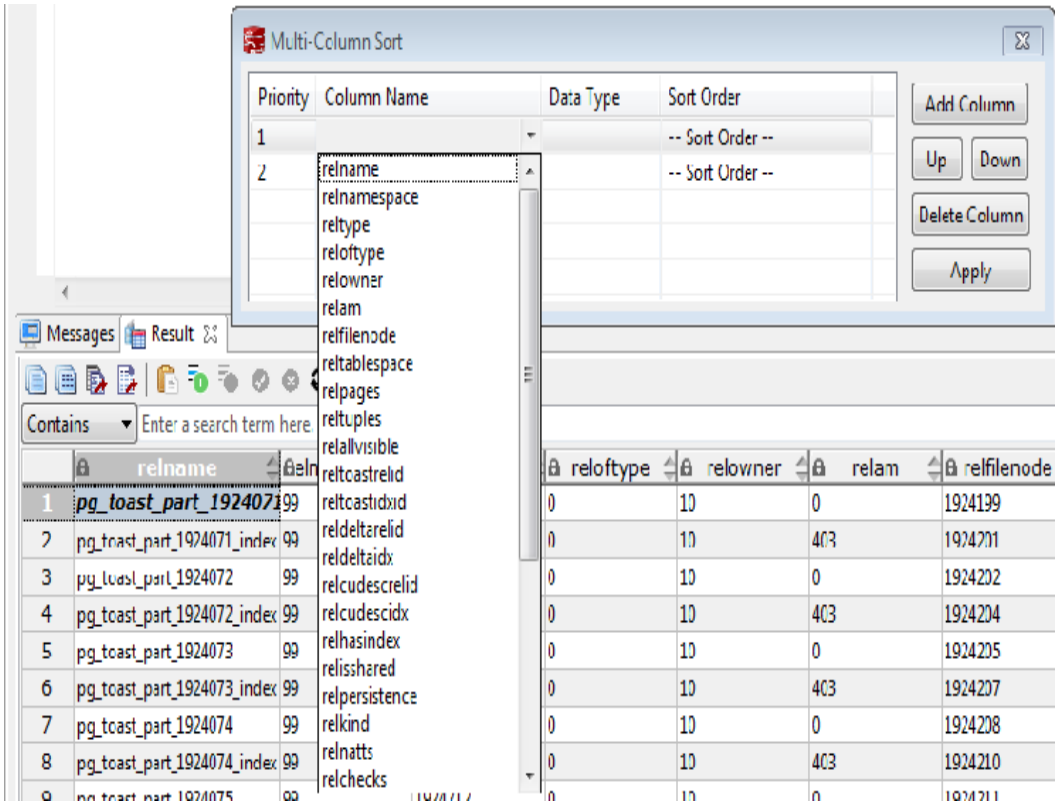
Follow the steps below to access Multi-column sort:

Step 1 Click  in the toolbar.

Multi-Column Sort pop-up is displayed.



Step 2 Click **Add Column**. Choose the column to be sorted from the drop-down list.



Step 3 Select the required sort order.

Step 4 Click **Apply**.

----End

Multi-sort pop up has following elements:

Table 4-32 Elements of multi-column pop-up:

Attribute Name	UI Element Type	Description/Action
Priority	Read only text field	Shows column priority in multi sort.
Column Name	Combo field having all column names of the table as its value set	Column name of the column added for sorting.

Attribute Name	UI Element Type	Description/Action
Data Type	Read only text field	Shows data type of the column selected.
Sort Order	Combo field having values {sort_ascending, sort_descending}	Sort order of the column.
Add Column	Button	Adds new row to multi-sort table.
Delete Column	Button	Deletes selected column from multi-sort table.
Up	Button	Moves selected column up by 1 step, thus changing sort priority.
Down	Button	Moves selected column down by 1 step, this changing sort priority.
Apply	Button	Apply prepared sort configuration.

NOTE

Except following data types, all the other data types will be sorted by their string value (Alphabetical order):

TINYINT, SMALLINT, INTEGER, BIGINT, FLOAT, REAL, DOUBLE, NUMERIC, BIT, BOOLEAN, DATE, TIME, TIME_WITH_TIMEZONE, TIMESTAMP, TIMESTAMP_WITH_TIMEZONE.

Elements of Multi-Column Pop-up:

Table 4-33 Icons of multi-column Pop-up

Icon	Description	Action
	Not Sorted	This icon in column header indicates that the column is not sorted. You can click this icon to sort the column in ascending order. Alternatively, use **Alt +Click** to select the column header.

Icon	Description	Action
	Ascending Sort	This icon in column header indicates that the column is sorted in ascending order. If you click this icon, the column will be sorted in descending order. Alternatively, use **Alt +Click** to select the column header.
	Descending Sort	This icon in column header indicates that the column is sorted in descending order. You can click this icon to cancel the column sorting. Alternatively, use **Alt +Click** to select the column header.

Icons for the sort priority are as follows:

 : Icon with three dots indicates the highest priority.

 : Icon with two dots indicates the second highest priority.

 : Icon with one dot indicates the lowest priority.

Table 4-34 Toolbar Menus

Toolbar Name	Description
Copy	This button is used to copy selected content from result window to clipboard. Shortcut key - **Ctrl+C**.
Advanced Copy	This button is used to copy content from result window to clipboard. Results can be copied to include column header. Refer to **Table 4-6** to set this preference. The shortcut key is **Ctrl+Shift+C**.

Toolbar Name	Description
Export all data	This button is used to export all data in Excel (xlsx/xls), CSV, text, or binary format. For details, see **Exporting Table Data**. **NOTE** • The columns involved in the query are automatically populated in the **Selected Columns** area. The **Available Columns** area is empty. • To export the query results, the query is re-executed using a new connection. The exported results may differ from the data in the results tab. • Disabled for explain/analyze queries. To export explain/analyze queries use the **Export current page data** option.
Export current page data	This button is used to export current page data in Excel (xlsx/xls) or CSV format.
Paste	This button is used to paste copied information.
Add	This button is used to add a row to the result set.
Delete	This button is used to delete a row from the result set.
Save	This button is used to save the changes made in the result set.
Rollback	This button is used to roll back the changes made to the result set.
Refresh	This button is used to refresh information in the result set. If multiple result sets are open for the same table, then changes made to one result set will reflect on the other post refresh. Similarly if the same table is edited, then the result set will be updated post refresh.
Clear Unique Key selection	This button is used to clear the previous unique key selection.
Show/Hide Query bar	This button is used to display/hide the query executed for that particular result set. This is a toggle button.
Show/Hide Search bar	This button is used to display/hide the search text field. This is a toggle button.

Toolbar Name	Description
Encoding	Whether you can configure this field depends on the settings in **Preferences** > **Result Management** > **Query Results** > **Result Data Encoding**. In this drop-down list, you can select the appropriate code to view the data accurately. By default, the text is encoded using UTF-8. Refer to **Table 4-6** to set the encoding preference. **NOTE** Data editing except for data insertion is restricted once the default encoding is modified.
Multi Sort	This button brings up multi-sort pop up.
Clear Sort	This button is used to reset all the sorted column.

Icons in Search field:

Icon Name	Icon	Description
Search		This icon is used to search the result set based on the criteria defined. The text is case-insensitive.
Clear Search Text		This icon is used to clear the search text entered in the search field.

Right-click options in the **Result** window:

Option	Description
Close	Closes only the active result window.
Close Others	Closes all other result windows except for the active result window.
Close Tabs to the Right	Closes only the right active result window.
Close All	Closes all result windows including the active result window.
Detach	Detach from current active result window.

Status information displayed in the **Result** window:

- **Query Submit Time** - Provides the query submitted time.
- Number of rows fetched with execution time is displayed. The default number of rows is displayed. If there are additional rows to be fetched, then it will be

denoted with the word "more". You can scroll to the bottom of the table to fetch and display all rows.

NOTICE

When viewing table data, Data Studio automatically adjusts the column widths for a good table view. Users can resize the columns as needed. If the text length exceeds the column width and you adjust the column width, Data Studio may fail to respond.

NOTE

- Each time a query is run in **SQL Terminal** tab, a new result window opens. To view the results in the new window, you must select the newly opened window.
- Set the **focusOnFirstResult** configuration parameter to **false** to automatically set focus to the newly opened **Result** window. For details, see **Configuring Data Studio**.
- Each row, column and selected cells can be copied from the result set.
- Export all data operation will be successful even after the connection is removed.
- If the text of a column contains spaces, word wrapping is applied to fit the column width. Word wrapping is not applied to columns without spaces.
- Select part of cell content and press **Ctrl+C** or click  to copy selected text from a cell.
- The size of the column is determined by the maximum content length column.
- You can save preference to define:
  - Number of records to be fetched
  - Column width
  - Copy option from result set

    For details, see **Table 4-6**.
- If any column of resultset tab has Lock Image icon in it, then values are not editable.

Backuping Unsaved Queries/Functions/Procedures

Data Studio creates back up of unsaved data in SQL Terminal and PL/SQL Viewer periodically based on the time interval defined in the **Preferences** tab. The data can be encrypted and saved based on **Preference** settings. Refer to **Table 4-6** to turn on/off backup, define time interval to save the data, and encrypt the saved data.

Unsaved changes of each SQL Terminal/PL/SQL Viewer are taken as backup and stored in ***DataStudio|UserData|<user name>|Autosave folder.*** Backup files saved before unexpected shutdown of Data Studio will be available at next login.

In case there is unsaved data in SQL Terminal/PL/SQL Viewer, during graceful exit, Data Studio will wait for backup to complete before closing.

Error Locator

During execution of query/function/procedure in case of an error the error locator message is displayed.

Yes - Click **Yes** to continue with the execution.

No - Click **No** to stop the execution.

You can select **Do not display other errors that occur during the execution** to hide the error messages and proceed with the current SQL query.

Line number and position of error displays in **Messages** tab. The corresponding line number is marked with  icon along with red underline at the position of the error in the Terminal/PL/SQL Viewer. Hovering over  displays the error message. For details about why the line number does not match the error detail, see **FAQs**.

NOTE

If the query/function/procedure is modified while execution is in progress, then error locator may not display the correct line and position number.

Search in PL/SQL Viewer or SQL Terminal

Follow the steps below to search in PL/SQL Viewer or SQL Terminal:

F3 key is used to search next word and **Shift+F3** key is used to search previous word. These shortcut keys will be enabled only after **Ctrl+F** is used to search a text. These keys will be active with the current search word until a new word is searched. The value searched using **Ctrl+F** and **F3/Shift+F3** will be applicable only for the current instance.

Step 1 Choose **Edit > Find and Replace** from the main menu.

Alternatively press **Ctrl+F**.

Find and Replace dialog box is displayed.

Step 2 Enter the text to be searched for in the **Find what** field, and click the **Find Next** button.

The desired text is highlighted.

F3 and **Shift+F3** key will now be enabled for forward and backward search.

NOTE

Select **Wrap around** option to continue the search after reaching the last line in the SQL queries or PL/SQL statements.

----End

Go to Line in PL/SQL Viewer or SQL Terminal

Go to line option is used to skip to a specific line in the terminal.

Follow the steps below to go to a line in PL/SQL Viewer or SQL Terminal:

Step 1 Choose **Edit** > **Go To Line** from the main menu or press **Ctrl+G**.

The **Go To Line** dialog box is displayed.

Step 2 Enter the desired number in the **Enter the line number** field, and then click the **OK** button.

The cursor moves to the beginning of the line entered in the **Go to Line** dialog box.

NOTE

The following inputs are invalid for the **Enter line number** field:

- Non-numeric value
- Special characters
- Line number entered does not exist in the editor.
- More than 10 digits is entered.

----End

Comment/Uncomment

Comment/uncomment option is used to comment/uncomment lines or block of lines.

Follow the steps below to comment/uncomment lines in PL/SQL Viewer or SQL Terminal:

Step 1 Select the lines to comment/uncomment.

Step 2 Choose **Edit** option. Choose **Comment/Uncomment Lines** from the main menu.

Alternatively, press **Ctrl+/** or right-click a line and select **Comment/Uncomment Lines**.

----End

Follow the steps below to comment/uncomment block of lines/content in PL/SQL Viewer or SQL Terminal:

Step 1 Select the lines/content to comment/uncomment.

Step 2 Choose **Edit** option. Choose **Comment/Uncomment Block** from the main menu.

Alternatively, press **Ctrl+Shift+/** or right-click a line or the entire block and select **Comment/Uncomment Block**.

----End

Indent/Un-indent Lines

The indent/un-indent option is used to shift lines as per the indent size defined in the **Preferences** tab.

Follow the steps to indent lines in PL/SQL Viewer or SQL Terminal:

Step 1 Select the lines to indent.

Step 2 Press **Tab** or click  .

Shift the selected line as per the indent size defined in the **Preferences** tab. For details about modifying the indent size, see **Table 4-6**.

----End

Follow the steps to un-indent lines in PL/SQL Viewer or SQL Terminal:

Step 1 Select the lines to un-indent.

Step 2 Press **Shift+Tab** or click  .

Move the selected lines according to the indent size defined in **Preferences**. For details about modifying the indent size, see **Table 4-6**.

NOTE

Only selected lines that have available tab space will be un-indented. For example, if multiple lines are selected, and one of the selected lines starts at position 1, then pressing **Shift+Tab** will un-indent all the lines except for the one starting at position 1.

----End

Insert Space

The **Insert Space** option is used to replace a tab with spaces based on the indent size defined in the **Preferences** tab.

Follow the steps below to replace a tab with spaces in PL/SQL Viewer or SQL Terminal:

Step 1 Select the lines to replace tab with spaces.

Step 2 Press **Tab** or **Shift+Tab**.

Replaces the tab with spaces as per the indent size defined in the **Preferences** tab. For details about modifying the indent size, see **Table 4-6**.

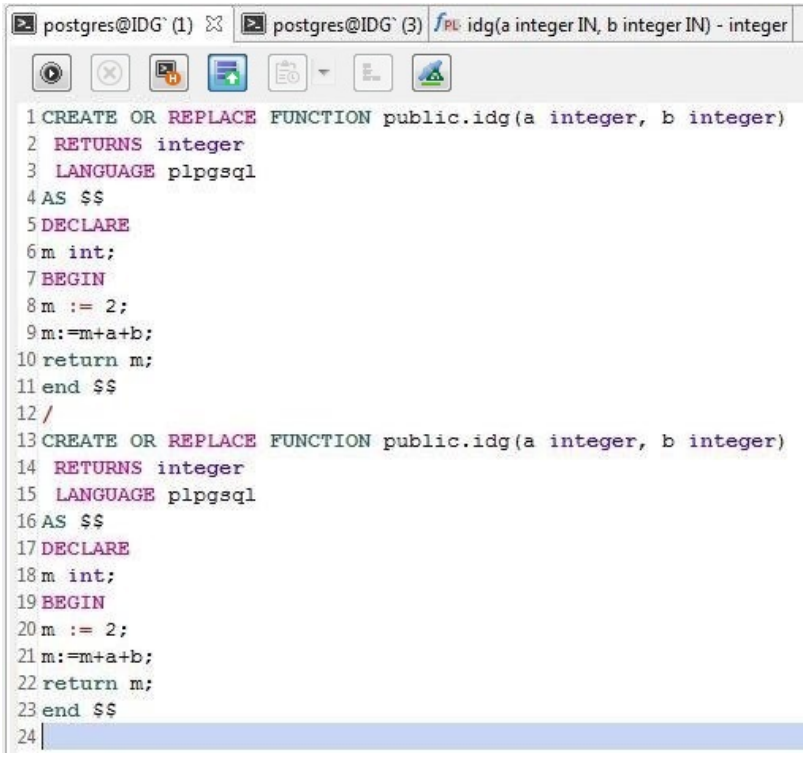
----End

Execute Multiple Functions/Procedures or Queries

Follow the steps below to execute multiple functions/procedures:

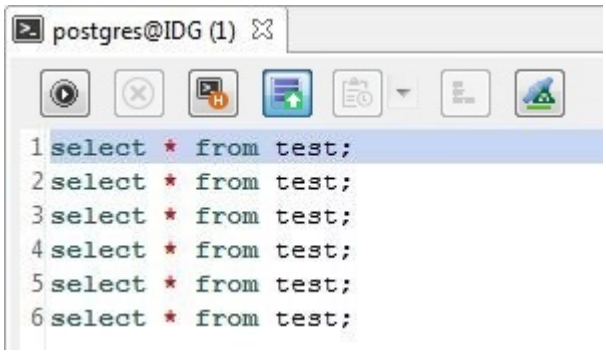
Insert a forward slash (/) in a new line after the function/procedure in the **SQL Terminal**.

Add the new function/procedure in the next line.



Follow the steps below to execute multiple SQL queries:

Step 1 Enter multiple SQL queries in the **SQL Terminal** tab as follows:



Step 2 Click in the **SQL Terminal** tab, or press **Ctrl+Enter**, or choose **Run > Compile/Execute Statement** from the main menu.

NOTE

- If the queries are not selected for execution, then only the query in the line where cursor is placed will be executed.
- If the cursor is placed next to an empty line, then the next available query statement will be executed.
- If the cursor is placed at the last line which is blank, then no query will be executed.
- If a single query is written in multiples lines and the cursor is placed at any line of the query, then that query is executed. Queries are separated using semicolon (;).

----End

Do as follow to execute an SQL query after a function/procedure:

Insert a forward slash (/) in a new line after the function/procedure and click in the **SQL Terminal** tab.

Do as follow to execute PL/SQL statements and SQL queries on different connections:

In the toolbar, select the required connection from the connection profiles drop-down list and click in the **SQL Terminal** tab.

Rename SQL Terminal

Follow the steps below to rename SQL Terminal:

Step 1 In the **SQL Terminal** tab right-click and select **Rename Terminal**.

A **Rename Terminal** dialog box is displayed prompting you to provide the new name for the Terminal.

Step 2 Enter the new name and select **OK** to rename the Terminal.

NOTE

- Terminal name follows Windows file naming convention.
- **Rename Terminal** allows a maximum of 150 characters.
- Restore option is not available to revert to the default name. You must manually rename the Terminal to default name.
- Tool tip of the renamed Terminal will display the old name.

----End

SQL Assistant

The **SQL Assistant** tool provides suggestion or reference for the information entered in **SQL Terminal** and **PL/SQL Viewer**. Follow the steps to open SQL Assistant:

When Data Studio is launched **SQL Assistant** panel displays with related syntax topics. As you type a query in the SQL Terminal topics related to the query is displayed. It also provides precautions, examples, syntax, function, and parameter description. Select the text and use the right-click option to copy selected information or copy and paste to SQL Terminal.

NOTE

- Choose **Settings** > **Preferences** > **Environment** > **Session Setting**. In the **SQL Assistant** area, enable or disable the **SQL Assistant** function permanently. By default, the **SQL Assistant** function is enabled permanently.
- After the **SQL Assistant** function is enabled, you can click the **SQL Assistant** icon ( ) on the toolbar to open the **SQL Assistant** window. If the **SQL Assistant** icon is gray after the **SQL Assistant** is enabled, the **SQL Assistant** is invalid.

Using Templates

Data Studio provides an option to insert frequently used SQL statements in **SQL Terminal** or **PL/SQL Viewer** using the **Templates** option. Some of the commonly used SQL statements are saved for ease of use. You can create, modify existing

templates or remove templates. For details about how to add, delete, and create a template, see **Table 4-6**.

The following table lists the default templates:

Name	Description
df	delete from
is	insert into
o	order by
s*	select from
sc	select row count
sf	select from
sl	select

Follow the steps to use the **Templates** option:

Step 1 Enter the name of the template in SQL Terminal/PL/SQL Viewer.

Step 2 Press **Alt+Ctrl+Space**.

A list of saved template information is displayed. The list displayed is based on the following criteria:

Exact Match	Display List
On	Displays all entries that match the input text case. **Example:** Entering "SF" in SQL Terminal/PL/SQL Viewer displays all entries that start with "SF".
Off	Displays all entries that match the input irrespective of the text case. **Example:** Entering "SF" in SQL Terminal/PL/SQL Viewer displays all entries that start with "SF", "Sf", "sF", or "sf".

Text Selection/Cursor Location	Display List
A text is selected and the shortcut key is used	Displays entries that match the text before the selection to the nearest space or new line character.
No text selected and the shortcut key is used	Displays entries that match the text before the cursor to the nearest space or new line character.

NOTE

- Using the shortcut key without entering text in SQL Terminal/PL/SQL Viewer displays all entries in the **Templates**.
- If the text entered in SQL Terminal/PL/SQL Viewer has only a single match, then it will be replaced directly in the SQL Terminal/PL/SQL Viewer without listing them out.

----End

4.7.11 Exporting Query Results

You can export the results of an SQL query into a CSV, Text or Binary file.

This section contains the following topics:

- **Exporting all data**
- **Exporting Data On the Current Page**

Exporting all data

The following functions are disabled while the export operation is in progress:

- Executing SQL queries in the **SQL Terminal**
- Executing PL/SQL statements
- Debugging PL/SQL statements

Follow the steps below to export all results:

Step 1 Select the **Result** tab.

Step 2 Click  . The **Export ResultSet Data** window appears.

Refer to **Exporting Table Data** to complete the export operation.

NOTE

You can check the status bar to view the status of the result being exported.

The **Data Exported Successfully** dialog box is displayed.

Step 3 Click **OK**. Data Studio displays the status of the operation in the **Messages** tab.

NOTE

If the disk is full while exporting the results, then Data Studio displays an error in the Messages tab. In this case, clear the disk, re-establish the connection and export the result data.

The Messages tab shows the **Execution Time**, **Total Result Records Fetched**, and the path where the file is saved.

----End

Exporting Data On the Current Page

It is recommended to export all results instead of exporting the current page. The **Export Current Page to CSV** function has been deleted.

Follow the steps below to export the current page:

Step 1 Select the **Result** tab.

Step 2 Click the  icon to export the current page.

The **Data Studio Security Disclaimer** dialog box is displayed.

Step 3 Click **OK**.

Step 4 Select the location to save the current page.

NOTE

You can check the status bar to view the status of the page being exported.

Step 5 Click **Save**. The **Data Exported Successfully** dialog box is displayed.

Step 6 Click **OK**. Data Studio displays the status of the operation in the **Messages** tab.

NOTE

If the disk is full while exporting the results, then Data Studio displays an error in the Messages tab. In this case, clear the disk, re-establish the connection and export the result data.

----End

4.7.12 Managing SQL Terminal Connections

Data Studio allows you to reuse an existing SQL Terminal connection or create a new SQL Terminal connection for execution plan and cost, visual explain plan, and operations in the resultset. By default, the SQL Terminal reuses the existing connection to perform these operations.

Use new connection when there are multiple queries queued for execution in existing connection as the queries are executed sequentially and there may be a delay. Always reuse existing connection while working on temp tables. Refer to the **Editing Temporary Tables** section to edit temp tables.

Complete the steps to enable or disable SQL Terminal connection reuse:

Step 1 Click  to enable or disable SQL Terminal connection reuse.

Refer to the **FAQs** section for the behavior of query execution with reuse and new connection.

NOTE

Use the existing SQL Terminal connection to edit temporary tables.

----End

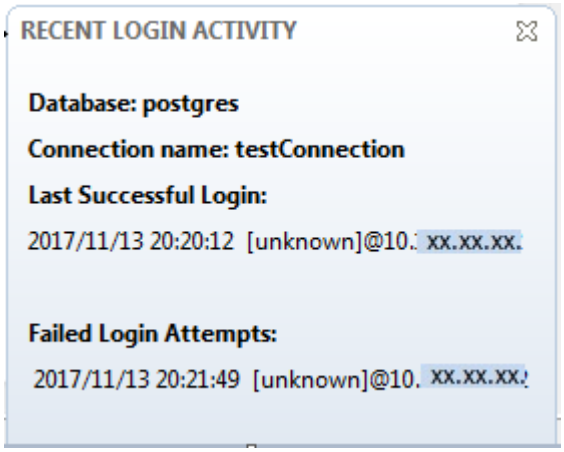
4.8 Security Management

NOTICE

Ensure that the operating system and the required software (refer to **System Requirements** for more details) are updated with the latest patches to prevent vulnerabilities and other security issues.

Login History

- When you log into a database, Data Studio displays a pop-up with details of the last successful login and failure attempts between the last two successful logins for you on the logged database.



NOTE

If the pop-up displays the message "Last login details not available", then it implies that the connected database does not support the last login display feature.

Password Expiry Notification

- Your password will expire within 7 days from the date of notification. If the password expires, contact the database administrator to reset the password.
- The password must be changed every 90 days.

Securing the Application In-Memory Data

While running Data Studio in a trusted environment, user must ensure to prevent malicious software scanning or accessing the memory which is used to store application data including sensitive information.

Data Encryption for Saved Data

You can ensure encryption of auto saved data by enabling encryption option from **Preferences** page. Refer to **Table 4-6** for steps to encrypt the saved data.

SQL History

- SQL History scripts are not encrypted.
- The SQL History list does not display sensitive queries that contain the following keywords:
  - Alter Role
  - Alter User
  - Create Role
  - Create User
  - Identified by
  - Password
- Some query syntax examples are listed below:
  - ALTER USER name [WITH] option [...]]
  - CREATE USER name [[WITH] option [...]]
  - CREATE ROLE name [[WITH] option [...]]
  - ALTER ROLE name [[WITH] option [...]]

4.9 Troubleshooting

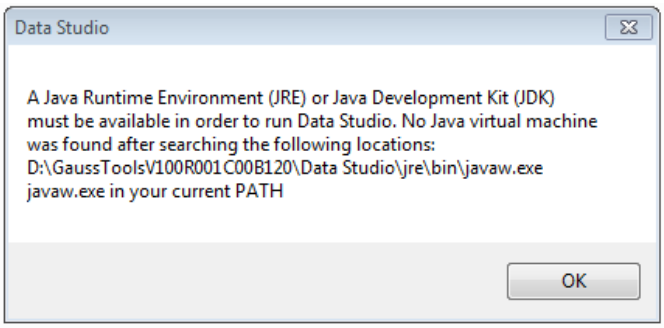
1. **Data Studio does not open.**

   Solution: Check whether JRE is missing. Verify the Java path in the environment. For details about the supported Java JDK versions, see **System Requirements**.

2. **Data Studio does not open and displays a 'Java Runtime' error when I double-click the Data Studio.exe file.**

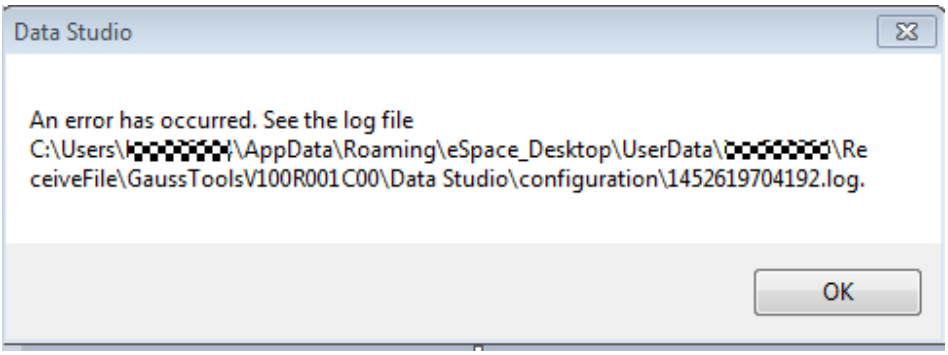
   Solution

   - Without JRE:

     Check whether the Java Runtime Environment (JRE) or Java Development Kit (JDK) version 1.8.0_141 or above with appropriate bit number is installed on the system and Java Home path is set. If there is more than one version of Java installed, set the **-vm** parameter in the configuration file by referring to **Configuring Data Studio**. This is a prerequisite for running Data Studio.

   - Old JRE version:



Check the version of Java Runtime Environment (JRE) or Java Development Kit (JDK) that is installed on the system. An older version installed on the system causes this error. Update the JRE to version 1.8.0_141 or above with appropriate bit number.

- Java Incompatibility:

Check the version of the JRE or JDK installed in the system. If the installed Java version is incompatible with the system, this error occurs. Update the JRE to version 1.8.0_141 or above with appropriate bit number.

You are advised to run the batch file to check compatibility and launch Data Studio. For details, see **Step 6**.

3. **The following information is displayed when you run the StartDataStudio.bat file:**

   Solution

Message	Solution
You are attempting to run 32-bit Data Studio on: • 64-bit OS • Windows 7 Professional • 64-bit Java 1.8 JDK (incompatible) Install 32-bit Java 1.8.	Install 32-bit Java 1.8.
The Java version supported by Data Studio must be 1.8 or later. Before using Data Studio, you need to install Java 1.8.	Install Java 1.8 that matches the number of bits of the operating system.
You are attempting to run 64-bit Data Studio on: • 64-bit OS • Windows 7 Professional • 32-bit Java 1.8 JDK (incompatible): Install 64-bit Java 1.8.	Install 64-bit Java 1.8.
You are attempting to run 64-bit Data Studio on: • 32 bit OS • Windows 7 Professional • 32-bit Java 1.8 JDK (incompatible) Install 32-bit Data Studio.	Install 32-bit Data Studio.

4. **Why does Data Studio not connect to the server even with all valid inputs?**

   Solution: Check whether the server is running on the specified IP address and port. Use gsql to connect to a specified user and check the user availability.

5. **What should I do for connection issues while using Data Studio?**

   Solution: A connection issue that may occur while using Data Studio is explained with an example:

   Establish a database connection.

   Run the query.

   When a connection exception occurs in any one of the databases (PostgreSQL), the connection is closed. When the database connection is closed, all the function and procedure tabs, if open, will be closed too.

The system displays an error message. The **Object Browser** navigation tree displays the database status.

NOTE

Only the current database will be disconnected. Other databases remain connected or are reconnected.

Re-connect to the database to proceed with execution.

6. **When a Java application is used to obtain a process that contains Chinese comments, the Chinese characters are invisible. What should I do?**

   Solution: Set **Preferences** > **Session Settings** > **Data Studio Encoding** and **File Encoding** to GBK, so that Chinese characters can be displayed properly.

7. **When I connect to the database and load a large number of SQL queries into SQL Terminal, the "Out Of Memory" or "Java Heap Error" error may occur in Data Studio. How do I solve this problem?**

   Solution: When the Data Studio has used up the allocated maximum Java memory, the message "Out of Memory" or "Java Heap Error" is displayed. By default, the **Data Studio.ini** configuration file (in the Data Studio installation path) contains the entry **-Xmx1200m**. 1200m indicates 1200 MB, which is the maximum Java memory that can be used by Data Studio. The memory usage of Data Studio depends on the size of data obtained by users during the use of Data Studio.

   To solve this problem, you can expand the Java memory size to an ideal value. For example, change **-Xmx1200m** to **-Xmx2000m** and restart Data Studio. If the updated memory is used up, the same problem may occur again.

   NOTE

   - For the 32-bit Data Studio with 8 GB RAM, the value of **Xmx** cannot exceed **2044**. For the 64-bit Data Studio with 8 GB RAM, the value of **Xmx** cannot exceed **6000**. The upper limit may vary with your current memory usage.

     For example:

     -Xms1024m

     -Xmx1800m

   - The maximum file size supported by Data Studio in the SQL Terminal depends on the value of Xmx in the Data Studio.ini file and the available memory.

8. **If a large amount of data is returned after the SQL query is executed, the Data Studio displays the "Insufficient Memory" error. What should I do?**

   Solution: Data Studio disconnects from the database specified in the file. Re-establish the connection and continue the operation.

9. **Why do I receive an export failure message when exporting DDL or data?**

   Solution: The possible causes are as follows:

   - An invalid client SSL certificate and/or client SSL key file was selected. Select a correct file and try again. For details, see **Adding a Connection**.
   - The identity of the object in the database may have changed. Check whether the identity of the object has changed and try again.
   - You do not have the required permissions. Contact the database administrator to obtain required permissions.

10. **Why does the system receive a message indicating that the DDL operation fails when the DDL operation is performed?**

Solution: The possible causes are as follows:

- An invalid client SSL certificate or client SSL key file was selected. Select a correct file and try again. For details, see **Adding a Connection**.
- The identity of the object in the database may have changed. Check whether the identity of the object has changed and try again.
- You do not have the required permissions. Contact the database administrator to obtain required permissions.

11. **Why do I receive the following error message when performing a Show DDL or Export DDL operation?**

    "Failed to start this program because MSVCRT100.dll is missing. Try reinstalling the program to resolve the problem."

    Solution: **gs_dump.exe** needs to be executed to display or export DDL, which requires the MS VC runtime library **msvcrt100.dll**.

    To resolve this issue, copy the **msvcrt100.dll file** from the **\Windows\System32** folder to the **\Windows\SysWOW64 folder**.

12. **Why is the saved connection details not displayed when I try to establish a connection?**

    Solution: If the **Profile** folder in the User Data folder is unavailable or has been manually modified, this problem may occur. Ensure that the **Profile** folder exists and its name meets the requirements.

13. **Why are historical sql query records lost when I close and restart data studio?**

    Solution: If the **Profile** folder in the User Data folder is lost or manually modified, this problem may occur. Ensure that the **Profile** folder exists and its name meets the requirements.

14. **Why does the system display a message indicating that the modification fails to be saved when I attempt to modify the syntax highlighting setting?**

    Solution: This problem may occur if the **Preferences** file does not exist or its name has been changed. Restart Data Studio to resolve this issue.

15. **What should I do if the Data Studio is in the idle state but the Data Studio.log file is in the No more handles state?**

    Solution: Restart Data Studio.

16. **What happens if I send a 303 error after editing a table and I cannot continue to modify the table?**

    Solution: All edited data will be lost. Close the **Edit Table Data** dialog box and modify the data again.

17. **Why is the message "The number of pasted cells does not match the number of selected cells" displayed when the operation is correct?**

    Solution: This problem occurs if you choose **Preferences** > **Query Results** and set the column headers to be included. The selected cells include the column header cells as well. Modify the settings to disable the Include column headers option and try again.

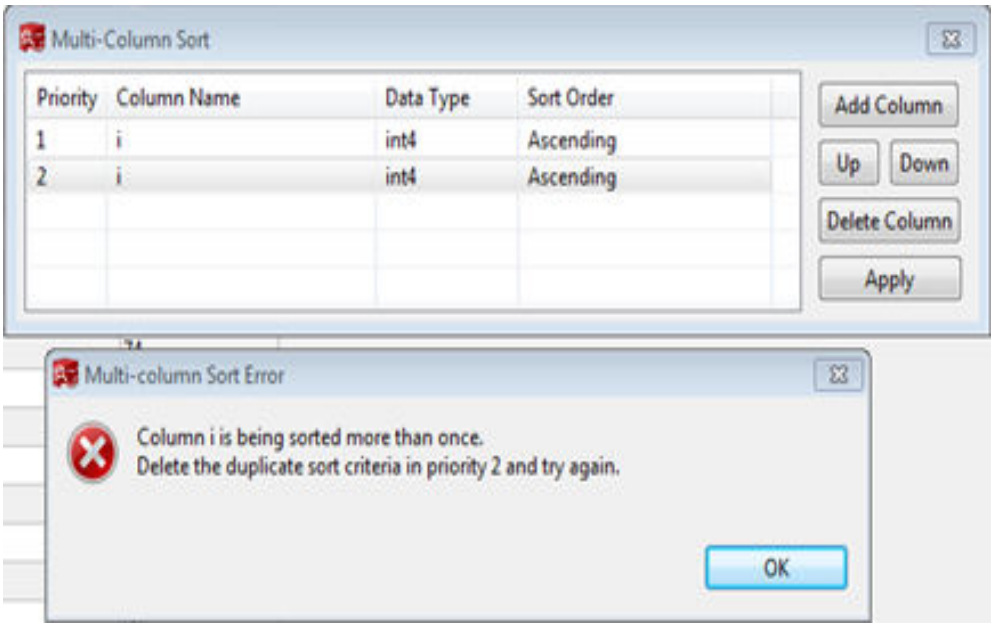
18. **Why can't I edit temporary tables when the Reuse Connections option is disabled?**

    Answer: After the Reuse Connection option is disabled, the tool creates a new session, but the temporary table can be edited only in the existing connection.

To edit temporary tables, enable the **Reuse Connection** option. For details, see **Managing SQL Terminal Connections**.

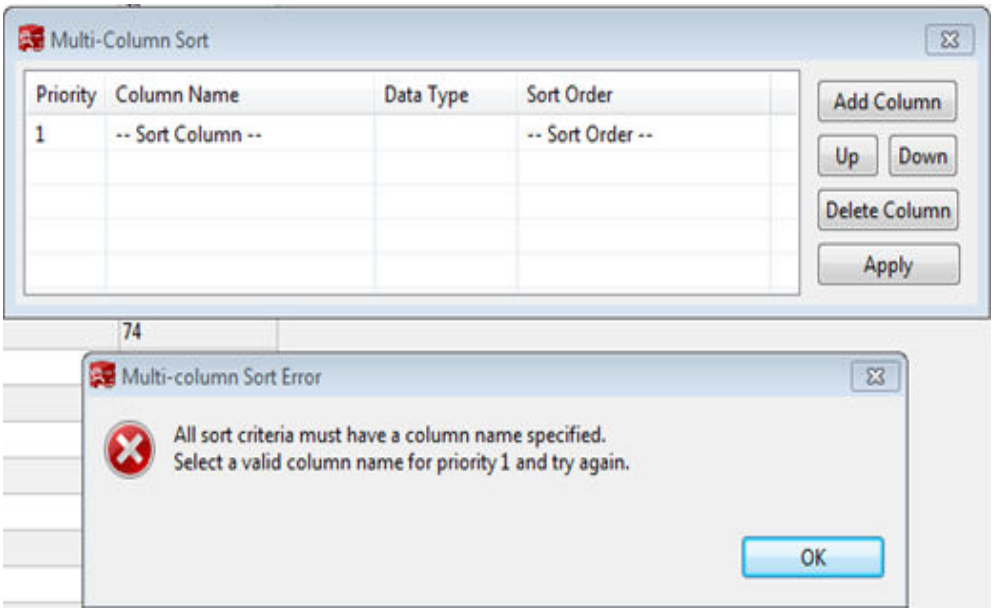
19. **What happens when I add the same column multiple times in a multi-column sort dialog box?**

    Answer: If you add the same column multiple times in the multi-column sorting dialog box and click Apply, the following message is displayed. You need to click **OK** and select non-duplicate columns for sorting.



20. **What happens when no column name is specified and Apply is clicked?**

    Answer: The following message is displayed. You need to set a valid column name and click **Apply** again. Then, the message is not displayed.



21. **What happens when I click Cancel while multiple table queries are running in the SQL terminal window?**

    Answer: Canceling a table query that is being executed may cause the console to display the names of tables that are not created. In this case, you

are advised to delete the table so that you can perform operations on tables with the same name.

22. What should I do if I cannot log in to Data Studio because the security key is cracked?

    Solution: Perform the following steps to generate a new security key:

    a. Choose **Datastudio** > **Userdata** and delete the security folder.

    b. Restart Data Studio.

    c. Create a security folder and generate a new key.

    d. Enter the password again to log in to Data Studio.

4.10 FAQs

1. **What do I need to check if my connection fails?**

   Answer: Check the following items:

   - Check whether **Connection Properties** are properly configured.
   - Check whether the server version is compatible with the client version.
   - Check whether the **database\pg_hba.conf** file is correctly configured.
   - Check whether the **Data Studio.ini** file is correctly configured.

2. **Why does the connection succeed when I try connecting to another server using an SSL certificate?**

   Answer: If the same SSL certificates are used by different servers, then the second connection will succeed because the certificates are cached.

   When you establish a connection with a different server using different SSL certificates, the connection will fail due to certificate mismatch.

3. **When I right-click on a function/procedure and refresh it in Object Browser, why does the function/procedure disappear?**

   Answer: This problem may occur if you drop a function/procedure and recreate it. In this case, refresh the parent folder to view the function/procedure in **Object Browser**.

4. **What do I do if a critical error occurs in a database session and operations cannot proceed?**

   Answer: Critical error may occur in some of the following cases. Check whether:

   - The connection is left idle for a long time and has timed out.
   - The server is running.
   - The server has sufficient memory and whether the Out Of Memory (OOM) error is reported to the server.

5. **What is a constraint?**

   Answer: Constraints are used to deny the insertion of unwanted data in columns. You can create restrictions on one or more columns in any table. It maintains the data integrity of the table.

   The following constraints are supported:

   - Primary Key constraint

- Unique Key constraint
- Check constraint

6. **What is an index?**

   Answer: An index is a copy of the selected column of a table that can be searched very efficiently. It also includes a low level disk block address or a direct link to the complete row of data it was copied from.

7. **What is the default encoding for Data Studio's files?**

   Answer: Exported, imported, and system files are encoded with the system's default encoding as configured in **Settings** > **Preferences**. The default encoding is UTF-8.

8. **When I try to open Data Studio, the system displays a message indicating that Data Studio does not support opening multiple instances. Why do I get this error message?**

   Answer: A user cannot open multiple instances in Data Studio.

9. **What do I do if a DDL statement running indefinitely and cannot be canceled?**

   Answer: This problem may occur if other DML/DDL operations are being performed on the same object. In this case, stop all the DML/DDL operations on the object and try again. If the problem persists, there may be another user performing DML/DDL operations on the object. Try again later. You can customize table data and check the operations in a transaction by following the instructions provided in **Data Studio GUI**.

10. **Why is the exported query result different from the data available on the Results tab?**

    Answer: When a result set data is exported, a new connection is used to execute the query again. The exported results may be different from the data on the **Result** tab.

11. **Why does last login information show "Last login details not available"?**

    Answer: This message is displayed when you connect to the database server of an earlier version or log in to the database for the first time after it is created.

12. **Why is the error marked incorrectly in SQL Terminal?**

    Answer: This problem occurs when the server returns an incorrect line number. You can view the error message on the **Message** tab and locate the correct row to rectify the fault.

13. **Will deleted columns be displayed after "Show DDL" or "Export DDL" operations?**

    Answer: Yes.

14. **Why cannot Data Studio be started after the -Xmx parameter is modified?**

    Answer: The value of **-Xmx** may be invalid. For details, see **Configuring Data Studio**.

15. **How do I quickly switch to the desired tab if there are multiple tabs open?**

    Answer: If the number of opened tabs reaches a certain limit (depending on your screen resolution), the  icon will be displayed at the end of the tab

list. Click this icon and select the required tab from the drop-down list. If this icon is not available, use the tooltip to identify the tabs. You also search for a **SQL Terminal** tab by its name. For example:

- *s, this displays all Terminal names that start with s.
- test, this displays all Terminal names that start with test.
- *2, this displays all Terminal names that contain 2 in them.

16. **Why does the language not change after I change the language setting and restart Data Studio?**

    Answer: Sometimes the language may not reflect the selected change post restart. Manually restart DS to open the tool in selected language.

17. **Why does the last login details information not display?**

    Answer: At times the server returns an error while trying to fetch last login details. In such scenarios the last login pop-up message does not display.

18. **When viewing/exporting DDL, why does the Chinese text not show properly?**

    Answer: This happens if the SQL, DDL, object names or data contains Chinese text and the Data Studio file encoding is not set to **GBK**. To solve this, go to **Settings** > **Preferences** > **Environment** > **File Encoding** and set the encoding to **GBK**. The supported combinations of Database and Data Studio encoding for export operation are shown in **Table1 Supported combinations of file encoding**.

    To open/view the exported files in Windows Explorer: Files exported with UTF-8 encoding can be opened/viewed by double-clicking it or by right-clicking on the file and selecting **Open**. Files exported with GBK encoding must be opened in Excel using the import external data feature (**Data** > **Get External Data** > From **Text**).

    Table 4-35 Supported combinations of file encoding

Database Encoding	Data Studio File Encoding	Support for Chinese Text in Table Names	Support for English Text in Table Names
GBK	GBK	Yes	Yes
GBK	UTF-8	No - Incorrect details	No - Incorrect details
UTF-8	GBK	No - Export Fails	No - Incorrect details
UTF-8	UTF-8	Yes	Yes
UTF-8	LATIN1	No - Export Fails	Yes
SQL_ASCII	GBK	Yes	Yes
SQL_ASCII	UTF-8	No - Incorrect details	No - Incorrect details

19. **Why do I get the error message "Conversion between GBK and LATIN1 is not supported"?**

Answer: This message occurs if the Data Studio and Database encoding selected are incompatible. To solve this, select the compatible encoding. Compatible encoding is shown in **Table 4-36**.

Table 4-36 Compatible encoding formats

Data Studio File Encoding	Database Encoding	Compatible or Not
UTF-8	GBK	Yes
	LATIN1	Yes
	SQL_ASCII	Yes
GBK	UTF-8	Yes
	LATIN1	No
	SQL_ASCII	Yes
SQL_ASCII	UTF-8	Yes
	LATIN1	Yes
	GBK	Yes

20. **Why is the PL/SQL procedure I compiled and executed is saved as PL/SQL function?**

    Answer: The database does not differentiate between PL/SQL function and procedure. All procedures in databases are functions. Hence PL/SQL procedure is saved as PL/SQL function.

21. **Why is that I am not able to edit the distribution key?**

    Answer: The database allows you to edit the distribution key only for the first insert operation.

22. **While editing table data if I do not enter a value for default value column, will the value be added by the database server?**

    Answer: Yes, the database server will add the value but the value will not be visible after save in the **Edit Table Data** tab. Use the refresh option from the **Edit Table Data** tab or re-open the table again to view the added default value(s).

23. **While modifying/deleting table data why do I get a pop-up stating that more than one matching row found?**

    Answer: This happens because there are additional rows detected for modification/deletion based on Custom Unique Key or All Columns selection. If Custom Unique Key is selected, then it will delete/modify the rows that have exact match of the data in the column selected for deletion/modification. If All Columns is selected, then it will delete/modify the rows that match data in all columns. Hence the duplicate records matching the Custom Unique Key or All Columns will be deleted/modified if Yes is selected. If **No** is selected, the row that is not saved will be marked for correction.

24. **When I right-click on a text box I see additional context menu options. Why does this happen?**

Answer: Windows 7 offers extra context menu choices, such as Right to Left Reading Order and Show Unicode Control Characters, if your keyboard supports both left-to-right and right-to-left input.

25. **What are the objects that are not supported for batch export DDL & DDL and Data operations?**

    Answer: Following objects are not supported for DDL & DDL and Data operations.

    Export DDL:

    Connection, database, foreign table, sequence, column, index, constraint, partition, function/procedure group, regular tables group, views group, schemas group, and system catalog group.

    Export DDL and Data

    Connection, database, namespace, foreign table, sequence, column, index, constraint, partition, function/procedure, view, regular tables group, schemas group, and system catalog group.

26. **Will the queries in SQL Terminal be committed if the result set is modified and saved with Reuse Connection on and Auto Commit off?**

    Answer: No. Queries will only be committed when COMMIT command is executed in the Terminal.

Auto Commit	Reuse Connection	Resultset Save
On	On	Commit
On	Off	Commit
Off	On	Does not commit
Off	Off	Not supported

27. **Why do incorrect table details appear when I query a temp table from a new SQL Terminal?**

    Answer: When you query a temp table from a new SQL Terminal or with the **Reuse Connection** off, the result set displays information of a regular/ partition/foreign table, if a table with the same name as the temp table exists.

    NOTE

    If the **Reuse Connection** is **On**, the result set displays information of the temp table even if another table with the same name exists.

28. **Which are the operations that are performed on a locked object does not run in the background but needs to be manually closed?**

    Answer: Following are the operations that do not run in background while the object is locked in another operation:

Operations	
Renaming a table	Creating a constraint
Setting schema on table	Creating an index

Operations	
Setting description in table	Adding column
Renaming a partition	-

29. **Do we have a limit on the column and row size while exporting table data to excel?**

    Answer: Yes, xlsx format supports maximum of 1 million rows and 16384 columns and xls format supports maximum of 64,000 rows and 256 columns.

30. **How Do I Delete Objects in Batches?**

    The batch drop operation allows you to drop multiple objects. This operation also applies to searched objects.

    NOTE

    - Batch drop is allowed only for databases.
    - An error is reported on batch dropping system objects, which cannot be dropped.

    Perform the following steps to batch drop objects:

    Press **Ctrl** and select objects one by one or press **Shift** and select objects in a bunch to select the objects to be dropped.

    Right-click and select **Drop Objects**.

    The **Drop Objects** tab displays the list of objects to be dropped.

Column Name	Description
Type	Displays information about the object type.
Name	Displays the name of the object.
Query	Displays the query that will be executed to drop objects.
Type	Displays the status of the drop operation. • To start: The drop operation has not been started. • In progress: The object is being dropped. • Completed: The drop operation has been completed. • Error: The object has not been dropped due to an error.
Error Message	Displays the failure cause of a drop operation.

Select the required parameters.

Option	Description
Cascade	The cascade drop operation is performed to drop dependent objects and attributes. The dropped dependent objects will be removed from **Object Browser** only after the refresh operation is performed.
Atomic	The atomic drop operation is performed to drop all objects. If the operation fails, no objects will be dropped.
No selection	If neither **Cascade** nor **Atomic** is selected, no dependent objects are dropped.

Click **Start**.

Runs: displays the number of objects that are dropped from the object list

Errors: displays the number of objects that are not dropped due to errors

Click **Stop** or close the **Drop Objects** dialog box to stop the drop operation.

For details about copy, advanced copy, show/hide search bar, sort, and column reorder options, see **Execute SQL Queries**.

NOTE

- Select part of a cell and press **Ctrl+C** or click **Copy** to copy the selected text in the cell.
- When you select multiple objects in **Object Browser** to drop, a batch drop window is displayed and the object icons are enabled in the menu bar. If you disconnect the database, the icons will remain disabled even after reconnection. In this case, you need to reselect the objects to drop and the selected objects will be displayed in the new batch drop window.

31. **How Do I Grant or Revoke Permissions in Batches for a Specified Object?**

    You can select multiple objects at a time or search for the target objects.

    Press **Ctrl** and select objects one by one, or press **Shift** and select objects in batches, and choose **Grant/Revoke** from the shortcut menu.

    NOTE

    - Only objects with the same schema and type can be granted or revoked in batches.
    - This feature is only supported in OLAP, not in OLTP.

5 GDS

5.1 Installing, Configuring, and Starting GDS

Scenario

GaussDB(DWS) uses GDS to allocate the source data for parallel data import. Deploy GDS on the data server.

If a large volume of data is stored on multiple data servers, install, configure, and start GDS on each server. Then, data on all the servers can be imported in parallel. The procedure for installing, configuring, and starting GDS is the same on each data server. This section describes how to perform this procedure on one data server.

Context

The GDS version must match the cluster version. For example, GDS V100R008C00 matches DWS 1.3.*X*. Otherwise, the import or export may fail, or the import or export process may fail to respond. Therefore, use the latest version of GDS.

After the database is upgraded, download the latest version of GaussDB(DWS) GDS as instructed in **Procedure**. When the import or export starts, GaussDB(DWS) checks the GDS versions. If the versions do not match, an error message is displayed and the import or export is terminated.

To obtain the version number of GDS, run the following command in the GDS decompression directory:

```
gds -V
```

To view the database version, run the following SQL statement after connecting to the database:

```
SELECT version();
```


Procedure

Step 1 Before using GDS to import or export data, see "Preparing an ECS as the GDS Server" and "Downloading the GDS Package" in **"Tutorial: Using GDS to Import Data from a Remote Server"**.

Step 2 Log in as user **root** to the data server where GDS is to be installed and run the following command to create the directory for storing the GDS package:

```
mkdir -p /opt/bin/dws
```

Step 3 Upload the GDS package to the created directory.

Use the SUSE Linux package as an example. Upload the GDS package **dws_client_8.*x.x*_suse_x64.zip** to the directory created in the previous step.

Step 4 (Optional) If SSL is used, upload the SSL certificates to the directory created in **Step 2**.

Step 5 Go to the directory and decompress the package.

```
cd /opt/bin/dws
unzip dws_client_8.x.x_suse_x64.zip
```

Step 6 Create a GDS user and the user group to which the user belongs. This user is used to start GDS and read source data.

```
groupadd gdsgrp
useradd -g gdsgrp gds_user
```

Step 7 Change the owner of the GDS package directory and source data file directory to the GDS user.

```
chown -R gds_user:gdsgrp /opt/bin/dws/gds
chown -R gds_user:gdsgrp /input_data
```

Step 8 Switch to user **gds_user**.

```
su - gds_user
```

If the current cluster version is 8.0.*x* or earlier, skip **Step 9** and go to **Step 10**.

If the current cluster version is 8.1.*x*, go to the next step.

Step 9 Execute the script on which the environment depends (applicable only to version 8.1.*x*).

```
cd /opt/bin/dws/gds/bin
source gds_env
```

Step 10 Start GDS.

GDS is eco-friendly software that can be launched once it is decompressed. You can start GDS in two ways: by using the **gds** command to configure startup parameters or by writing the parameters into the **gds.conf** configuration file and running the **gds_ctl.py** command to initiate GDS.

The first method is recommended when you do not need to import data again. The second method is recommended when you need to import data regularly.

- Method 1: Run the **gds** command to start GDS.
  - If data is transmitted in non-SSL mode, run the following command to start GDS:

    ```
    gds -d dir -p ip:port -H address_string -l log_file -D -t worker_num
    ```

    Example:


```
/opt/bin/dws/gds/bin/gds -d /input_data/ -p 192.168.0.90:5000 -H 10.10.0.1/24 -
l /opt/bin/dws/gds/gds_log.txt -D -t 2
```

  - If data is transmitted in SSL mode, run the following command to start GDS:

    ```
    gds -d dir -p ip:port -H address_string -l log_file -D
    -t worker_num --enable-ssl --ssl-dir Cert_file
    ```

    Example:

    Run the following command to upload the SSL certificate mentioned in **Step 4** to **/opt/bin**:

    ```
    /opt/bin/dws/gds/bin/gds -d /input_data/ -p 192.168.0.90:5000 -H 10.10.0.1/24 -
    l /opt/bin/dws/gds/gds_log.txt -D --enable-ssl --ssl-dir /opt/bin/
    ```

  Replace the information in italic as required.

  - **-d** *dir*: directory for storing data files that contain data to be imported. This tutorial uses **/input_data/** as an example.
  - **-p** *ip:port*: listening IP address and port for GDS. The default value is **127.0.0.1**. Replace it with the IP address of a 10GE network that can communicate with GaussDB(DWS). The port number ranges from 1024 to 65535. The default port is **8098**. This tutorial uses **192.168.0.90:5000** as an example.
  - **-H** *address_string*: specifies the hosts that are allowed to connect to and use GDS. The value must be in CIDR format. Configure this parameter to enable a GaussDB(DWS) cluster to access GDS for data import. Ensure that the network segment covers all hosts in a GaussDB(DWS) cluster.
  - **-l** *log_file*: GDS log directory and log file name. This tutorial uses **/opt/bin/dws/gds/gds_log.txt** as an example.
  - **-D**: GDS in daemon mode. This parameter is used only in Linux.
  - **-t** *worker_num*: number of concurrent GDS threads. If the data server and GaussDB(DWS) have available I/O resources, you can increase the number of concurrent GDS threads.

    GDS determines the number of threads based on the number of concurrent import transactions. Even if multi-thread import is configured before GDS startup, the import of a single transaction will not be accelerated. By default, an **INSERT** statement is an import transaction.
  - **--enable-ssl**: enables SSL for data transmission.
  - **--ssl-dir** *Cert_file*: SSL certificate directory. Set this parameter to the certificate directory in **Step 4**.
  - For details about GDS parameters, see **gds**.

- Method 2: Write the startup parameters into the **gds.conf** configuration file and run the **gds_ctl.py** command to start GDS.

  a. Run the following command to go to the **config** directory of the GDS package and modify the **gds.conf** configuration file. For details about the parameters in the **gds.conf** configuration file, see **Table 5-1**.

  ```
  vim /opt/bin/dws/gds/config/gds.conf
  ```

  Example:

  The **gds.conf** configuration file contains the following information:

  ```
  <?xml version="1.0"?>
  <config>
  <gds name="gds1" ip="192.168.0.90" port="5000" data_dir="/input_data/" err_dir="/err"
  ```



```
data_seg="100MB" err_seg="100MB" log_file="/log/gds_log.txt" host="10.10.0.1/24"
daemon='true' recursive="true" parallel="32"></gds>
</config>
```

Information in the configuration file is described as follows:

- The data server IP address is **192.168.0.90** and the GDS listening port is **5000**.
- Data files are stored in the **/input_data/** directory.
- Error log files are stored in the **/err** directory. The directory must be created by a user who has the GDS read and write permissions.
- The size of a single data file is 100 MB.
- The size of a single error log file is 100 MB.
- Logs are stored in the **/log/gds_log.txt** file. The directory must be created by a user who has the GDS read and write permissions.
- Only nodes with the IP address **10.10.0.***** can be connected.
- The GDS process is running in daemon mode.
- Recursive data file directories are used.
- The number of concurrent import threads is 2.

b. Start GDS and check whether it has been started.

```
python3 gds_ctl.py start
```

Example:

```
cd /opt/bin/dws/gds/bin
python3 gds_ctl.py start
Start GDS gds1                  [OK]
gds [options]:
 -d dir            Set data directory.
 -p port           Set GDS listening port.
    ip:port        Set GDS listening ip address and port.
 -l log_file       Set log file.
 -H secure_ip_range
                   Set secure IP checklist in CIDR notation. Required for GDS to start.
 -e dir            Set error log directory.
 -E size           Set size of per error log segment.(0 < size < 1TB)
 -S size           Set size of data segment.(1MB < size < 100TB)
 -t worker_num     Set number of worker thread in multi-thread mode, the upper limit is 200. If
without setting, the default value is 8.
 -s status_file    Enable GDS status report.
 -D                Run the GDS as a daemon process.
 -r                Read the working directory recursively.
 -h                Display usage.
```

----End

gds.conf Parameter Description

Table 5-1 gds.conf configuration description

Attribute	Description	Value Range
name	Identifier	-

Attribute	Description	Value Range
ip	Listening IP address	The IP address must be valid. Default value: **127.0.0.1**
port	Listening port	Value range: 1024 to 65535 (integer) Default value: **8098**
data_dir	Data file directory	-
err_dir	Error log file directory	Default value: data file directory
log_file	Log file Path	-
host	Host IP address allowed to be connected to GDS (The value must in CIDR format and this parameter is available for the Linux OS only.)	-
recursive	Whether the data file directories are recursive	Value range: • **true**: recursive • **false**: not recursive Default value: **false**
daemon	Whether the process is running in daemon mode	Value range: • **true**: The process is running in daemon mode. • **false**: The process is not running in daemon mode. Default value: **false**
parallel	Number of concurrent data import threads	Value range: 0 to 200 (integer) Default value: **8**

5.2 Stopping GDS

Scenarios

Stop GDS after data is imported successfully.

Procedure

Step 1 Log in as user **gds_user** to the data server where GDS is installed.

Step 2 Select the mode of stopping GDS based on the mode of starting it.

- If GDS is started using the **gds** command, perform the following operations to stop GDS:

a. Query the GDS process ID:
```
ps -ef|grep gds
```
For example, the GDS process ID is 128954.
```
ps -ef|grep gds
gds_user 128954      1  0 15:03 ?        00:00:00 gds -d /input_data/ -p 192.168.0.90:5000 -l /log/gds_log.txt -D
gds_user 129003 118723  0 15:04 pts/0    00:00:00 grep gds
```
b. Run the **kill** command to stop GDS. **128954** in the command is the GDS process ID.
```
kill -9 128954
```

----End

5.3 Example of Importing Data Using GDS

Example: Parallel Import from Multiple Data Servers

The data servers reside on the same intranet as the cluster. Their IP addresses are 192.168.0.90 and 192.168.0.91. Source data files are in CSV format.

1. Create the target table **tpcds.reasons**.
```
CREATE TABLE tpcds.reasons
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
);
```
2. Log in to each GDS data server as user **root** and create the **/input_data** directory for storing data files on the servers. The following takes the data server whose IP address is 192.168.0.90 as an example. Operations on the other server are the same.
```
mkdir -p /input_data
```
3. (Optional) Create a user and the user group it belongs to. The user is used to start GDS. If the user and user group already exist, skip this step.
```
groupadd gdsgrp
useradd -g gdsgrp gds_user
```
4. Evenly distribute source data files to the **/input_data** directories on the data servers.
5. Change the owners of source data files and the **/input_data** directory on each data server to **gds_user**. The data server with the IP address 192.168.0.90 is used as an example.
```
chown -R gds_user:gdsgrp /input_data
```
6. Log in to each data server as user **gds_user** and start GDS.

   The GDS installation path is **/opt/bin/dws/gds**. Source data files are stored in **/input_data/**. The IP addresses of the data servers are 192.168.0.90 and 192.168.0.91. The GDS listening port is 5000. GDS runs in daemon mode.

   Start GDS on the data server whose IP address is 192.168.0.90.
```
/opt/bin/dws/gds/gds -d /input_data -p 192.168.0.90:5000 -H 10.10.0.1/24 -D
```
   Start GDS on the data server whose IP address is 192.168.0.91.
```
/opt/bin/dws/gds/gds -d /input_data -p 192.168.0.91:5000 -H 10.10.0.1/24  -D
```
7. Create the foreign table **tpcds.foreign_tpcds_reasons** for the source data.

   Set import mode parameters as follows:

- Set the import mode to **Normal**.
- When GDS is started, the source data file directory is **/input_data** and the GDS listening port is 5000. Therefore, set **location** to **gsfs://192.168.0.90:5000/* | gsfs://192.168.0.91:5000/***.

Information about the data format is set based on data format parameters specified during data export. The parameter settings are as follows:

- **format** is set to **CSV**.
- **encoding** is set to **UTF-8**.
- **delimiter** is set to **E'\x08'**.
- **quote** is set to **0x1b**.
- **null** is set to an empty string without quotation marks.
- **escape** is set to the same value as that of **quote** by default.
- **header** is set to **false**, indicating that the first row is regarded as a data row when a file is imported.

Set import error tolerance parameters as follows:

- Set **PER NODE REJECT LIMIT** (number of allowed data format errors) to **unlimited**. In this case, all the data format errors detected during data import will be tolerated.
- Set **LOG INTO** to **err_tpcds_reasons**. The data format errors detected during data import will be recorded in the **err_tpcds_reasons** table.

Based on the above settings, the foreign table is created using the following statement:

```
CREATE FOREIGN TABLE tpcds.foreign_tpcds_reasons
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
)
SERVER gsmpp_server OPTIONS (location 'gsfs://192.168.0.90:5000/* | gsfs://192.168.0.91:5000/*',
format 'CSV',mode 'Normal', encoding 'utf8', delimiter E'\x08', quote E'\x1b', null '', fill_missing_fields
'false') LOG INTO err_tpcds_reasons PER NODE REJECT LIMIT 'unlimited';
```

8. Import data through the foreign table **tpcds.foreign_tpcds_reasons** to the target table **tpcds.reasons**.

   ```
   INSERT INTO tpcds.reasons SELECT * FROM tpcds.foreign_tpcds_reasons;
   ```

9. Query data import errors in the **err_tpcds_reasons** table and rectify the errors (if any). For details, see **Handling Import Errors**.

   ```
   SELECT * FROM err_tpcds_reasons;
   ```

10. After data import is complete, log in to each data server as user **gds_user** and stop GDS.

    The data server with the IP address 192.168.0.90 is used as an example. The GDS process ID is 128954.

    ```
    ps -ef|grep gds
    gds_user 128954      1  0 15:03 ?        00:00:00 gds -d /input_data -p 192.168.0.90:5000 -D
    gds_user 129003 118723  0 15:04 pts/0    00:00:00 grep gds
    kill -9 128954
    ```

Example: Data Import Using Multiple Threads

The data server resides on the same intranet as the cluster. The server IP address is 192.168.0.90. Source data files are in CSV format. Data will be imported to two tables using multiple threads in **Normal** mode.

1. In the database, create the target tables **tpcds.reasons1** and **tpcds.reasons2**.
```
CREATE TABLE tpcds.reasons1
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
) ;
CREATE TABLE tpcds.reasons2
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
) ;
```
2. Log in to the GDS data server as user **root**, and then create the data file directory **/input_data** and its sub-directories **/input_data/import1/** and **/input_data/import2/**.
```
mkdir -p /input_data
```
3. Store the source data files of the target table **tpcds.reasons1** in **/input_data/import1/** and the source data files of the target table **tpcds.reasons2** in **/input_data/import2/**.
4. (Optional) Create a user and the user group it belongs to. The user is used to start GDS. If the user and user group already exist, skip this step.
```
groupadd gdsgrp
useradd -g gdsgrp gds_user
```
5. Change the owners of source data files and the **/input_data** directory on the data server to **gds_user**.
```
chown -R gds_user:gdsgrp /input_data
```
6. Log in to the data server as user **gds_user** and start GDS.

   The GDS installation path is **/gds**. Source data files are stored in **/input_data/**. The IP address of the data server is 192.168.0.90. The GDS listening port is 5000. GDS runs in daemon mode. The degree of parallelism is 2. A recursive directory is specified.
```
/gds/gds -d /input_data -p 192.168.0.90:5000 -H 10.10.0.1/24  -D -t 2 -r
```
7. In the database, create the foreign tables **tpcds.foreign_tpcds_reasons1** and **tpcds.foreign_tpcds_reasons2** for the source data.

   The foreign table **tpcds.foreign_tpcds_reasons1** is used as an example to describe how to configure parameters in a foreign table.

   Set import mode parameters as follows:

   - Set the import mode to **Normal**.
   - When GDS is started, the configured source data file directory is **/input_data** and the GDS listening port is 5000. However, source data files are actually stored in **/input_data/import1/**. Therefore, set **location** to **gsfs://192.168.0.90:5000/import1/***.

   Information about the data format is set based on data format parameters specified during data export. The parameter settings are as follows:

   - **format** is set to **CSV**.
   - **encoding** is set to **UTF-8**.
   - **delimiter** is set to **E'\x08'**.
   - **quote** is set to **0x1b**.
   - **null** is set to an empty string without quotation marks.
   - **escape** is set to the same value as that of **quote** by default.

- **header** is set to **false**, indicating that the first row is regarded as a data row when a file is imported.

Set import error tolerance parameters as follows:

- Set **PER NODE REJECT LIMIT** (number of allowed data format errors) to **unlimited**. In this case, all the data format errors detected during data import will be tolerated.
- Set **LOG INTO** to **err_tpcds_reasons1**. The data format errors detected during data import will be recorded in the **err_tpcds_reasons1** table.
- If the last column (**fill_missing_fields**) in a source data file is missing, the **NULL** column will be automatically added to the target file.

Based on the preceding settings, the foreign table **tpcds.foreign_tpcds_reasons1** is created as follows:

```
CREATE FOREIGN TABLE tpcds.foreign_tpcds_reasons1
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
) SERVER gsmpp_server OPTIONS (location 'gsfs://192.168.0.90:5000/import1/*', format 'CSV',mode 'Normal', encoding 'utf8', delimiter E'\x08', quote E'\x1b', null '',fill_missing_fields 'on')LOG INTO err_tpcds_reasons1 PER NODE REJECT LIMIT 'unlimited';
```

Based on the preceding settings, the foreign table **tpcds.foreign_tpcds_reasons2** is created as follows:

```
CREATE FOREIGN TABLE tpcds.foreign_tpcds_reasons2
(
  r_reason_sk integer not null,
  r_reason_id char(16) not null,
  r_reason_desc char(100)
) SERVER gsmpp_server OPTIONS (location 'gsfs://192.168.0.90:5000/import2/*', format 'CSV',mode 'Normal', encoding 'utf8', delimiter E'\x08', quote E'\x1b', null '',fill_missing_fields 'on')LOG INTO err_tpcds_reasons2 PER NODE REJECT LIMIT 'unlimited';
```

8. Import data to **tpcds.reasons1** using the foreign table **tpcds.foreign_tpcds_reasons1**, and to **tpcds.reasons2** using the foreign table **tpcds.foreign_tpcds_reasons2**.

   ```
   INSERT INTO tpcds.reasons1 SELECT * FROM tpcds.foreign_tpcds_reasons1;
   INSERT INTO tpcds.reasons2 SELECT * FROM tpcds.foreign_tpcds_reasons2;
   ```

9. Query data import errors in the **err_tpcds_reasons1** and **err_tpcds_reasons2** tables and rectify the errors (if any). For details, see **Handling Import Errors**.

   ```
   SELECT * FROM err_tpcds_reasons1;
   SELECT * FROM err_tpcds_reasons2;
   ```

10. After data import is complete, log in to the data server as user **gds_user** and stop GDS.

    The GDS process ID is 128954.

    ```
    ps -ef|grep gds
    gds_user 128954      1  0 15:03 ?        00:00:00 gds -d /input_data -p 192.168.0.90:5000 -D -t 2 -r
    gds_user 129003 118723  0 15:04 pts/0    00:00:00 grep gds
    kill -9 128954
    ```

5.4 gds

Context

gds is used to import and export data of GaussDB(DWS). For details, see "Importing Data" and "Exporting Data" in the *Developer Guide*.

Syntax

```
gds [ OPTION ] -d DIRECTORY
```

The **-d** and -**H** parameters are mandatory and **option** is optional. gds provides the file data of **DIRECTORY** for GaussDB(DWS) to access.

Before starting GDS, you need to ensure that your GDS version is consistent with the database version. Otherwise, the database will display an error message and terminate the import and export operations. You can view the specific version through the **-V** parameter.

Parameter Description

- -d dir

  Set the directory of the data file to be imported. If the GDS process permission is sufficient, the directory specified by **-**d is automatically created.

- -p ip:port

  Set the IP address and port to be listened to of the GDS.

  Value range of the IP address: The IP address must be valid.

  Default value: **127.0.0.1**

  Value range of the listening port is a positive integer ranging from 1024 to 65535.

  Default value of **port**: **8098**

- -l log_file

  Set the log file. This feature adds the function of automatical log splitting. After the **-R** parameter is set, GDS generates a new file based on the set value to prevent a single log file from being too large.

  Generation rule: By default, GDS identifies only files with the **.log** extension name and generates new log files.

  For example, if **-l** is set to **gds.log** and **-R** is set to 20 MB, a **gds-2020-01-17_115425.log** file will be created when the size of **gds.log** reaches 20 MB.

  If the log file name specified by **-l** does not end with **.log**, for example, **gds.log.txt**, the name of the new log file is **gds.log-2020-01-19_122739.txt**.

  When GDS is started, it checks whether the log file specified by **-l** exists. If the log file exists, a new log file is generated based on the current date and time, and the original log file is not overwritten.

  NOTE

  If the number of GDS process logs in the current directory exceeds the value of --**log-filecount**, old logs will be reclaimed. To save a large number of logs, you are advised to set an independent directory for each GDS process and increase the value of --**log-filecount**.

- -H address_string

  Set the hosts that can be connected to the GDS. This parameter must be the CIDR format and it supports the Linux system only. If multiple network segments need to be configured, use commas (,) to separate them. For example, **-H 10.10.0.0/24, 10.10.5.0/24**.

- -e dir

Set the saving path of error logs generated when data is imported.

Default value: **data file directory**

- -E size

  Set the upper thread of error logs generated when data is imported.

  Value range: 0 < size < 1 TB. The value must be a positive integer plus the unit. The unit can be KB, MB, or GB.

- -S size

  Set the upper limit of the exported file size.

  Value range: 1 MB < size < 100 TB. The value must be a positive integer plus the unit. The unit can be KB, MB, or GB. If KB is used, the value must be greater than 1024 KB.

- -R size

  Set the maximum size of a single GDS log file specified by **-l**.

  Value range: 1 MB < size < 100 TB. The value must be a positive integer plus the unit. The unit can be KB, MB, or GB. If KB is used, the value must be greater than 1024 KB.

  Default value: 16 MB

- -t worker_num

  Set the number of concurrent imported and exported working threads.

  Value range: The value is a positive integer ranging between 0 and 200 (included).

  Default value: **8**

  Recommended value: 2 x CPU cores in the common file import and export scenario; in the pipe file import and export scenario, set the value to **64**.

  NOTE

  If a large number of pipe files are imported and exported concurrently, the value of this parameter must be greater than or equal to the number of concurrent services.

- -s status_file

  Set the status file. This parameter supports the Linux system only.

- -D

  The GDS is running on the backend and this parameter supports the Linux system only.

- -r

  Traverse files in the recursion directory and this parameter supports the Linux system only.

- -h

  Show help information.

- --enable-ssl

  Use the SSL authentication mode to communicate with clusters.

- --ssl-dir Cert_file

  Before using the SSL authentication mode, specify the path for storing the authentication certificates.

- --debug-level

Set the debug log level of the GDS to control the output of GDS debug logs.

Value range: 0, **1**, and **2**

- **0**: Only the file list related to log import and export is printed. If the log volume is small, set the parameter to this value only when the system is at normal state.
- **1**: All the log information is printed, including the connection information, session switch information, and statistics on each node.
- **2**: Detailed interaction logs and their status are printed to generate a huge number of debug logs to help identify the fault causes. You are advised to set the parameter to this value only during troubleshooting.

Default value: **0**

- --log-filecount

  Specifies the maximum number of log files that can be retained. When the number of log files exceeds the value of this parameter, the latest created log files are retained.

  Value range: 5<=**log-filecount**<=1024 (positive integer)

  Default value: **50**

- --pipe-timeout

  Specify the timeout period for GDS to wait for operating a pipe.

  NOTE

  - This parameter is used to prevent the following situation: One end of the pipe file is not read or written for a long time due to human or program problems. As a result, the read or write operation on the other end of the pipe is hung.
  - This parameter does not indicate the maximum duration of a data import or export task. It indicates the maximum timeout duration of each read, open, or write operation on the pipe. If the timeout duration exceeds the value of **--pipe-timeout**, an error is reported to the frontend.

  Value range: greater than 1s Use a positive integer with the time unit, seconds (s), minutes (m), or hours (h). Example: **3600s**, **60m**, or **1h**, indicating one hour.

  Default value: **1h**/**60m**/**3600s**

- --pipe-size

  Specifies the size of the GDS pipe file to be imported or exported.

  Value range: greater than 1K

  Default value: maximum value allowed by the operating system. You can run the **cat /proc/sys/fs/pipe-max-size** command to view the default value.

  NOTE

  This parameter can be used only when the Linux kernel version is 2.6.35 or later.

Examples

Data file is saved in the **/data** directory, the IP address is 192.168.0.90, and the listening port number is 5000.

```
gds -d /data/ -p 192.168.0.90:5000 -H 10.10.0.1/24
```


Data file is saved in the subdirectory of the **/data** directory, the IP address is 192.168.0.90, and the listening port number is 5000.

```
gds -d /data/ -p 192.168.0.90:5000 -H 10.10.0.1/24 -r
```

Data file is saved in the **/data** directory, the IP address is 192.168.0.90, and the listening port number is 5000 which is running on the backend. The log file is saved in the **/log/gds_log.txt** file, and the specified number of the concurrently imported working threads is 32.

```
gds -d /data/ -p 192.168.0.90:5000 -H 10.10.0.1/24 -l /log/gds_log.txt -D  -t 32
```

Data file is saved in the **/data** directory, the IP address is 192.168.0.90, and the listening port number is 5000. Only the IP address of **10.10.0.*** can be connected.

```
gds -d /data/ -p 192.168.0.90:5000 -H 10.10.0.1/24
```

Data files are stored in the **/data/** directory, the IP address of the directory is **192.168.0.90**, and the listening port number is **5000**. Only the node whose IP address is **10.10.0.*** can be connected to. The node communicates with the cluster using the SSL authentication mode, and the certificate files are stored in the **/certfiles/** directory.

```
gds -d /data/ -p 192.168.0.90:5000 -H 10.10.0.1/24 --enable-ssl --ssl-dir /certfiles/
```

NOTE

- One GDS provides the import and export services for one cluster only at a time.
- For security purpose, specify the IP address and the listening port through **-p**.
- The certificate file includes the root certificate **cacert.pem**, level-2 certificate file **client.crt**, and private key file **client.key**.
- The password protection files **client.key.rand** and **client.key.cipher** are used when the system loading certificates.

5.5 gds_ctl.py

Context

gds_ctl.py can be used to start and stop **gds** if **gds.conf** has been configured.

Prerequisites

Run the following commands on Linux OS: You need to ensure that the directory structure is as follows before the execution:

|----gds

|----gds_ctl.py

|----config

|-------gds.conf

|-------gds.conf.sample

or

|----gds

|----gds_ctl.py

|-------gds.conf

|-------gds.conf.sample

Content of **gds.conf**:

```
<?xml version="1.0"?>
<config>
<gds name="gds1" ip="127.0.0.1" port="8098" data_dir="/data" err_dir="/err" data_seg="100MB"
err_seg="1000MB" log_file="./gds.log" host="10.10.0.1/24" daemon='true' recursive="true" parallel="32"></
gds>
</config>
```

Configuration description of **gds.conf**:

- name: tag name
- ip: IP addresses to be listened to
- port: Port number to be listened to

  Value range: an integer ranging from 1024 to 65535

  Default value: **8098**
- **data_dir**: data file directory
- **err_dir**: error log file directory
- **log_file**: log file path
- **host**: hosts that can be connected to the GDS.
- **recursive**: whether the data file directory is recursive

  Value range:
  - **true**: indicates the recursion data file directory.
  - **false**: indicates the data file directory is not recursive.
- **daemon**: specifies whether the service is running in DAEMON mode.

  Value range:
  - **true** indicates the server is running in the DAEMON mode.
  - **false** indicates the server is not running in the DAEMON mode.
- **parallel**: indicates the number of concurrently imported and exported working threads.

  The default number of concurrencies is **8** and the maximum number is **200**.

Syntax

```
gds_ctl.py [ start | stop all | stop [ ip: ] port | stop | status ]
```

Description

gds_ctl.py can be used to start or stop GDS if **gds.conf** is configured.

Parameter Description

- start

  Enable the GDS configured in **gds.conf**.

- stop

  Stop the running instance started by the configuration file in the GDS that can be disabled by the current users.

- stop all

  Stop all the running instances in the GDS that can be disabled by the current users.

- stop [ip:] port

  Stop the specific running GDS instance that can be closed by the current user. If **ip:port** is specified when the GDS is started, stop the corresponding **ip:port** to be specified. If the IP address is not specified when the GDS is started, you need to stop the specified port only. The stop fails if different information is specified when the GDS is started or stopped.

- status

  Query the running status of the GDS instance started by the **gds.conf**.

Examples

Start the GDS.

```
python3 gds_ctl.py start
```

Stop the GDS started by the configuration file.

```
python3 gds_ctl.py stop
```

Stop all the GDS instances that can be stopped by the current user.

```
python3 gds_ctl.py stop all
```

Stop the GDS instance specified by **[ip:]port** that can be stopped by the current user.

```
python3 gds_ctl.py stop 127.0.0.1:8098
```

Query the GDS status.

```
python3 gds_ctl.py status
```

5.6 Handling Import Errors

Scenarios

Handle errors that occurred during data import.

Querying Error Information

Errors that occur when data is imported are divided into data format errors and non-data format errors.

- Data format error

  When creating a foreign table, specify **LOG INTO** *error_table_name*. Data format errors occurring during the data import will be written into the specified table. You can run the following SQL statement to query error details:


```
SELECT * FROM error_table_name;
```

Table 5-2 lists the columns of the *error_table_name* table.

Table 5-2 Columns in the *error_table_name* table

Column	Type	Description
nodeid	integer	ID of the node where an error is reported
begintime	timestamp with time zone	Time when a data format error is reported
filename	character varying	Name of the source data file where a data format error occurs If you use GDS for importing data, the error information includes the IP address and port number of the GDS server.
rownum	bigint	Number of the row where an error occurs in a source data file
rawrecord	text	Raw record of the data format error in the source data file
detail	text	Error details

- Non-data format error

  A non-data format error leads to the failure of an entire data import task. You can locate and troubleshoot a non-data format error based on the error message displayed during data import.

Handling data import errors

Troubleshoot data import errors based on obtained error information and the description in the following table.

Table 5-3 Handling data import errors

Error Information	Cause	Solution
missing data for column "r_reason_desc"	1. The number of columns in the source data file is less than that in the foreign table. 2. In a TEXT format source data file, an escape character (for example, \) leads to delimiter or quote mislocation. Example: The target table contains three columns as shown in the following command output. The escape character (\) converts the delimiter (|) into the value of the second column, causing loss of the value of the third column. `BE|Belgium|1`	1. If an error is reported due to missing columns, perform the following operations: • Add the **r_reason_desc** column to the source data file. • When creating a foreign table, set the parameter **fill_missing_fields** to **on**. In this way, if the last column of a row in the source data file is missing, it is set to **NULL** and no error will be reported. 2. Check whether the row where an error occurred contains the escape character (\). If the row contains such a character, you are advised to set the parameter **noescaping** to **true** when creating a foreign table, indicating that the escape character (\) and the characters following it are not escaped.
extra data after last expected column	The number of columns in the source data file is greater than that in the foreign table.	• Delete the unnecessary columns from the source data file. • When creating a foreign table, set the parameter **ignore_extra_data** to **on**. In this way, if the number of columns in a source data file is greater than that in the foreign table, the extra columns at the end of rows will not be imported.

Error Information	Cause	Solution
invalid input syntax for type numeric: "a"	The data type is incorrect.	In the source data file, change the data type of the columns to be imported. If this error information is displayed, change the data type to **numeric**.
null value in column "staff_id" violates not-null constraint	The not-null constraint is violated.	In the source data file, add values to the specified columns. If this error information is displayed, add values to the **staff_id** column.
duplicate key value violates unique constraint "reg_id_pk"	The unique constraint is violated.	• Delete the duplicate rows from the source data file. • Run the **SELECT** statement with the **DISTINCT** keyword to ensure that all imported rows are unique. **INSERT INTO** reasons **SELECT DISTINCT * FROM** foreign_tpcds_reasons;
value too long for type character varying(16)	The column length exceeds the upper limit.	In the source data file, change the column length. If this error information is displayed, reduce the column length to no greater than 16 bytes (VARCHAR2).

6 DSC

6.1 About This Document

6.1.1 Intended Audience

This document is intended for the following DSC users:

- Database migration engineers
- Database administrators
- Technical support engineers

DSC users need to have a basic knowledge of:

- Basic concepts and policies of database migration
- Teradata/MySQL(ADB For MySQL)
- GaussDB(DWS)

6.1.2 Document Conventions

This section describes the content, symbol, and command conventions of this document.

Disclaimer

The purchased products, services, and features are subject to the signed contract. All or part of the products, services, and features described in this document may not be available for purchasing or using. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided in the form of "as-is" and do not constitute a warranty, guarantee, or representation of any kind, express or implied.

Symbol Convention

Symbols that may be found in this document are defined as follows:

Symbol	Description
DANGER	Indicates an imminently hazardous situation which, if not avoided, will result in serious injury or death.
CAUTION	Indicates a potentially hazardous situation which, if not avoided, will result in death or serious injury.
NOTICE	Indicates a potentially hazardous situation which, if not avoided, may result in minor or moderate injury.
NOTICE	Indicates a potentially hazardous situation which, if not avoided, could result in equipment damage, data loss, performance deterioration, or unanticipated results. NOTICE is used to address practices not related to personal injury.
NOTE	Provides additional information to emphasize or supplement important points of the main text. **NOTE** is used to address information not related to personal injury, equipment damage, or environment damage.

Command Conventions

Commands in this document comply with the following conventions:

Table 6-1 Command conventions

Format	Description
Boldface	The keywords of a command are in **Boldface**.
Italic	Command arguments, paths, files, or folders are in italics.
[ ]	Square brackets enclose syntax choices (keywords or arguments) that are optional.
{ x | y | ... }	Braces enclose a set of required syntax choices separated by vertical bars (|), from which you select one.
[x | y | ...]	Square brackets enclose a set of optional syntax choices separated by vertical bars, from which you select one or none.
{ x | y | ... }*	Asterisk marked braces enclose a set of required syntax choices separated by vertical bars, from which you select at least one.

Format	Description
[x | y | ...]*	Asterisk marked square brackets enclose optional syntax choices separated by vertical bars, from which you select one choice, multiple choices, or none.
&<1-n>	The argument or keyword and argument combination before the ampersand (&) sign can be entered 1 to n times.
#	A line that starts with a pound (#) sign is comments.

6.1.3 Third-party Licenses

This section lists the applicable third-party licenses.

- ANTLR v4.9.3
- Apache Commons IO 2.11
- Apache Commons CLI 1.5
- Apache Log4j 2.17.2
- JSON.org json 20220320
- postgresql 42.4.1
- sql-formatter 2.0.3

6.2 Introduction to DSC

6.2.1 Overview

After switching to GaussDB(DWS) databases, you may need to migrate user data and application SQL scripts to new databases. The migration of application SQL scripts is complex, risky, and time-consuming.

DSC is a CLI tool running on Linux or Windows. It provides users with simple, fast, and reliable application SQL script migration services. It parses the SQL scripts of source database applications using the built-in syntax migration logic, and converts them to SQL scripts applicable to GaussDB(DWS) databases.

DSC does not require a connection to databases, and performs migration offline. The tool also displays the status of a migration process and logs the errors that occur during the process, helping quickly locate faults.

Migration Objects

DSC can migrate the following Teradata and MySQL database objects:

- Common objects supported by Teradata and MySQL: SQL schemas and SQL queries

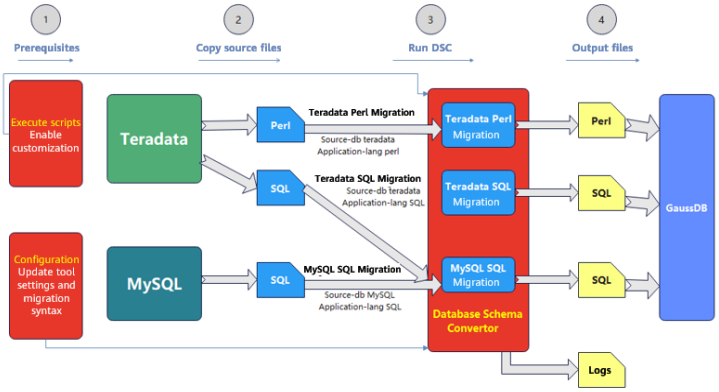
- Objects supported only by Teradata: Perl files containing **BTEQ** and **SQL_LANG** scripts

Migration Process

The process of using DSC to migrate SQL scripts is as follows:

1. Export the SQL scripts from a Teradata or MySQL database to the Linux or Windows server installed with DSC.
2. Execute DSC to migrate the syntax. In the command, specify the paths of the input file, output file, and logs.
3. DSC automatically archives the migrated SQL scripts and the logs into the specified paths.

Figure 6-1 DSC migration process



6.2.2 Operating Environment

Supported Databases

The following table lists the source databases supported by DSC.

Table 6-2 Source databases supported by DSC

Database	Version
Teradata	13.1
MySQL	8.0

The following table lists the destination databases supported by DSC.

Table 6-3 Destination databases supported by DSC

Database	Version
GaussDB(DWS)	GaussDB(DWS) 8.1.0 or later

Hardware Requirements

Table 6-4 lists the hardware requirements of DSC.

Table 6-4 Hardware requirements of DSC

Hardware	Configuration
CPU	AMD or Intel Pentium (minimum frequency: 500 MHz)
Min memory	1 GB
Disk space	1 GB

Software Requirements

OS

Table 6-5 lists the operating systems compatible with DSC.

Table 6-5 OSs compatible with DSC

Server	OS	Version
x86 servers	SUSE Linux Enterprise Server 11	SP1 (SUSE 11.1)
		SP2 (SUSE 11.2)
		SP3 (SUSE 11.3)
		SP4 (SUSE 11.4)
	SUSE Linux Enterprise Server 12	SP0 (SUSE 12.0)
		SP1 (SUSE 12.1)
		SP2 (SUSE 12.2)
		SP3 (SUSE 12.3)
	RHEL	6.4-x86_64 (RHEL 6.4)
		6.5-x86_64 (RHEL 6.5)
		6.6-x86_64 (RHEL 6.6)
		6.7-x86_64 (RHEL 6.7)

Server	OS	Version
		6.8-x86_64 (RHEL 6.8)
		6.9-x86_64 (RHEL 6.9)
		7.0-x86_64 (RHEL 7.0)
		7.1-x86_64 (RHEL 7.1)
		7.2-x86_64 (RHEL 7.2)
		7.3-x86_64 (RHEL 7.3)
		7.4-x86_64 (RHEL 7.4)
	CentOS	6.4 (CentOS 6.4)
		6.5 (CentOS 6.5)
		6.6 (CentOS 6.6)
		6.7 (CentOS 6.7)
		6.8 (CentOS 6.8)
		6.9 (CentOS 6.9)
		7.0 (CentOS 7.0)
		7.1 (CentOS 7.1)
		7.2 (CentOS 7.2)
		7.3 (CentOS 7.3)
		7.4 (CentOS 7.4)
	Windows	7.0, 10, 11

Other software requirements

Table 6-6 lists the requirements of DSC on other software versions.

Table 6-6 Requirements of DSC on other software versions

Software	Description
JDK 1.8.0_141 or later	Used to run DSC.
Perl 5.8.8	Used to migrate Perl files.
Perl 5.28.2 and later	Used to migrate Perl files in Windows.
Python 3.8.2	Used to verify post migration script.

6.3 Using DSC

6.3.1 Overview

This chapter describes how to use DSC, including how to install and configure DSC and how to use DSC to migrate data.

NOTICE

Use the latest patches to update the operating system and related software to prevent vulnerabilities and other security issues.

Security in DSC is managed by access control to the files and folders that are created by the tool. To access these files and folders, you must have the required permissions. For example, you need the permission 600/400 to access target files and log files, and the permission 700 to access target folders and log folders. The tool also ensures data security by not saving sensitive data in the log files.

The file or folder specified in **--input-folder** must not have write privileges to GROUP and/or OTHERS. For security reasons, the tool will not execute if the input files/folders have write privileges.

It is required that the root privileged user must not be used for installation and execution of the DSC for Linux.

The **umask** value provided in the **DSC** file is a set value that is related to file permissions. It is recommended that users do not modify this value. Modifying this value will affect file permissions.

NOTE

DSC is a standalone application that can run without any network or database connection. It can run on any machine that is isolated from any network.

6.3.2 Downloading and Installing DSC

Before using DSC, install it on a Linux or Windows server. DSC supports 64-bit Linux OSs. For details about other OSs supported by DSC, see **Table 6-5**.

Prerequisites

- In Linux, do not install or perform operations on DSC as the **root** user. To execute the **install.sh** script, you must have the permission for creating folders.
- The size of the target folder must be at least four times that of the SQL files in the input folder.

  For example, if the size of the SQL files in the input folder is 100 KB, the target folder must have available disk space of at least 400 KB to process the SQL files.

NOTE

- To query the available disk space of the target folder in Linux, run the following command:
  ```
  df -P <Folder path>
  ```
- To query the size of the input file in Linux, run the following command in the directory where the file resides:
  ```
  ls -l
  ```

- JRE 1.8 or later and Perl have been installed. For details about the hardware and software requirements, see **Operating Environment**.

  To check the installed Java version and set the Java path, perform the following steps:

  a. Check whether the Java version meets requirements.
  ```
  java -version
  ```

  b. Ensure that the Java path is correctly set.

  - Linux

    1) Check whether the Java path is correctly set.
    ```
    echo $JAVA_HOME
    ```

    2) If no information is returned, add the following two lines to the **.bashrc** file of the current user, save the modification, and exit.

    Assume that the Java installation path is **/home/user/Java/jdk1.8.0_141**.
    ```
    export JAVA_HOME=/home/user/Java/jdk1.8.0_141
    export PATH=$JAVA_HOME/bin:$PATH
    ```

    3) Activate Java environment variables.
    ```
    source ~/.bashrc
    ```

  - Windows

    1) Right-click **My Computer** and choose **Properties** from the shortcut menu. The **System** window is displayed.

    2) Click **Advanced System Settings**. The **System Properties** dialog box is displayed.

    3) On the **Advanced** tab page, click **Environment Variables**. The **Environment Variables** dialog box is displayed.

    4) Select **Path** in the **System variables** area. Click **Edit** and check the Java installation path.

    If the Java installation path does not exist or is incorrect, add the Java path of this PC to **Path**.

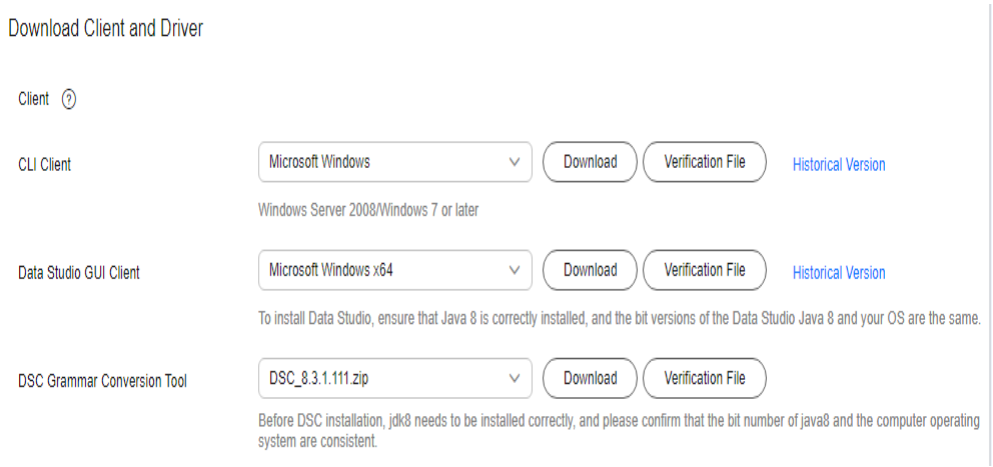
    Assume that the Java installation path is **C:\Program Files\Java\jdk1.8.0_141\bin** and the Path environment variable is *c:|windows|system32;*, the path must be set to *c:|windows|system32;***C:\Program Files\Java\jdk1.8.0_141\bin;**.

Downloading DSC

Step 1 Log in to the GaussDB(DWS) console . In the navigation tree on the left, choose **Management** > **Client Connections**.

Step 2 On the **Download Client and Driver** page, click **here** to download the **DSC** software package.

Figure 6-2 Downloading the DSC client



Step 3 Decompress the downloaded client software package to a user-defined path.

----End

Installing DSC

DSC is a command line tool running on the Linux or Windows operating system. It can be used without installation. After downloading the software package, you can decompress it to use it.

Windows:

Step 1 Decompress the **DSC.zip** package.

The **DSC** folder is generated.

NOTE

You can decompress **DSC.zip** to any folder you need.

Step 2 Go to the **DSC** directory.

Step 3 Find and check the files in the **DSC** directory.

Table 6-7 describes the obtained folders and files.

----End

Linux:

Step 1 Extract files from **DSC.zip**.

```
sh install.sh
```

Step 2 Go to the **DSC** directory.

```
cd DSC
```

Step 3 Check the files in the **DSC** directory.


```
ls
config   lib   scripts   bin  input output runDSC.sh  runDSC.bat
```

----End

Table 6-7 DSC directory

Folder or File		Description
DSC	bin	DSC-related JAR package (executable)
	config	Configuration file of DSC
	input	Input folder
	lib	Library files required for the normal running of DSC
	output	Output folder
	scripts	Customized configuration scripts for Oracle and Teradata migration, which can be executed to implement corresponding functions
	runDSC.sh	Application executed on the Linux OS
	runDSC.bat	Application executed on the Windows OS
changelog		To notify users of the current modifications
Install.sh		To set the file permissions for DSC
readme.txt		Instructions of installation and configuration

NOTE

If you do not need DSC, you can uninstall it by deleting the **DSC** folder.

6.3.3 Configuring DSC

6.3.3.1 DSC Configuration

Configure the following items:

- **Setting application.properties**: Configure the migration behavior of DSC, for example, whether to overwrite the files in the target folder and whether to format the SQL files.
- **Setting Java Memory Allocation**: Configure the memory that can be used by DSC. If the memory usage exceeds the threshold, DSC displays an error and exits.

Setting application.properties

The **application.properties** file contains DSC behavior parameters, which are applicable to Teradata, Oracle.

Perform the following steps to configure the parameters:

Step 1 Open the **application.properties** file in the **config** folder.

Step 2 Set parameters in the **application.properties** file as needed.

Table 6-8 describes the parameters in the **application.properties** file.

NOTE

- Parameter values are case-insensitive.
- Do not modify any other parameter except the listed ones.

Step 3 Save the configuration and exit.

----End

Table 6-8 Parameters in the application.properties file

Parameter	Description	Value Range	Default Value	Example
• formatterrequired	If there are any changes in this configuration item, then the tool will not function as expected.	• true • false	true	formatterrequired=true
• prevalidationFlag	If there are any changes in this configuration item, then the tool will not function as expected.	• true • false	true	prevalidationFlag=true
• commentSeparatorFlag	If there are any changes in this configuration item, then the tool will not function as expected.	• true • false	true	commentSeparatorFlag=true

Parameter	Description	Value Range	Default Value	Example
• queryDelimiter	If there are any changes in this configuration item, then the tool will not function as expected.	N/A	N/A	queryDelimiter=;
• blogicDelimiter	If there are any changes in this configuration item, then the tool will not function as expected.	N/A	N/A	blogicDelimiter=/
• Timeout	Tool migration timeout duration. If there are any changes in this configuration item, then the tool will not function as expected.	-	4 hours	Timeout=4

Parameter	Description	Value Range	Default Value	Example
• fileExtension	Valid file extension separated by comma. If there are any modifications in this configuration item, then the tool will not function as expected. **NOTE** The exported scripts must be with the following postfix such as: • .sql • .txt • .fnc • .proc • .tbl • .tbs • .pl • .dsql and so on.	• csv • txt • SQL	SQL	fileExtension=SQL
• formattedSourceRequired	Whether to use SQL Formatter to format the source SQL files. If this parameter is set to **true**, the copies of the input files are formatted and saved to the *Output path*/**formattedSource** directory.	• true • false	true	formattedSourceRequired=true

Parameter	Description	Value Range	Default Value	Example
• target_files	Specifies the operations to be performed on the output folder. **Overwrite**: This option specifies whether the files in the output folder must be overwritten. **Delete**: deletes all files in the output folder. **Cancel**: cancels an operation on the output folder that contains files.	• overwrite • delete • cancel	overwrite	target_files=overwrite
• encodingFormat	Encoding format of input files. If this parameter is not set (or is commented out), DSC uses the default encoding format based on locale settings. **NOTE** • The auto detection of file encoding is inaccurate. To ensure that the correct encoding format is used, specify the format using this parameter.	• UTF8 • UTF16 • UTF32 • GB2312 • ASCII and others	Default based on locale	encodingFormat=UTF8

Parameter	Description	Value Range	Default Value	Example
• NoOfThreads	Number of threads used for migration	Depending on available system resources	3	NoOfThreads=3
• MaxFileSizeWarning	Warning threshold for the input file size (unit: KB, MB, or GB). If an invalid value is specified, the default value is used. The following warning will be displayed if the specified source file size exceeds the threshold: ********** [WARNING] : Migration of the following files(>100KB) will take more time: bigfile001.SQL bigfile008.SQL **********	10 KB~1 GB	10MB	MaxFileSizeWarning=10MB
• MaxFileSize	Maximum size of the input file allowed. If crossing this limit, the file migration will be skipped.	-	20MB	MaxFileSize=20MB

Parameter	Description	Value Range	Default Value	Example

• MaxSqlLen	Maximum size of a query to be migrated.
	If an invalid value is specified, DSC resets it to the default value and displays the following warning:
	`The query length parameter (MaxSqlLen) value is out of range. Resetting to default value.`
	If an input query exceeds the specified maximum length, the pre-validation of the query migration will fail. DSC skips this query and logs the following error:
	`2018-07-06 12:05:57,598 ERROR TeradataBulkHandler:195 Error occurred during processing of input in Bulk Migration. PreQueryValidation failed due to: Invalid termination; OR exclude keyword found in query; OR query exceeds maximum length (MaxSqlLen config parameter). filename.SQL for Query in position : xx`
		1 .. 52,428,800 bytes
		(1 byte to 50 MB)
			1048576
			(1 MB)
				MaxSqlLen=1048576

• initialJVMMemory	Initial memory	N/A	256 MB	initialJVMMemory=256MB This indicates that the process will start up with 256 MB of memory

Parameter	Description	Value Range	Default Value	Example
• maxJVMMemory	Maximum memory	N/A	1024 MB	maxJVMMemory=2048m This indicates that the process will use up to 2048 MB of memory.
• executesqlingauss	Runs the migrated scripts in GaussDB(DWS). The value can be true or false. The script can be executed only on the server running the Linux operating system.	• true • false	false	executesqlingauss=false

NOTE

- If a parameter is set to an incorrect or invalid value, DSC uses the default value of the parameter.
- If the extensions (for example, **.doc**) are not supported, then it is recommended you add extension in **fileExtension** configuration parameter in **application.properties** file.

Setting Java Memory Allocation

DSC has preset settings for the memory allocation of the Java Virtual Machine (JVM).

If the memory usage exceeds the limit during migration, DSC displays the "java.lang.OutOfMemoryError: GC overhead limit exceeded" error and exit. In this case, you can increase the values of **initialJVMMemory** and **maxJVMMemory** in the **application.properties** file to allocate more memory.

NOTE

The available system resources also determine the memory allocation.

Table 6-9 Parameters for JVM memory allocation

Parameter	Description	Recommended Value
Xms	Initial memory allocation (unit: MB)	The minimum value is 256 MB. The maximum value depends on the available system resources. Default value: **256**
Xmx	Upper limit for memory allocation (unit: MB)	The minimum value is 1024 MB. The maximum value depends on the available system resources. Default value: **1024**

Open the **gaussdb.properties** file in the **config** folder of the verification file and configure parameters by referring to **Table 6-10** to connect to GaussDB(DWS).

Table 6-10 Parameters in the gaussdb.properties file

Parameter	Description	Value Range	Default	Example
gaussdb-user	GaussDB(DWS) database user who has all permissions	N/A	N/A	user1
gaussdb-port	Port of the GaussDB(DWS) database	N/A	N/A	8000
gaussdb-name	Name of the GaussDB(DWS) database	N/A	N/A	gaussdb
gaussdb-ip	IP address of the GaussDB(DWS) database	N/A	N/A	10.XX.XX.XX

6.3.3.2 Teradata SQL Configuration

Teradata parameters are used to customize rules for Teradata script migration.

Open the **features-teradata.properties** file in the **config** folder and set the parameters in **Table 6-11** as required.

Table 6-11 Parameters in the **features-teradata.properties** file

Parameter	Description	Value Range	Default Value	Example
• deleteToTruncate	Rule for migrating **DELETE** statements without a **WHERE** clause. **true**: enables the migration of **DELETE** to **TRUNCATE**. **false**: disables the migration of **DELETE** to **TRUNCATE**.	• true • false	false	deleteToTruncate=true
• distributeByHash	Which columns specified in the primary index will be used for data distribution across nodes in the cluster. **one**: Data is distributed based on the first column specified in the primary index. **many**: Data is distributed based on all the columns specified in the primary index. This function is addressed by using the **DISTRIBUTE BY** clause. **NOTE** This parameter is set to **one** in V100R002C60 because this version does not support multiple columns in the **DISTRIBUTE BY** clause.	• one • many	many	distributeByHash =many

Parameter	Description	Value Range	Default Value	Example
• extendedGroupByClause	This parameter is used to enable or disable **Group By** (grouping sets/cube/rollup) migration. **true**: enables the migration of **GROUP BY()**. **false**: disables the migration of **GROUP BY()**.	• true • false	false	extendedGroupByClause=false
• inToExists	This parameter can be used to enable and disable query optimization from **IN/NOT IN** to **EXISTS/NOT EXISTS**.	• true • false	false	inToExists=false
• rowstoreToColumnstore	Whether to convert row-store tables to **column-store** tables. If this parameter is set to **true**, all row-store tables are converted to column-store tables during script migration.	• true • false	false	rowstoreToColumnstore=false
• session_mode	Default table type (**SET**/**MULTISET**) for **CREATE TABLE**. **Teradata**: The default table type is **SET**. **ANSI**: The default table type is **MULTISET**.	• Teradata • ANSI	Teradata	session_mode=ANSI
• tdMigrateALIAS	Whether to enable the migration of **ALIAS**. **true**: enables the migration of **ALIAS**. **false**: disables the migration of **ALIAS**.	• true • false	false	tdMigrateALIAS=true

Parameter	Description	Value Range	Default Value	Example
• tdMigrateDOLLAR	Whether to enable the migration of static objects whose name starts with a dollar sign ($). This parameter is not applicable to dynamic objects, in format of ${}. **true**: Enclose the name of a static object with double quotation marks ("). **false**: Disable the migration of static objects. **NOTE** For details, see **Object name starting with $**.	• true • false	true	tdMigrateDOLLAR=true
• tdMigrateLOCKoption	Whether to enable the migration of queries with the **LOCK** keyword. **true**: enables the migration of such queries and comments out the **LOCK** keyword. **false**: disables the migration of such queries. DSC Migration Tool skips this query and logs the following information: Gauss does not have equivalent syntax for LOCK option in CREATE VIEW and INSERT statement. Please enable the config_param tdMigrateLockOption to comment the LOCK syntax in the statement. **NOTE** For details, see **ACCESS LOCK**.	• true • false	false	tdMigrateLOCKoption=true

Parameter	Description	Value Range	Default Value	Example
• tdMigrateNULLIFZERO	Whether to enable the migration of **ZEROIFNULL()**. **true**: enables the migration of **NULLIFZERO()**. **false**: disables the migration of **NULLIFZERO()**.	• true • false	true	tdMigrateNullIFZero=true
• tdMigrateVIEWCHECKOPTION	Whether to enable the migration of views containing **CHECK OPTION**. **true**: comments out such views during migration. If this parameter is set to **false**, such views are not migrated. The tool will copy the query as it is and record the following message: Gauss does not support WITH CHECK OPTION in CREATE VIEW. Please enable the config_param tdMigrateViewCheckOption to comment the WITH CHECK OPTION syntax in the statement.	• true • false	false	tdMigrateVIEWCHECKOPTION=true
• tdMigrateZEROIFNULL	Whether to enable the migration of **ZEROIFNULL**. **true**: enables the migration of **ZEROIFNULL()**. **false**: disables the migration of **ZEROIFNULL()**.	• true • false	true	tdMigrateZEROIFNULL=true

Parameter	Description	Value Range	Default Value	Example
• volatile	Type of tables whose data is specific to a session and is stored only for the session. When the session ends, the data and tables are deleted. A volatile table can be a **Migrating Tables** table or an unlogged table. **NOTE** **unlogged** is supported in V100R002C60 and **local temporary** is not.	• local temporary • unlogged	local temporary	volatile=unlogged
• tdMigrateCharsetCase	Whether to enable the migration of **CHARACTER SET** and **CASESPECIFIC**. **true**: comments out **CHARACTER SET** and **CASESPECIFIC** during script migration. **false**: disables the migration of **CHARACTER SET** and **DSC**. In this case, DSC copies **CHARACTER SET**, **CASESPECIFIC** and logs the following information with query details (such as the file name and statement position): Gauss does not have an equivalent syntax for CHARACTER SET & CASE SPECIFIC option in column-level. You can rewrite this statement or set the configuration parameter tdMigrateCharsetCase to TRUE to comment the Character set & Case specific syntax in this statement.	• true • false	false	tdMigrateCharsetCase=false **NOTE** If **tdminatecharsetcase** is set to **true**, comment out the special keyword of the character.

Parameter	Description	Value Range	Default Value	Example
• terdataUtilities	Specifies whether the Teradata command line tool can be migrated. The following options are supported: • true • false	• true • false	true	terdataUtilities=true
• unique_primary_index_in_column_table	Specifies whether unique indexes can be created for column-store tables.	• true • false	true	unique_primary_index_in_column_table=true
• default_charset	Specifies whether default_charset can be migrated. The following options are supported: • LATIN • UNICODE • GRAPHIC	• LATIN • UNICODE • GRAPHIC	LATIN	default_charset=LATIN
• mergeImplementation	**mergeImplementation** has the following two types: • using WITH clause • splitting the queries The following options are supported: • With • Split • None	• With • Split • None	None	mergeImplementation=None
• dsqlSupport	Specifies whether dsql is supported. The following options are supported: • true • false	• true • false	false	dsqlSupport=false

Parameter	Description	Value Range	Default Value	Example
• tdcolumnInSensitive	Whether to remove column names that contain double quotes during migration. The following options are supported: • true • false	• true • false	false	tdcolumnInSensitive=false
• tdMigrateCASE_N	Specifies the migration mode of the CASE_N for partitioning. Gauss does not support multilevel (nested) partitioning: The following options are supported: • comment • none	• comment • none	comment	tdMigrateCASE_N=comment
• tdMigrateRANGE_N	Specifies the migration mode of the RANGE_N for partitioning. Gauss does not support multilevel (nested) partitioning: The following options are supported: • comment • none • range	• comment • none • range	range	tdMigrateRANGE_N=range

Parameter	Description	Value Range	Default Value	Example
• tdMigrateAddMonth	Specifies whether addMonth can be migrated. The following options are supported: • true • false If this parameter is set to **true**, **ADD_MONTHS** changes into **mig_td_ext.ADD_MONTHS** (added **mig_td_ext**) after the migration. Otherwise, migration is not supported.	• true • false	false	tdMigrateAddMonth=false

6.3.3.3 Teradata Perl Configuration

Teradata Perl parameters are used to customize rules for Teradata Perl script migration.

Open the **perl-migration.properties** file in the **config** folder and set parameters in **Table 6-12** as required.

NOTE

- Parameter values are case-insensitive.
- You can modify the values of the following two parameters **db-bteq-tag-name** and **db-tdsql-tag-name** parameters in the following table:

Table 6-12 Parameters in the perl-migration.properties file

Parameter	Description	Value Range	Default	Example
• db-bteq-tag-name	Scripts to be processed in Perl files. **BTEQ**: Only the scripts under the **BTEQ** tag are processed.	• bteq	bteq	db-bteq-tag-name=bteq
• db-tdsql-tag-name	Only the scripts under the **db-tdsql-tag-name** tag are processed. **SQL_LANG:** Only the scripts under the **SQL_LANG** tag are processed.	sql_lang	sql_lang	db-tdsql-tag-name=sql_lang
• add-timing-on	Whether to enable the insertion of scripts to calculate execution time. If it is enabled, the script will be added to each input file.	• true • false	false	add-timing-on=true

Parameter	Description	Value Range	Default	Example
• remove-intermediate-files	Whether to delete the intermediate SQL file generated by the DSC after the migration is complete. The intermediate files contain the **BTEQ** and **SQL_LANG** syntax in SQL files. These files are used as input for DSC. **true**: deletes the intermediate files. **false**: does not delete the intermediate files.	• true • false	true	remove-intermediate-files=true

Parameter	Description	Value Range	Default	Example

• migrate-variables	Whether to enable the migration of Perl variables containing SQL statements.
	Perl files can contain Perl variables with SQL statements. These variables are executed by using the **PREPARE** and **EXECUTE** statement in Perl. DSC can extract SQL statements from Perl variables and migrate them.
	true: enables the migration of **Perl** variables containing SQL statements.
	false: disables the migration of **Perl** variables containing SQL statements.
	Example 1:
	If **migrate-variables** is set to **true** and the target table is:
	`$V_SQL = "CT X1(C1 INT,C2 CHAR(30))";`
	Output
	`CREATE TABLE X1(C1 INT,C2 CHAR(30));`
	Example 2:
	Input
	`$onesql ="SELECT trim(tablename) from dbc.tables WHERE databasename = '${AUTO_DQDB}' and`
		• true
		• false
			true	migrate-variables=true

Parameter	Description	Value Range	Default	Example
	tablename like 'V_%' order by 1;''; $sth_rundq = $dbh->execute_query($onesql); **Output** $onesql =''SELECT TRIM(tablename) FROM dbc.tables WHERE databasename = '${AUTO_DQDB}' AND tablename LIKE 'V_%' ORDER BY 1 ; ''; $sth_rundq = $dbh->execute_query($onesql);			
• logging-level	Logging level of Teradata Perl migration log files. **error**: Log only errors. **warning**: Log errors and warnings. **info**: Log errors, warnings, and activity information. This level contains all log information.	• error • warning • info	info	logging-level=info

Parameter	Description	Value Range	Default	Example
• log-file-count	Maximum number of log files retained, including the log files in use and archived log files. If the number of log files exceeds the upper limit, the earliest files will be deleted until the new log files are successfully archived.	3 - 10	5	log-file-count=10
• log-file-size	Maximum file size. Upon reaching the specified size, a file is archived by adding a timestamp to the file name. **Example:** perlDSC_2018-07-08_16_12_08.log After the archiving, a new log file *perlDSC.log* with a timestamp is generated.	1MB - 10MB	5MB	log-file-size=10MB

Parameter	Description	Value Range	Default	Example

• migrate-executequery	Whether to enable the migration of **execute_query** containing SQL statements.
	true: enables the migration of **execute_query** containing SQL statements.
	false: disables the migration of **execute_query** containing SQL statements.
	The following is an example:
	migrate-executequery is set to **true** and input is as follows:
	`my $rows1=$conn1->execute_query("sel ${selectclause} from ${databasename}.${tablename};");`
	Output
	`my $rows1=$conn1->execute_query("SELECT ${selectclause} FROM ${databasename}.${tablename};");`
		• true
		• false
			true	migrate-executequery=true

6.3.3.4 MySQL Configuration

MySQL parameters are used to customize rules for MySQL script migration.

Open the **features-mysql.properties** file in the **config** folder and configure **Parameters in the features-mysql.properties file** as required.

Table 6-13 Parameters in the **features-mysql.properties** file

Parameter	Description	Value Range	Default Value	Example
• table.databaseAsSchema • table.defaultSchema	Whether to use a database name as the schema name. If a database name does not exist, the user-defined schema specified by **table.schema** will be used. If **table.schema** is not specified, the default schema will be used.	• true • false • public	• true • public	• table.databaseAsSchema=true • table.defaultSchema=public
• table.schema	Name of the user-defined schema. If this parameter is specified, it will be directly used. In this case, even if **useDatabaseAsSchema** is set to **true**, the name of the current schema will be used.	• schemaName	• null	• table.schema=

Parameter	Description	Value Range	Default Value	Example
• table.orientation	Default data storage mode. Value **ROW** means row storage, and value **COLUMN** means column storage.	• ROW • COLUMN	• ROW	• table.orientation=ROW
• table.type	Default table type, which can be partitioned table, replication table, or round-robin table. **REPLICATION**, **HASH**, and **ROUND-ROBIN**	• HASH • REPLICATION • ROUND-ROBIN	• HASH	• table.type=HASH
• table.tablespace	Tablespace options	• COMMENT • RESERVE	• RESERVE	• table.tablespace=RESERVE
• table.partition-key.choose.strategy	Policy for selecting the partition key	• partitionKeyChooserStrategy	• partitionKeyChooserStrategy	• table.partition-key.choose.strategy=partitionKeyChooserStrategy

Parameter	Description	Value Range	Default Value	Example
• table.partition-key.name	Partition key. If this parameter is not specified, the default selection policy will be used. If there are multiple columns, separate them with commas (,). The column names are case insensitive.	• Reserved parameter	• null	• table.partition-key.name=
• table.compress.mode	Compression mode. If keyword **COMPRESS** is specified in **CREATE TABLE**, the compression feature will be triggered in the case of bulk **INSERT** operations. If this feature is enabled, a scan will be performed on all tuple data within the page to generate a dictionary and then the tuple data will be compressed and stored. If **NOCOMPRESS** is used, tables will not be compressed.	• COMPRESS • NOCOMPRESS	• NOCOMPRESS	• table.compress.mode=NOCOMPRESS

Parameter	Description	Value Range	Default Value	Example
• table.compress.row • table.compress.column	Compression level of table data. It determines the compression ratio and duration.	• YES • NO • YES • NO • LOW • MIDDLE • HIGH	• NO • LOW	• table.compress.row=NO • table.compress.column=LOW
• table.compress.level	Table data compression ratio and duration at the same compression level. This divides a compression level into sublevels, providing more choices for the compression ratio and duration. As the value becomes larger, the compression ratio becomes higher and duration longer at the same compression level.	• 0 • 1 • 2 • 3	• 0	• table.compress.level=0
• table.database.template	Database template	• Reserved parameter	• template0	table.database.template=template0
• table.database.encoding	A database code.	• UTF8 • SQL_ASCII • GBK • Latin1 codes	• UTF8	table.database.encoding=UTF8

Parameter	Description	Value Range	Default Value	Example
• table.index.rename	Whether to rename an index during its creation.	• true • false	false	table.index.rename=false
• table.database.onlyFullGroupBy	Whether all non-aggregated columns in a **SELECT** statement are included in the **GROUP BY** clause.	• true • false	true	table.database.onlyFullGroupBy=true
• table.database.realAsFloat	This parameter allows you to convert the REAL data type. By default, it is set to **false**, which means that the data type is converted to **DOUBLE PRECISION**. However, if you set it to **true**, the data type will be converted to **REAL**.	• true • false	false	table.database.realAsFloat=false

6.3.3.5 Oracle SQL Configuration

Oracle parameters are used to customize rules for Oracle script migration.

Open the **features-oracle.properties** file in the **config** folder and set parameters in **Table 6-14** as needed.

Table 6-14 Parameters in the **features-oracle.properties** file

Parameter	Description	Value Range	Default Value	Example
• exceptionHandler	Whether to comment out exception blocks in PL/SQL. **true**: Comment out the exception blocks. **false**: Retain the exception blocks as they are. **NOTE** **exceptionHandler** is not supported in V100R002C60.	• true • false	false	exceptionHandler=TRUE
• TxHandler	Whether to comment out COMMIT and ROLLBACK operations in PL/SQL. **true**: Comment out the operations. **false**: Retain the operations as they are.	• True • False	True	TxHandler=True
• foreignKeyHandler	Whether to comment out foreign key constraints. **true**: Comment out the constraints. **false**: Retain the constraints as they are.	• true • false	true	foreignKeyHandler=true

Parameter	Description	Value Range	Default Value	Example
• globalTempTable	Valid values are **GLOBAL** and **LOCAL**. Currently, the target database does not support **GLOBAL**.	• GLOBAL • LOCAL	LOCAL	encodingFormat=LOCAL
• onCommitDeleteRows	Valid values are **DELETE** and **PRESERVE**. Current Version V100R008	• DELETE • PRESERVE	DELETE	onCommitDeleteRows=DELETE
• maxValInSequence	Maximum sequence value supported by the database. Currently, the maximum value supported by the database is **9223372036854775807**.	1-9223372036854775807	9223372036854775807	maxValInSequence=9223372036854775807
• mergeImplementation	Method for migrating a merge statement. **SPLIT**: The merge statement is split into individual queries during migration for query optimization. **WITH**: The merge statement is migrated using a WITH clause.	• WITH • SPLIT • None	WITH	mergeImplementation=None

Parameter	Description	Value Range	Default Value	Example
• RemoveHashPartition	Whether to comment out **HASH PARTITION** statements. **true**: Comment out the constraints. **false**: Retain the **HASH PARTITION** statements as they are.	• true • false	true	RemoveHashPartition=false
• RemoveHashSubPartition	Whether to comment out **HASH SUBPARTITION** statements. **true**: Comment out the constraints. **false**: Retain **HASH SUBPARTITION** statements as they are.	• true • false	true	RemoveHashSubPartition=false
• RemoveListPartition	Whether to comment out **LIST PARTITION** statements. **true**: Comment out the constraints. **false**: Retain **LIST PARTITION** statements as they are.	• true • false	true	RemoveListPartition=false

Parameter	Description	Value Range	Default Value	Example
• RemoveListSubPartition	Whether to comment out **LIST SUBPARTITION** statements. **true**: Comment out the constraints. **false**: Retain the **LIST SUBPARTITION** statements as they are.	• true • false	true	RemoveListSubPartition=false
• RemoveRangeSubPartition	Whether to comment out **RANGESUBPARTITION** statements. **true**: Comment out the constraints. **false**: Retain the **RANGESUBPARTITION** statements as they are.	• true • false	true	RemoveRangeSubPartition=false
• MigSupportSequence	Whether to enable the migration of **SEQUENCE** statements. **true**: enables the conversion of **CREATE** to **INSERT**. **false**: disables the migration of **CREATE**.	• true • false	true	MigSupportSequence=false

Parameter	Description	Value Range	Default Value	Example
• RemovePartitionTS	Whether to comment out **PartitionTS** statements. **true**: Comment out the **PartitionTS** statements. **false**: Retain the **PartitionTS** statements as they are.	• true • false	true	RemovePartitionTS=true
• BitmapIndexSupport	Whether to comment out for BitmapIndex **COMMENT** will comment the entire input script **BTREE** will retain as they are	• comment • btree	comment	BitmapIndexSupport=comment
• pkgSchemaNaming	The following options are supported: **TRUE** - schema1.package1#procedure1 should be changed to package1.procedure1 **FALSE** - schema1.package1#procedure1 will not be removed	• true • false	true	pkgSchemaNaming=true
• plsqlCollection	The following options are supported: • varray • localtable • none	• varray • localtable • none	varray	plsqlCollection=varray

Parameter	Description	Value Range	Default Value	Example
• commentstorageparameter	Whether to comment out the storage parameters in a table or index. **true**: Comment out the storage parameters. **false**: Retain the storage parameters as they are.	• true • false	true	commentStorageParameter=true
• MigSupportForListAgg	Whether to enable the migration of **ListAgg** statements. **true**: enables the migration of **ListAgg**. **false**: disables the migration of **ListAgg**.	• true • false	true	MigSupportForListAgg=false
• MigSupportForRegexReplace	Whether to enable the migration of **RegexReplace**statements. **true**: enables the migration of **RegexReplace**. **false**: disables the migration of **RegexReplace**.	• true • false	true	MigSupportForRegexReplace=false

Parameter	Description	Value Range	Default Value	Example
• commentPragmaAutomationTrans	Whether to comment out the AutomationTrans parameters in a table or index. **true**: Comment out the AutomationTrans. **false**: Retain the AutomationTrans parameters as they are.	• true • false	true	commentPragmaAutomationTrans=true
• supportJoinOperator	Indicates whether the left/right outer join operator (+) is supported. The following options are supported: • true • false	• true • false	false	supportJoinOperator=false
• migInsertWithTableAlias	The following options are supported: • true • false	• true • false	true	migInsertWithTableAlias=true
• varraySize	Varray datatype size	• NA	1024	varraySize=1024
• varrayObjectSize	VarrayObject datatype size	• NA	10240	varrayObjectSize= 10240
• migrationScope	Whether to package split or migrate completely.	• pkgSplit • completeMigration	completeMigration	migrationScope=completeMigration

Parameter	Description	Value Range	Default Value	Example
• migSupportConnectBy	The following options are supported: • true • false **true**: enables the migration of **connectBy**.	• true • false	true	migSupportConnectBy = true
• migrate_ConnectBy_Unnest	Whether to migrate **connectBy** and **Unrest**. **true**: enables the migration of **connectBy** and **Unnest**. **false**: retains the original value.	• true • false	true	migrate_ConnectBy_Unnest=true
• extendedGroupByClause	Whether to migrate the following extended functions of **GROUP BY**: • grouping sets • cube • rollup	• true • false	false	extendedGroupByClause=false
• supportDupValOnIndex	Whether to migrate **DUP_VAL_ON_INDEX**.	• UNIQUE_VIOLATION(V1R8) • OTHERS(older versions)	UNIQUE_VIOLATION	supportDupValOnIndex=UNIQUE_VIOLATION
• pkgvariable	Whether to migrate **pkgvariable**.	• localtable • sys_set_context • none	localtable	pkgvariable = localtable

Parameter	Description	Value Range	Default Value	Example
• addPackageNameList	The following options are supported: • true • false Set **package_name_list** to *<schema_name>* and invoke this schema.	• true • false	false	addPackageNameList = false
• addPackageTag	The following options are supported: • true • false If this parameter is set to **true**, **PACKAGE** is added in front of **AS|IS** in the stored procedure/function declaration.	• true • false	false	addPackageTag = true
• addGrantLine	The following options are supported: • true • false **true**: adds **GRANT** rows to each stored procedure/function at the end of the file.	• true • false	false	addGrantLine = true

Parameter	Description	Value Range	Default Value	Example
• MigDbmsLob	The following options are supported: • true • false • **TRUE** - DBMS_LOB is migrated. • **FALSE** - DBMS_LOB is not migrated	• true • false	false	MigDbmsLob=false
• uniqueConsForPartitonedTable	Unique or primary key constraint for partitioned table.	• comment_partition • comment_unique • none	comment_partition	uniqueConsForPartitonedTable = comment_partition
• MigSupportForRegexFunc	Possible values for **MigSupportForRegexReplace**.	• true • false	false	MigSupportForRegexFunc=false
• migSupportUnnest	Possible values for **migSupportUnnest**.	• true • false	true	migSupportUnnest = true
• MDSYS.MBRCOORDLIST	Replace the unsupported datatype **MDSYS.MBRCOORDLIST** with a user-defined datatype.	• None • Can enter any free text	None	MDSYS.MBRCOORDLIST=None
• MDSYS.SDO_GEOMETRY	Replace the unsupported datatype **MDSYS.SDO_GEOMETRY** with a user-defined datatype.	• None • Can enter any free text	None	MDSYS.SDO_GEOMETRY=None

Parameter	Description	Value Range	Default Value	Example
• GEOMETRY	Replace the unsupported datatype **GEOMETRY** with a user-defined datatype.	• None • Can enter any free text	None Input should not migrate.	GEOMETRY=None
• tdMigrateAddMonth	Possible values for **addMonth**.	• true • false	None	tdMigrateAddMonth=false IF TRUE THEN mig_ORA_ext.ADD_MONTHS (APPENDING mig_ORA_ext) OTHERWISE NOT APPEND. tdMigrateAddMonth=false

NOTE

DSC provides parameters for deleting partitions and subpartitions because the keywords for these features are not supported currently. You can comment out the statements containing these parameters or retain them as they are during script migration.

6.3.3.6 Netezza Configuration

Netezza parameters are used to customize rules for Netezza script migration.

Open the **features-netezza.properties** file in the **config** folder and set parameters in **Table 6-15** as required.

Table 6-15 Parameters in the **features-netezza.properties** file

Parameter	Description	Value Range	Default Value	Example
• rowstoreToColumnstore	Whether to convert row-store tables to column-store tables.	• true • false	false	rowstoreToColumnstore=false
• cstore_blob	The options are as follows: • bytea • none	• bytea • none	bytea	cstore_blob=bytea

Parameter	Description	Value Range	Default Value	Example
• keywords_addressed_using_as	The options are as follows: • OWNER • ATTRIBUTE • SOURCE • FREEZE	• OWNER • ATTRIBUTE • SOURCE • FREEZE	OWNER	keywords_addressed_using_as=OWNER
• keywords_addressed_using_doublequote	Possible values of **addressed_using_doublequote**	FREEZE	FREEZE	keywords_addressed_using_doublequote=FREEZE

6.3.4 Migration Process

6.3.4.1 Prerequisites

Executing Custom DB Scripts

Custom scripts are executed to support input keywords that do not exist in certain versions of the target database. These scripts must be executed in each target database before the migration.

Table 6-16 describes the custom scripts in the **DSC/scripts/** directory. For details about how to execute custom scripts, see **Custom DB Script Configuration**.

Table 6-16 Custom DB scripts

Script	Description
mig_fn_get_datatype_short_name.sql	Custom DB script for Teradata functions
mig_fn_castasint.sql	Custom DB script for migration of CAST AS INTEGER
vw_td_dbc_tables.sql	Custom DB script for migration of DBC.TABLES
vw_td_dbc_indices.sql	Custom DB script for migration of DBC.INDICES

Follow the steps to execute custom DB scripts:

Step 1 Use any of the following methods to execute the required scripts in all target databases for which migration is to be performed:

- Use **gsql**.
  - Use **gsql** to connect to the target database and paste all content in the SQL file to **gsql**, which will automatically execute the pasted contents.

    Run the following command to connect to the database:

    ```
    gsql -h <host_addr_xxx.xxx.xxx.xxx> -d <database_name> -U <user_name> -W <password> -p <port_number> -r
    ```

  - Use **gsql** to connect to the target database and execute a SQL file.

    Run the following command to connect to the database and run the SQL file:

    ```
    gsql -h <host_addr_xxx.xxx.xxx.xxx> -d <database_name> -U <user_name>  -W <password>  -p <port_number> -f <filename.sql> -o <output_filename> -L <log_filename.log>  -r
    ```

- Use Data Studio.

  Use Data Studio to connect to the target database, and then open and run the SQL file in Data Studio.

----End

Custom DB Script Configuration

Custom scripts are SQL files used to migrate from Teradata/Oracle the input keywords that do not exist in the target database.

These scripts must be executed in each target database before the migration.

Open the **scripts** folder in the release package. **Table 6-17** lists the folders and files in the **scripts** folder.

The SQL files contain the scripts for the custom migration functions. The GaussDB(DWS) database needs to use these functions to support specific features of Teradata.

Table 6-17 Custom DB scripts for DSC

Folder	Script File	Description
-- scripts	-	Folder: all scripts
------ teradata	-	Folder: Teradata functions and scripts
-------- view	-	Folder: scripts to configure views
-	vw_td_dbc_tables.sql	Script: used to enable migration of Teradata DBC.TABLES
-	vw_td_dbc_indices.sql	Script: used to enable migration of Teradata DBC.INDICES
-------- function	-	Folder: scripts to configure Teradata system functions

Folder	Script File	Description
-X	mig_fn_get_datatype_short_name.sql	Script: used to enable migration of Teradata DBC.COLUMNS
-	mig_fn_castasint.sql	Script: used to enable migration of CAST AS INTEGER
--------db_scripts	-	Folder: scripts to enable Teradata custom functions
-	mig_fn_get_datatype_short_name.sql	Script: used to enable migration of Teradata DBC.COLUMNS
--------core	-	Folder: Teradata core scripts
-	teradatacore.pm	Script: used to perform Perl migration

Configuring DSC and Migration Properties

To configure DSC, configure parameters in the configuration files in the **config** folder of DSC. **Table 6-18** describes the parameters.

Table 6-18 Parameters for configuring DSC

Migration	Configuration File	Parameter
Teradata SQL Migration	• DSC: *application.properties* • **Teradata SQL Configuration**: *features-teradata.properties*	deleteToTruncate=True/**False** distributeByHash=one/**many** extendedGroupByClause=True/**False** inToExists=True/**False** rowstoreToColumnstore=True/**False** session_mode=**Teradata**/ANSI tdMigrateDollar=**True**/False tdMigrateALIAS=True/**False** tdMigrateNULLIFZero=**True**/False tdMigrateZEROIFNULL=True/False volatile=**local temporary**/unlogged
Teradata Perl Migration	• DSC: *application.properties* • **Teradata Perl Configuration**: *perl-migration.properties*	add-timing-on=True/**False** db-bteq-tag-name=bteq db-tdsql-tag-name=sql_lang logging-level=error/warning/**info** migrate-variables=True/**False** remove-intermediate-files=True/**False** target_files=**overwrite**/cancel migrate-executequery=**True**/False

Migration	Configuration File	Parameter
MySQL SQL Migration	• DSC: *application.properties* • MySQL Configuration: *features-mysql.properties*	table.databaseAsSchema=true table.defaultSchema=public table.schema= table.orientation=ROW table.type=HASH table.partition-key.choose.strategy=partitionKeyChooserStrategy table.partition-key.name= table.compress.mode=NOCOMPRESS table.compress.level=0 table.compress.row=NO table.compress.column=LOW table.database.template=template0 table.index.rename=false table.database.onlyFullGroupBy=true table.database.realAsFloat=false

6.3.4.2 Preparations

Before the migration, create an input folder and an output folder, and copy all the SQL scripts to be migrated to the input folder. The following procedure describes how to prepare for the migration in Linux.

Step 1 Run the following commands to create an input folder and an output folder. You can create the folder anywhere based on your preferences. You can also use the default folders for input, output, provided as part of package.

```
mkdir input
mkdir output
```

DANGER

The tool reads the input folder in batches randomly. After the migration starts, it is recommended the users should not perform any modification on the input folder and files. Abnormal operations will affect the output result of the tool.

Step 2 Copy all SQL scripts to be migrated to the input folder.

NOTE

- If the encoding format of source files is not UTF-8, perform the following steps:
  1. Open the **application.properties** file in the **config** folder.
  2. Change the value of **encodingFormat** in the **application.properties** file to the required encoding format.

  DSC supports the UTF-8, ASCII, and GB2312 encoding formats. The values of **encodingFormat** are case-insensitive.
- To obtain the encoding format of a source file in Linux, run the following command on the server where the source file is located:
  file -bi *<Input file name>*

----End

6.3.4.3 Executing DSC

Important Notes

- Before starting DSC, specify the output folder path. Separate the input folder path, output folder path, and log path with spaces. The input folder path cannot contain spaces which will cause an error when DSC is used for migrating data. For details, see **Troubleshooting**.
- If the output folder contains subfolders or files, DSC deletes the subfolders and files or overwrites them based on parameter settings in the **application.properties** configuration file in the **config** folder before the migration. Deleted or overwritten subfolders and files cannot be restored by DSC.
- If migration tasks are performed concurrently on the same server (by the same or different DSCs), different migration tasks must use different output folder paths and log paths.
- You can specify a log path by configuring optional parameters. If the path is not specified, DSC automatically creates a log folder under **TOOL_HOME**. For details, see **Log Reference**.
- The maximum size of a single SQL statement is 20 KB. If the size exceeds 20 KB, the execution may be slow and the conversion may fail.

Migration Methods

You can run the **runDSC.sh** or **runDSC.bat** command to perform a migration task on Windows and Linux. For details, see **Table 6-19**.

Table 6-19 Migration on Windows and Linux

Migration	CLI Parameter
Teradata SQL Migration	**> ./runDSC.sh** **--source-db** *Teradata* [**--application-lang** SQL] [**--input-folder** *<input-script-path>*] [**--output-folder** *<output-script-path>*] [**--log-folder** *<log-path>*] [**--target-db/-T**][Optional] **> runDSC.bat** **--source-db** *Teradata* [**--application-lang** SQL] [**--input-folder** *<input-script-path>*] [**--output-folder** *<output-script-path>*] [**--log-folder** *<log-path>*] [**--target-db/-T**][Optional]

Migration	CLI Parameter
Teradata Perl Migration	> **./runDSC.sh** **--source-db** Teradata [**--application-lang** Perl] [**--input-folder** <input-script-path>] [**--output-folder** <output-script-path>] [**--log-folder** <log-path>] [**--target-db/-T**][Optional] > **runDSC.bat** **--source-db** *Teradata* [**--application-lang** Perl] [**--input-folder** *<input-script-path>*] [**--output-folder** *<output-script-path>*] [**--log-folder** *<log-path>*] [**--target-db/-T**][Optional]
MySQL SQL Migration	> **./runDSC.sh** **--source-db** MySql **[--application-lang** SQL] **[--input-folder** <input-script-path>] **[--output-folder** <output-script-path>] [**--log-folder** <log-path>] [--conversion-type <conversion-Type-BulkOrBlogic>] [**--target-db/-T**] > **runDSC.bat** **--source-db** MySql **[--application-lang** SQL] **[--input-folder** <input-script-path>] **[--output-folder** <output-script-path>] [**--log-folder** <log-path>] [--conversion-type <conversion-Type-BulkOrBlogic>] [**--target-db/-T**]

NOTE

- The CLI parameters are described as follows:
  - **source-db** specifies the source database. The value can be **Teradata**, which is case-insensitive.
  - **conversion-type** specifies the migration type. This parameter is optional. DSC supports the following migration types:

    Bulk: migrates DML and DDL scripts.

    BLogic: migrates service logic, such as stored procedures and functions.
  - **target-db** specifies the target database. The parameter value is **GaussDB(DWS)**.
- Command output description:

  Migration process start time indicates the migration start time and **Migration process end time** indicates the migration end time. **Total process time** indicates the total migration duration, in milliseconds. In addition, the total number of migrated files, total number of processors, number of used processors, log file path, and error log file path are also displayed on the console.

Examples

- Example 1: Run the following command to migrate the SQL file of the Teradata database to the SQL script of GaussDB(DWS) on Linux:
  ./runDSC.sh --source-db *Teradata* **--input-folder** D:\test\conversion\input **--output-folder** *D:|test|conversion|output* **--log-folder** *D:|test|conversion|log* **--conversion-type** *ddl* **--targetdb** gaussdb
- Example 2: Run the following command to migrate the SQL file of the Oracle database to the SQL script of GaussDB(DWS) on Windows:


```
runDSC.bat --source-db Teradata --input-folder D:\test\conversion\input --output-folder D:|test
|conversion|output --log-folder D:|test|conversion|log --conversion-type ddl --targetdb gaussdb
```

Migration details are displayed on the console (including the progress and completion status):

```
********** Schema Conversion Started **********
DSC process start time : Mon Jan 20 17:24:49 IST 2020
Statement count progress 100% completed [FILE(1/1)]
Schema Conversion Progress 100% completed
**************************************************************************
Total number of files in input folder : 1
Total number of valid files in input folder : 1
**************************************************************************
Log file path :...../DSC/DSC/log/dsc.log
Error Log file :
DSC process end time : Mon Jan 20 17:24:49 IST 2020
DSC total process time : 0 seconds
********** Schema Conversion Completed **********
```

6.3.4.4 Viewing Output Files and Logs

Viewing and Verifying Output Files

After the migration is complete, you can use a comparison tool (for example, BeyondCompare®) to compare the output file with its input file. Input SQL files can also be formatted for easier comparison.

1. Run the following command in Linux and view output files in the output folder. Operations in Windows are not described here.

   ```
   cd OUTPUT
   ls
   ```

   Information similar to the following is displayed:

   ```
   formattedSource    output
   user1@node79:~/Documentation/DSC/OUTPUT> cd output
   user1@node79:~/Documentation/DSC/OUTPUT/output> ls
   in_index.sql    input.sql    Input_table.sql    in_view.sql    MetadataInput.sql
   user1@node79:~/Documentation/DSC/OUTPUT/output>
   ```

2. Use the comparison tool to compare the output file with its input file. Check whether the keywords in the migrated SQL file meet the requirements of the target database. If not, contact technical support.

Viewing Log Files

Execution information and error messages are written into corresponding log files. For details, see **Log Reference**.

Check whether errors are logged. If they are, rectify the faults by following the instructions in **Troubleshooting**.

6.3.4.5 Troubleshooting

Migration related issues can be classified into:

- Tool execution issues: No output or incorrect output is displayed because DSC partially or fully failed to be executed. For more information about known issues and their solutions, see **Troubleshooting**.
- Migration syntax issues: DSC did not correctly recognize or migrate the migration syntax. For details, see **Constraints and Limitations**.

6.3.5 Database Schema Conversion

Functionality

runDSC.sh and runDSC.bat are used to migrate the schemas and queries of Teradata and MySQL to GaussDB(DWS), respectively.

6.3.5.1 Migration Parameters

Parameter Description

Table 6-20 Parameters

Long	Short	Data Type	Description	Value Range	Default Value	Example
--source-db	-S	String	Source database	• Teradata • MySQL	N/A	--source-db Teradata(or) -S Teradata
--input-folder	-I	String	Input folder containing Teradata or Oracle scripts	N/A	N/A	--input-folder */home/testmigration/Documentation/input* (or) -I */home/testmigration/Documentation/input*
--output-folder	-O	String	Output folder where the migrated scripts are saved	N/A	N/A	--output-folder */home/testmigration/Documentation/output*(or)-O */home/testmigration/Documentation/output*

Long	Short	Data Type	Description	Value Range	Default Value	Example
--application-lang	-A	String	Application language parser used for migration **SQL**: Migrate SQL schemas or scripts in SQL files. **Perl**: Migrate **BTEQ** or **SQL_LANG** scripts in Perl files.	• SQL • Perl	SQL	--application-lang Perl or -A *Perl*
--conversion-type	-M	String	Migration type. Set this parameter based on input scripts. **Bulk**: Migrate DML and DDL scripts. **BLogic**: Migrate service logic, such as procedures and functions. **BLogic** is used only for Oracle PL/SQL.	• Bulk • BLogic	Bulk	--conversion-type *ddl* or -M *ddl*
--log-folder	-L	String	Log file Path	N/A	N/A	--log-folder */home/testmigration/Documentation*(or)-L */home/testmigration/Documentation*

Long	Short	Data Type	Description	Value Range	Default Value	Example
--version-number	-VN	String	Oracle specified parameter	Oracle	N/A	--version-number or -V1R8_330
--target-db	-T	String	Target database	• gaussdbA	gaussdbA	--target-db gaussdbA (or) *-T gaussdbA*

Usage Guideline

It is mandatory to specify the source database, input folder path, and output folder path, and optional to specify the migration type and log path.

NOTE

If a user does not specify the logging path, then the tool creates the **log** folder in the TOOL_HOME path and saves all the logs in this *log* folder.

Examples

./runDSC.sh --source-db Teradata **--input-folder** opt/DSC/DSC/input/oracle/ --output-folder /opt/DSC/DSC/output/ --log-folder /opt/DSC/DSC/log/ **--application-lang** SQL **--conversion-type** ddl **--targetdb** gaussdbA

System Response

```
********** Schema Conversion Started **********
DSC process start time : Mon Jan 20 17:24:49 IST 2020
Statement count progress 100% completed [FILE(1/1)]

Schema Conversion Progress 100% completed
********************************************************************************
Total number of files in input folder : 1
********************************************************************************
Log file path :...../DSC/DSC/log/dsc.log
DSC process end time : Mon Jan 20 17:24:49 IST 2020
DSC total process time : 0 seconds
********** Schema Conversion Completed **********
```

NOTE

If there is no sql file present in the input folder, then the following message is displayed in console:

```
DSC process start time : Tue Jan 21 16:04:28 IST 2020
No valid files found in the input folder. Hence DSC stopped.
DSC Application failed to start : No valid files found in the input folder.
```


Environment Creation and Restoration Procedure (database and database user)

Creating a GaussDB(DWS) database and schema

Step 1 Log in to postgres:

```
gsql -p <port> -d postgres
drop database <database name>;
create database <database name>;
\c <database name>
GRANT ALL PRIVILEGES ON DATABASE <database name> TO <user>;
grant database to <user>;\q
gsql -p <port> -d <database name> -U <user> -W <password> -h <IP> -f
drop database <database name>;
create database <database name>;
\c <database name>;
GRANT ALL PRIVILEGES ON DATABASE <database name> TO <user>;
gsql -p <port> -d <database name> -U <user> -W <password>  -f
```

Step 2 Run all files in setup.

----End

Commands:

```
sh runDSC.sh -S oracle -M blogic -I <input path>
sh runDSC.sh -I input/ -S oracle -M ddl -L log_temp -P input/bulk/1_table/
```

6.3.5.2 Teradata SQL Migration

DSC supports the migration from Teradata to GaussDB(DWS), including the migration of schemas, DML, queries, system functions, and type conversion.

Performing Teradata SQL Migration

Run the following commands to set the source database, input and output folder paths, log paths, and application language.

Linux:

```
./runDSC.sh
--source-db Teradata
[--input-folder <input-script-path>]
[--output-folder <output-script-path>]
[--log-folder <log-path>]
[--application-lang SQL]
```

Windows:

```
runDSC.bat
--source-db Teradata
[--input-folder <input-script-path>]
[--output-folder <output-script-path>]
[--log-folder <log-path>]
[--application-lang SQL]
```

For example:

Linux:

```
./runDSC.sh --source-db Teradata --target-db GaussDBA --input-folder /opt/DSC/DSC/input/teradata/ --output-folder /opt/DSC/DSC/output/ --log-folder /opt/DSC/DSC/log/ --application-lang SQL --conversion-type Bulk
```

Windows:


```
runDSC.bat --source-db Teradata --target-db GaussDBA --input-folder D:\test\conversion\input --output-folder D:\test\conversion\output --log-folder D:\test\conversion\log --application-lang SQL --conversion-type Bulk
```

During the execution of DSC, the migration summary, including the progress and completion status, is displayed on the console. Execution information and errors will be recorded in **SQL Migration Logs**.

```
********** Schema Conversion Started **********
DSC process start time : Mon Jan 20 17:24:49 IST 2020
Statement count progress 100% completed [FILE(1/1)]
Schema Conversion Progress 100% completed
**************************************************************************
Total number of files in input folder : 1
**************************************************************************
Log file path :...../DSC/DSC/log/dsc.log
DSC process end time : Mon Jan 20 17:24:49 IST 2020
DSC total process time : 0 seconds
********** Schema Conversion Completed **********
```

For details about how to migrate Teradata SQL using DSC, see **Executing DSC**.

NOTE

During the migration, the metadata of the input script can be called and is stored in the following files:

- Teradata migration

  1.teradata-set-table.properties

Clear the preceding files in the following scenarios:

- Migration of different files
- Migration of the same file with different parameter settings

6.3.5.3 Teradata Perl Migration

Overview

This section describes how to migrate Teradata Perl files.

Run the **runDSC.sh** or **runDSC.bat** command and set **--application-lang** to **perl** to migrate Teradata **BTEQ** or **SQL_LANG** scripts in Perl files to Perl-compatible GaussDB(DWS). After migrating Perl files, you can verify the migration by comparing the output file with its input file using a comparison tool.

The process of migrating Perl files is as follows:

1. Complete the steps in section **Prerequisites**.
2. Create an input folder and copy the Perl files to be migrated to the folder. For example, create a **/migrationfiles/perlfiles** folder.
3. Execute DSC to migrate Perl scripts and set **db-bteq-tag-name** to **BTEQ** or **db-tdsql-tag-name** to **SQL_LANG**.
   a. The DSC extracts the **BTEQ** or **SQL_LANG** scripts from the Perl files.
      i. BTEQ is the tag name, which contains a set of BTEQ scripts. This tag name is configurable using the **db-bteq-tag-name** configuration parameter in **perl-migration.properties** file.
      ii. SQL_LANG is another tag name, which contains Teradata SQL statements. This is also configurable using the **db-tdsql-tag-name** configuration parameter.

b. DSC invokes the Teradata SQL to migrate the extracted SQL scripts. For details about Teradata SQL migration, see **Teradata SQL Migration**.

c. Perl files are embedded in the migrated scripts.

4. DSC creates the migrated files in the specified output folder. If no output folder is specified, DSC creates an output folder named **converted** in the input folder, for example, **/migrationfiles/perlfiles/converted**.

   NOTE

   - Perl variables containing SQL statements can also be migrated to SQL by setting the **migrate-variables** parameter.
   - For perl v 5.10.0 and later are compatible.

Teradata Perl Migration

To migrate Perl files, execute DSC with **--source-db Teradata** and **--application-lang Perl** parameter values. DSC supports the migration of **BTEQ** and **SQL_LANG** scripts. You can specify the scripts to be migrated by setting **db-bteq-tag-name** or **db-tdsql-tag-name**.

Run the following commands to set the source database, input and output folder paths, log paths, and application language.

Linux:

```
./runDSC.sh
--source-db|-S Teradata
[--application-lang|-A Perl]
[--input-folder|-I <input-script-path>]
[--output-folder|-O <output-script-path>]
[--conversion-type|-M <Bulk or BLogic>]
[--log-folder|-L <log-path>]
```

Windows:

```
runDSC.bat
--source-db|-S Teradata
[--application-lang|-A Perl]
[--input-folder|-I <input-script-path>]
[--output-folder|-O <output-script-path>]
[--conversion-type|-M <Bulk or BLogic>]
[--log-folder|-L <log-path>]
```

For example:

```
./runDSC.sh --input-folder /opt/DSC/DSC/input/teradata_perl/ --output-folder /opt/DSC/DSC/output/ --source-db teradata --conversion-type Bulk --application-lang PERL
```

During the execution of DSC, the migration summary, including the progress and completion status, is displayed on the console.

```
********** Schema Conversion Started **********
DSC process start time : Mon Jan 20 17:24:49 IST 2020
Statement count progress 100% completed [FILE(1/1)]
Schema Conversion Progress 100% completed
**************************************************************************
Total number of files in input folder : 1
**************************************************************************
Log file path :...../DSC/DSC/log/dsc.log
DSC process end time : Mon Jan 20 17:24:49 IST 2020
DSC total process time : 0 seconds
 ********** Schema Conversion Completed **********
```

For details about the parameters for Teradata Perl migration, see **Teradata Perl Configuration**.

For details about CLI parameters, see **Database Schema Conversion**.

NOTE

- DSC formats the input files and saves them in the output folder. You can compare the formatted input files with the output files.
- Ensure that there are no spaces in the input path. If there is a space, DSC throws an error. For details, see **Troubleshooting**.
- For details about logs, see **Log Reference**.
- If the output folder contains subfolders or files, DSC deletes the subfolders and files or overwrites them based on parameter settings in the **application.properties** configuration file in the **config** folder before the migration. Deleted or overwritten subfolders and files cannot be restored by DSC.
- **Process start time** indicates the migration start time and **Process end time** indicates the migration end time. **Process total time** indicates the total migration duration, in milliseconds. In addition, the total number of migrated files, total number of processors, number of used processors, log file path, and error log file path are also displayed on the console.
- Set **--add-timing-on** to **true** in the **perl-migration.properties** file to add a custom script to calculate statement execution time.

  Example:

  Input

```
$V_SQL2 = "SELECT T1.userTypeInd FROM T07_EBM_CAMP T1  WHERE T1.Camp_List_Id = '$abc'";
$STH = $dbh->prepare($V_SQL2);
$sth->execute();
@rows = $sth->fetchrow();
```

  Output

```
$V_SQL2 = "SELECT T1.userTypeInd FROM T07_EBM_CAMP T1  WHERE T1.Camp_List_Id = '$abc'";
$STH = $dbh->prepare($V_SQL2);
use Time::HiRes qw/gettimeofday/;
my $start = [Time::HiRes::gettimeofday()];
$sth->execute();
my $elapsed = Time::HiRes::tv_interval($start);
$elapsed = $elapsed * 1000;
printf("Time: %.3f ms\n", $elapsed);
@rows = $sth->fetchrow();
```

- GROUP and OTHERS must not have write permission for the files or folders specified by**--input-folder**. That is, the privilege for the folder specified by **--input-folder** must not be higher than **755**. For security purposes, DSC will not be executed if the input files or folders have the write permission.
- If migration tasks are executed concurrently, the input folder must be unique for each task.

Best Practices

To optimize the migration, you are advised to follow the standard practices:

- **BTEQ** scripts must be in the following format:

```
print BTEQ <<ENDOFINPUT;
TRUNCATE TABLE employee;
ENDOFINPUT
close(BTEQ);
```

- **SQL_LANG** scripts must be in the following format:

```
my $sSQL=<<SQL_LANG;
TRUNCATE TABLE employee;
SQL_LANG
```

- Comment must not contain the following information:
  - print BTEQ <<ENDOFINPUT

- ENDOFINPUT
- close(BTEQ)
- my $sSQL=<<SQL_LANG
- SQL_LANG

6.3.5.4 MySQL SQL Migration

DSC supports the migration from MySQL to GaussDB(DWS), including the migration of schemas, DML, queries, system functions, and PL/SQL.

Performing MySQL Migration on Linux

Run the following command on Linux to start the migration. You need to specify the source database, input and output folder paths, and log paths. The application language is SQL.

```
./runDSC.sh
--source-db MySQL
[--input-folder <input-script-path>]
[--output-folder <output-script-path>]
[--application-lang SQL]
[--conversion-type <conversion-type>]
[--log-folder <log-path>]
```

During the execution of DSC, the migration summary, including the progress and completion status, is displayed on the console.

```
./runDSC.sh --source-db MySQL --input-folder /opt/DSC/DSC/input/mysql/ --output-folder /opt/DSC/DSC/
output/ --application-lang SQL --conversion-type BULK --log-folder/opt/DSC/DSC/log/
********** Schema Conversion Started **********
DSC process start time : Mon Jan 20 17:24:49 IST 2020
Statement count progress 100% completed [FILE(1/1)]
Schema Conversion Progress 100% completed
**************************************************************************
Total number of files in input folder : 1
Total number of valid files in input folder : 1
**************************************************************************
Log file path :...../DSC/DSC/log/dsc.log
Error Log file :
DSC process end time : Mon Jan 20 17:24:49 IST 2020
DSC total process time : 0 seconds
********** Schema Conversion Completed **********
```

Performing MySQL Migration on Windows

Run the following command on Windows to start the migration. You need to specify the source database, input and output folder paths, and log paths. The application language can be SQL or Perl. The default language is SQL. The migration type can be **Bulk** or **BLogic**.

```
runDSC.bat
--source-db MySQL
[--input-folder <input-script-path>]
[--output-folder <output-script-path>]
[--application-lang SQL]
[--conversion-type <conversion-type>]
[--log-folder <log-path>]
```

During the execution of DSC, the migration summary, including the progress and completion status, is displayed on the console.


```
runDSC.bat --source-db MySQL --target-db GaussDBA --input-folder D:\test\conversion\input --output-folder D:\test\conversion\output --log-folder D:\test\conversion\log --application-lang SQL --conversion-type  BULK
********** Schema Conversion Started **********
DSC process start time : Mon Jan 20 17:24:49 IST 2020
Statement count progress 100% completed [FILE(1/1)]
Schema Conversion Progress 100% completed
**********
Total number of files in input folder : 1
**********
Log file path :...../DSC/DSC/log/dsc.log
DSC process end time : Mon Jan 20 17:24:49 IST 2020
DSC total process time : 0 seconds
********** Schema Conversion Completed **********
```

6.3.5.5 Oracle SQL Migration

DSC supports the migration from Oracle to GaussDB(DWS), including the migration of schemas, DML, queries, system functions, and PL/SQL.

Performing Oracle SQL Migration

Run the following commands to set the source database, input and output folder paths, log paths, application language, and conversion type:

Linux:

```
./runDSC.sh
--source-db Oracle
[--input-folder <input-script-path>]
[--output-folder <output-script-path>]
[--log-folder <log-path>]
[--application-lang Oracle]
[--conversion-type <conversion-type>]
```

Windows:

```
runDSC.bat
--source-db Oracle
[--input-folder <input-script-path>]
[--output-folder <output-script-path>]
[--log-folder <log-path>]
[--application-lang Oracle]
[--conversion-type <conversion-type>]
```

- When migrating common DDL statements (such as tables, views, indexes, sequences, and so on) that do not contain PL/SQL statements, use the Bulk mode (that is, set **conversion-type** to **Bulk**).

  Run the following commands to set **conversion-type** to **Bulk**. The commands include the folder paths in the example.

  Linux:

  ```
  ./runDSC.sh --source-db Oracle --input-folder /opt/DSC/DSC/input/oracle/ --output-folder /opt/DSC/DSC/output/ --log-folder /opt/DSC/DSC/log/ --application-lang SQL --conversion-type bulk --target-db gaussdbA
  ```

  Windows:

  ```
  runDSC.bat --source-db Oracle --input-folder D:\test\conversion\input --output-folder D:\test\conversion\output --log-folder D:\test\conversion\log --application-lang SQL --conversion-type bulk --target-db gaussdbA
  ```

  During the execution of DSC, the migration summary, including the progress and completion status, is displayed on the console. Execution information and errors will be recorded in **SQL Migration Logs**.


```
********************* Schema Conversion Started ************************
DSC process start time : Mon Jan 20 17:24:49 IST 2020
Statement count progress 100% completed [FILE(1/1)]

Schema Conversion Progress 100% completed
********************************************************************************
Total number of files in input folder : 1
********************************************************************************
Log file path :...../DSC/DSC/log/dsc.log
DSC process end time : Mon Jan 20 17:24:49 IST 2020
DSC total process time : 0 seconds
******************** Schema Conversion Completed ************************
```

- When migrating objects such as functions, procedures, and packages that contain PL/SQL statements, use the BLogic mode. That is, set **conversion-type** to **BLogic**.

  Run the following command to set **conversion-type** to **BLogic**. The command includes the folder paths in the example.

  ```
  runDSC.bat --source-db Oracle --input-folder D:\test\conversion\input --output-folder D:\test
  \conversion\output --log-folder D:\test\conversion\log --application-lang SQL --conversion-type
  blogic --target-db gaussdbA
  ```

  During the execution of DSC, the migration summary, including the progress and completion status, is displayed on the console. Execution information and errors will be recorded in **SQL Migration Logs**.

  ```
  ********************* Schema Conversion Started ************************
  DSC process start time : Mon Jan 20 17:24:49 IST 2020
  Statement count progress 100% completed [FILE(1/1)]

  Schema Conversion Progress 100% completed
  ********************************************************************************
  Total number of files in input folder : 1
  Total number of valid files in input folder : 1
  ********************************************************************************
  Log file path :...../DSC/DSC/log/dsc.log
  Error Log file :
  DSC process end time : Mon Jan 20 17:24:49 IST 2020
  DSC total process time : 0 seconds
  ******************** Schema Conversion Completed ************************
  ```

  Common DDL scripts and PL/SQL scripts should be stored in different input folders for migration.

Oracle PACKAGE Migration Precautions

1. The package specifications (that is, the package header) and the package body should be stored in different files and in the same input path for migration.

2. You need to migrate common DDL statements in Bulk mode, including all table structure information referenced in the **PACKAGE** script, to form a dictionary in the **config/create-types-UDT.properties** file. Then, migrate the package specifications (that is, the package header) and the package body in Blogic mode. The details are as follows:

When some Oracle **PACKAGE** packages define package specifications, the **tbName.colName%TYPE** syntax is used to declare custom record types based on other table objects.

```
For example:
   CREATE OR REPLACE PACKAGE p_emp
   AS
      --Define the RECORD type
      TYPE re_emp IS RECORD(
         rno emp.empno%TYPE,
```



```
        rname emp.empname%TYPE
    );

END;
```

The column data type cannot be specified using the **tbName.colName%TYPE** syntax in a **CREATE TYPE** statement on GaussDB(DWS). Therefore, DSC needs to build a database context environment containing the **EMP** table information during migration. In this case, you need to use DSC to migrate all table creation scripts, that is, to migrate common DDL statements in Bulk mode. DSC automatically generates corresponding data dictionaries. After the context environment containing various table information is built, you can migrate the Oracle PACKAGE in Blogic mode. In this case, the **re_emp** record type is migrated based on the column type of the **EMP** table.

```
Expected output
  CREATE TYPE p_emp.re_emp AS (
    rno NUMBER(4),
    rname VARCHAR2(10)
  );
```

For details about how to migrate Oracle SQL using DSC, see **Executing DSC**.

6.3.5.6 Netezza SQL Migration

DSC supports the migration from Netezza to GaussDB(DWS), including the migration of schemas, DML, queries, system functions, and PL/SQL.

Run the following commands to set the source database, input and output folder paths, log paths, application language, and conversion type:

Linux:

```
./runDSC.sh
--source-db Netezza
[--input-folder <input-script-path>]
[--output-folder <output-script-path>]
[--log-folder <log-path>]
[--application-lang SQL]
[--conversion-type <conversion-type>]
```

Windows:

```
runDSC.bat
--source-db Netezza
[--input-folder <input-script-path>]
[--output-folder <output-script-path>]
[--log-folder <log-path>]
[--application-lang SQL]
[--conversion-type <conversion-type>]
```

For example:

Linux:

```
./runDSC.sh --source-db Netezza --input-folder /opt/DSC/DSC/input/mysql/ --output-folder /opt/DSC/DSC/output/ --application-lang SQL --conversion-type BULK --log-folder/opt/DSC/DSC/log/
```

Windows:

```
runDSC.bat --source-db Netezza--target-db GaussDBA --input-folder D:\test\conversion\input --output-folder D:\test\conversion\output --log-folder D:\test\conversion\log --application-lang SQL --conversion-type Bulk
```


During the execution of DSC, the migration summary, including the progress and completion status, is displayed on the console. Execution information and errors will be recorded in **SQL Migration Logs**.

```
********** Schema Conversion Started **********
DSC process start time : Mon Jan 20 17:24:49 IST 2020
Statement count progress 100% completed [FILE(1/1)]
Schema Conversion Progress 100% completed
**************************************************************************
Total number of files in input folder : 1
**************************************************************************
Log file path :...../DSC/DSC/log/dsc.log
DSC process end time : Mon Jan 20 17:24:49 IST 2020
DSC total process time : 0 seconds
********** Schema Conversion Completed **********
```

For details about how to migrate Netezza SQL using DSC, see **Executing DSC**.

6.3.5.7 Verification

Verification After Migration

After DSC converts the source sql files, execute the converted files on target GaussDB(DWS) and provide a report with details of number of statements succeeded and failed.

After the DSC finishes the translation, it will invoke (controlled through a configuration item) post migration verification script. The verification script (for details about the configuration, see the configuration file) is connected to the target GaussDB database and executed.

The post migration verification script will connect to the target gauss database (details are configured in a configuration file) and executes the scripts.

1. **application.properties** in config folder

   Execute migrated script on Gauss DB: true/false, default value = false

   executesqlingauss=true

   true: It will execute the migrated script on gaussdb

2. **gaussdb.properties** in config folder

   #Target Database configurations

   ```
   #gauss database user with all privileges
   gaussdb-user=
   gaussdb-port=
   #Database name for GaussDBA
   gaussdb-name=
   #gaussdb ip
   gaussdb-ip=
   ```

   Dependency on the gsql client:

   a. gsql (GaussDB(DWS)) is required for executing scripts on GaussDB(DWS). Therefore, to ensure the smooth running of DSC, DSC is required to run on a node installed with a GaussDB(DWS) instance or client (gsql), and the user that performs verification must have the permission for executing commands using gsql.

   b. Since the Gauss DB Instance/Client can be installed on a Linux OS only, this can be used to verify functionality only on a Linux environment.

c. To execute the gsql command on a remote GaussDB instance, it is advised to add the client system IP/hostname in the following configuration file of Gauss DB instance.

```
/home/gsmig/database/coordinator
---pg_hba.conf
```

Response

GaussDB(DWS)

```
********************* Verification Started ****************************
Sql script execution on Gauss DB start time : Wed Jan 22 17:27:07 CST 2020
Sql script execution on Gauss DB end time : Wed Jan 22 17:27:44 CST 2020

Summary of Verification :
================================================================================================================================================
Statement              | Total             | Passed            | Failed            | Success Rate(%)
------------------------------------------------------------------------------------------------------------------------------------------------
-----
COMMENT                | 15                | 15                | 0                 | 100
CREATE VIEW            | 4                 | 3                 | 1                 | 75
CREATE INDEX           | 4                 | 3                 | 1                 | 75
CREATE TABLE           | 6                 | 6                 | 0                 | 100
ALTER TABLE            | 3                 | 3                 | 0                 | 100
------------------------------------------------------------------------------------------------------------------------------------------------
---
Total                  | 32                | 30                | 2                 | 93

Gauss Execution Log file : /home/gsmig/18Jan/DSC/DSC/log/gaussexecutionlog.log
Gauss Execution Error Log file : /home/gsmig/18Jan/DSC/DSC/log/gaussexecutionerror.log
Verification finished in 38 seconds

********************* Verification Completed ****************************
```

6.3.6 Version Command Migration

Functionality

The **version** command is used to display the version number of the DSC.

Syntax

Linux:

```
./runDSC.sh --version
```

Windows:

```
runDSC.bat --version
```

How to Use

Linux:

```
./runDSC.sh --version
```

Windows:

```
runDSC.bat --version
```


System Response

```
Version: DSC (Gauss Tools v2.0.0)
```

6.3.7 Help Command Migration

Functionality

The **help** command is used to provide the help information for the commands supported by DSC.

Syntax

Linux:

```
./runDSC.sh --help
```

Windows:

```
runDSC.bat --help
```

Examples

Linux:

```
./runDSC.sh --help
```

Windows:

```
runDSC.bat --help
```

System Response

Linux:

```
./runDSC.sh --help
To migrate teradata/oracle/netezza/mysql/db2 database scripts to DWS
runDSC.sh -S <source-database> [-T <target-database>] -I <input-script-path> -O <output-script-path> [-M
<conversion-type>]  [-A <application-lang>]  [-L <log-path>]  [-VN <version-number>]


  -S | --source-db
  The source database, which can be either Teradata or Oracle or Netezza or MySql or DB2.


  -T | --target-db
  The target database, which can be either GaussDBT or GaussDBA.


  -I | --input-folder
  The input/source folder that contains the Teradata/Oracle/Netezza/MySql/DB2 scripts to be migrated.


  -O | --output-folder
  The output/target folder where the migrated scripts are placed.


  -M | --conversion-type
  The conversion type, which can be either Bulk or BLogic.


  -A | --application-lang
  The application language type, which can be either SQL or Perl.
```



```
   -L | --log-folder
   The log file path where the log files are created.


   -VN | --version-number
   The version number, which can be either V1R7 or V1R8_330.

  " -P | --pre-execution-path%n" + "
   Path containig the dependent Objects scripts which needs to be executed before the verification step.%n"
+ "%n" +



To display DSC version details
runDSC.sh -V | --version


To display DSC help details
runDSC.sh -H | --help


[Internal] To guess encoding for a file
runDSC.sh guessencoding -F <filename>


   -F | --file-name
   The filename for which the encoding must be guessed.


Refer the user manual for more details.
```

Windows:

```
runDSC.bat --help
To migrate teradata/oracle/netezza/mysql/db2 database scripts to DWS
runDSC.bat -S <source-database> [-T <target-database>] -I <input-script-path> -O <output-script-path> [-
M <conversion-type>]  [-A <application-lang>]  [-L <log-path>]  [-VN <version-number>]


   -S | --source-db
   The source database, which can be either Teradata or Oracle or Netezza or MySql or DB2.


   -T | --target-db
   The target database, which can be either GaussDBT or GaussDBA.


   -I | --input-folder
   The input/source folder that contains the Teradata/Oracle/Netezza/MySql/DB2 scripts to be migrated.


   -O | --output-folder
   The output/target folder where the migrated scripts are placed.


   -M | --conversion-type
   The conversion type, which can be either Bulk or BLogic.


   -A | --application-lang
   The application language type, which can be either SQL or Perl.


   -L | --log-folder
   The log file path where the log files are created.


   -VN | --version-number
   The version number, which can be either V1R7 or V1R8_330.
```



```
To display DSC version details
runDSC.sh -V | --version


To display DSC help details
runDSC.sh -H | --help


[Internal] To guess encoding for a file
runDSC.sh guessencoding -F <filename>


   -F | --file-name
   The filename for which the encoding must be guessed.


Refer the user manual for more details.
```

6.3.8 Log Reference

6.3.8.1 Overview

The log files are the repository for all operations and status of the DSC. The following log files are available:

- **SQL Migration Logs**
  a. *DSC.log*: SQL Migration all activities.
  b. *DSCError.log*: SQL Migration errors.
  c. *successRead.log*: SQL Migration successful input file reads.
  d. *successWrite.log*: SQL Migration successful output file writes.
- **Perl Migration Logs**
  a. *perlDSC.log*: Perl Migration all activities, warnings and errors.

Apache Log4j is used for the DSC logging framework. The following Log4j configuration files are used and can be customized as required:

- Teradata/Oracle/Netezza/DB2 : config/log4j2.xml
- MySQL : config/log4j2_mysql.xml

6.3.8.2 SQL Migration Logs

The SQL DSC (DSC.jar) supports the following types of logging:

- Activity logging
- Error logging
- successRead
- successWrite

NOTE

- If a user specifies a log path, then all the logs are saved in the specified log path.
- If a user does not specify the logging path, then the tool creates the **log** folder in the TOOL_HOME path and saves all the logs in this *log* folder.
- To control the disk space usage, the maximum size of a log file is 10 MB. You can have a maximum of 10 log files.
- The tool does not log sensitive data such as queries.

Activity Logging

DSC saves all log and error information to **DSC.log**. This file is available in the **log** folder. The **DSC.log** file consists of details such as user who executed the migration and files that have been migrated along with the timestamp. The logging level for activity logging is INFO.

The file structure of the **DSC.log** file is as follows:

```
2020-01-22 09:35:10,769 INFO CLMigrationUtility:159 DSC is initiated by xxxxx
2020-01-22 09:35:10,828 INFO CLMigrationUtility:456 Successfully changed permission of files in
D:\Migration\Gauss_Tools_18_Migration\code\migration\config
2020-01-22 09:35:10,832 INFO PropertyLoader:90 Successfully loaded Property file : D:\Migration
\Gauss_Tools_18_Migration\code\migration\config\application.properties
2020-01-22 09:35:10,833 INFO ApplicationPropertyLoader:42 Application properties have been loaded
Successfully
2020-01-22 09:35:10,917 INFO MigrationValidatorService:549 Files in output directory has been overwritten
as configured by xxxxx
2020-01-22 09:35:10,920 INFO PropertyLoader:90 Successfully loaded Property file : D:\Migration
\Gauss_Tools_18_Migration\code\migration\config\features-oracle.properties
2020-01-22 09:35:10,921 INFO FeatureLoader:41 Features have been loaded Successfully
2020-01-22 09:35:10,926 INFO MigrationService:80 DSC process start time : Wed Jan 22 09:35:10 GMT
+05:30 2020
2020-01-22 09:35:10,933 INFO FileHandler:179  File is not supported. D:\Migration_Output\Source
\ARRYTYPE.sql-
2020-01-22 09:35:10,934 INFO FileHandler:179  File is not supported. D:\Migration_Output\Source
\varray.sql-
2020-01-22 09:35:12,816 INFO PropertyLoader:90 Successfully loaded Property file : D:\Migration
\Gauss_Tools_18_Migration\code\migration\config\global-temp-tables.properties
2020-01-22 09:35:12,830 INFO PropertyLoader:90 Successfully loaded Property file : D:\Migration
\Gauss_Tools_18_Migration\code\migration\config\create-types-UDT.properties
2020-01-22 09:35:12,834 INFO PropertyLoader:90 Successfully loaded Property file : D:\Migration
\Gauss_Tools_18_Migration\code\migration\config\package-names-oracle.properties
2020-01-22 09:35:12,849 INFO DBMigrationService:76 Number of Available Processors: 4
2020-01-22 09:35:12,850 INFO DBMigrationService:78 Configured simultaneous processes in the Tool : 3
2020-01-22 09:35:13,032 INFO MigrationProcessor:94 File name: D:\Migration_Output\Source\Input.sql is
started
2020-01-22 09:35:13,270 INFO FileHandler:606 guessencoding command output = Error: Unable to access
jarfile D:\Migration\Gauss_Tools_18_Migration\code\migration\RE_migration\target\dsctool.jar , for file=
D:\Migration_Output\Source\Input.sql
2020-01-22 09:35:13,272 INFO FileHandler:625 couldn't get the encoding format, so using the default
charset for D:\Migration_Output\Source\Input.sql
2020-01-22 09:35:13,272 INFO FileHandler:310 File D:\Migration_Output\Source\Input.sql will be read with
charset : UTF-8
2020-01-22 09:35:13,390 INFO FileHandler:668 D:\Migration_Output\target\output\Input.sql - File already
exists/Failed to create target file
2020-01-22 09:35:13,562 INFO FileHandler:606 guessencoding command output = Error: Unable to access
jarfile D:\Migration\Gauss_Tools_18_Migration\code\migration\RE_migration\target\dsctool.jar , for file=
D:\Migration_Output\Source\Input.sql
2020-01-22 09:35:13,563 INFO FileHandler:625 couldn't get the encoding format, so using the default
charset for D:\Migration_Output\Source\Input.sql
2020-01-22 09:35:13,563 INFO FileHandler:675 File D:\Migration_Output\Source\Input.sql will be written
with charset : UTF-8
2020-01-22 09:35:13,604 INFO MigrationProcessor:139 File name: D:\Migration_Output\Source\Input.sql is
processed successfully
2020-01-22 09:35:13,605 INFO MigrationService:147 Total number of files in Input folder : 3
```



```
2020-01-22 09:35:13,605 INFO MigrationService:148 Total number of queries : 1
22020-01-22 09:35:13,607 INFO PropertyLoader:164 Successfully updated Property file : D:\Migration
\Gauss_Tools_18_Migration\code\migration\config\global-temp-tables.properties
2020-01-22 09:35:13,630 INFO PropertyLoader:164 Successfully updated Property file : D:\Migration
\Gauss_Tools_18_Migration\code\migration\config\create-types-UDT.properties
2020-01-22 09:35:13,631 INFO PropertyLoader:164 Successfully updated Property file : D:\Migration
\Gauss_Tools_18_Migration\code\migration\config\package-names-oracle.properties
2020-01-22 09:35:13,632 INFO CLMigrationUtility:305 Log file : dsc.log and the file is present in the path :
D:\Migration_Output\log
2020-01-22 09:35:13,632 INFO CLMigrationUtility:312 DSC process end time : Wed Jan 22 09:35:13 GMT
+05:30 2020
2020-01-22 09:35:13,632 INFO CLMigrationUtility:217 Total process time : 2842 seconds
```

Error Logging

DSC logs only the errors that are encountered during the migration process to **DSCError.log**. This file is available in the **log** folder. The **DSCError.log** file consists of details such as date and time of the error and the details of the file (file name) along with the query position. The logging level for error logging is ERROR.

The file structure of the **DSCError.log** file is as follows:

```
2017-06-29 14:07:39,585 ERROR TeradataBulkHandler:172 Error occurred during processing of input in Bulk
Migration. PreQueryValidation failed in not proper termination or exclude keyword. /home/testmigration/
Documentation/Input/c005.sql for Query in position : 4
2017-06-29 14:07:39,962 ERROR TeradataBulkHandler:172 Error occurred during processing of input in Bulk
Migration. PreQueryValidation failed in not proper termination or exclude keyword. /home/testmigration/
Documentation/Input/c013.sql for Query in position : 11
2017-06-29 14:07:40,136 ERROR QueryConversionUtility:250 Query is not converted as it contains
unsupported keyword: join select
2017-06-29 14:07:40,136 ERROR TeradataBulkHandler:172 Error occurred during processing of input in Bulk
Migration. PreQueryValidation failed in not proper termination or exclude keyword. /home/testmigration/
Documentation/Input/sample.sql for Query in position : 1
2017-06-29 14:07:40,136 ERROR TeradataBulkHandler:172 Error occurred during processing of input in Bulk
Migration. PreQueryValidation failed in not proper termination or exclude keyword. /home/testmigration/
Documentation/Input/sample.sql for Query in position : 3
```

successRead

After a file has been read by the DSC, the file is logged for tracking purposes. In certain scenarios, these logs let the user know the status of the execution of files. This file is available in the **log** folder. The log file consists of details such as date and time, and the details of the file name. The logging level for this log file is INFO.

The file structure of the **successRead.log** file is as follows:

```
2017-07-21 14:13:00,461 INFO readlogger:213 /home/testmigration/Documentation/is not in.sql is read
successfully.
2017-07-21 14:13:00,957 INFO readlogger:213 /home/testmigration/Documentation/date quotes.sql is read
successfully.
2017-07-21 14:13:01,509 INFO readlogger:213 /home/testmigration/Documentation/column alias
replace.sql is read successfully.
2017-07-21 14:13:02,034 INFO readlogger:213 /home/testmigration/Documentation/sampleRownum.sql is
read successfully.
2017-07-21 14:13:02,578 INFO readlogger:213 /home/testmigration/Documentation/samp.sql is read
successfully.
2017-07-21 14:13:03,145 INFO readlogger:213 /home/testmigration/Documentation/2.6BuildInputs/
testWithNodataSamples.sql is read successfully.
```

successWrite

DSC reads a file, processes it, and writes the output to the disk. This is logged to the success write log file. In some scenarios, this log lets the user know which of

the files are successfully processed. In case of a re-run, the user can skip these files and run the remaining files. This file is available in the **log** folder. The log file consists of details such as date and time, and the details of the file name. The logging level for this log file is INFO.

The file structure of the **successWrite.log** file is as follows:

```
2017-07-21 14:13:00,616 INFO writelogger:595 /home/testmigration/Documentation/is not in.sql has written successfully.
2017-07-21 14:13:01,055 INFO writelogger:595 /home/testmigration/Documentation/date quotes.sql has written successfully.
2017-07-21 14:13:01,569 INFO writelogger:595 /home/testmigration/Documentation/column alias replace.sql has written successfully.
2017-07-21 14:13:02,055 INFO writelogger:595 /home/testmigration/Documentation/sampleRownum.sql has written successfully.
2017-07-21 14:13:02,597 INFO writelogger:595 /home/testmigration/Documentation/samp.sql has written successfully.
2017-07-21 14:13:03,178 INFO writelogger:595 /home/testmigration/Documentation/testWithNodataSamples.sql has written successfully.
```

6.3.8.3 Perl Migration Logs

The DSC writes all log information to a single file, **perlDSC.log**.

NOTE

Since the DSC will execute the SQL to migrate the SQL scripts inside Perl files, the following **SQL migration logs** are also supported:

- Activity Logging
- Error Logging
- successRead
- successWrite

Logging Levels

The logging level for Perl Migration logs is configured using the **logging-level** parameter.

Logging

DSC saves all logs, warnings and error information to **perlDSC.log**. This file is available in the **log** folder. The log file consists of details such as user who executed the migration and files that have been migrated along with the timestamp.

The structure of the **perlDSC.log** file is as follows:

```
2018-07-08 13:35:10 INFO teradatacore.pm:1316 Extracting SQL contents from perl files started
2018-07-08 13:35:10 INFO teradatacore.pm:1329 Extracting SQL contents from perl files completed
2018-07-08 13:35:10 INFO teradatacore.pm:1331 Migrating SQL files
2018-07-08 13:35:12 INFO teradatacore.pm:1348 Migrating SQL files completed
2018-07-08 13:35:12 INFO teradatacore.pm:1349 Merging migrated SQL contents to perl files started
2018-07-08 13:35:12 INFO teradatacore.pm:1362 Merging migrated SQL contents to perl files completed
2018-07-08 13:35:12 INFO teradatacore.pm:1364 Perl file migration completed
2018-07-08 13:35:32 INFO teradatacore.pm:1316 Extracting SQL contents from perl files started
2018-07-08 13:35:58 ERROR teradatacore.pm:426 opendir ../../../../perltest/ failed
2018-07-08 13:36:17 INFO teradatacore.pm:1316 Extracting SQL contents from perl files started
2018-07-08 13:38:21 INFO teradatacore.pm:1329 Extracting SQL contents from perl files completed
2018-07-08 13:38:21 INFO teradatacore.pm:1331 Migrating SQL files
2018-07-08 13:38:22 INFO teradatacore.pm:1348 Migrating SQL files completed
2018-07-08 13:38:22 INFO teradatacore.pm:1349 Merging migrated SQL contents to perl files started
```



```
2018-07-08 13:38:37 ERROR teradatacore.pm:1044 Directory ../../../../../perltest/ should have 700, but has  0
permission
2018-07-08 13:38:53 ERROR teradatacore.pm:1241 Another migration process is running on same folder, re-
execute after the process has completed
2018-07-08 13:39:01 INFO teradatacore.pm:1316 Extracting SQL contents from perl files started
2018-07-08 13:39:51 INFO teradatacore.pm:1329 Extracting SQL contents from perl files completed
2018-07-08 13:39:51 INFO teradatacore.pm:1331 Migrating SQL files
2018-07-08 13:39:53 INFO teradatacore.pm:1348 Migrating SQL files completed
2018-07-08 13:39:54 INFO teradatacore.pm:1349 Merging migrated SQL contents to perl files started
2018-07-08 13:39:55 INFO teradatacore.pm:1362 Merging migrated SQL contents to perl files completed
2018-07-08 13:39:57 INFO teradatacore.pm:1364 Perl file migration completed
```

6.4 Teradata Syntax Migration

6.4.1 Supported Keywords and Features

Table 6-21 lists the Teradata keywords and features that can be migrated.

- The **Version** column contains the earliest versions that support the keywords or features.
- The **Remarks** column contains the configuration parameters to customize how the migration tool migrates the corresponding keywords or features.

Table 6-21 Supported Teradata keywords and features

Section	Object->Keyword/ Feature		Version	Remarks
Data Type	Data Type		6.5.1, 18.2.0	-
Functions and Operators	Analytical Functions	Analytical Functions in ORDER BY	18.0.0	-
		Analytical Functions in PARTITION BY	18.1.0	-
Functions and Operators	Math Functions	• ** • MOD • NULLIFZERO • ZEROIFNULL	V100R003C00	Configurable: tdMigrateNULLIF-ZERO and tdMigrateZEROIF-NULL (18.0.0)
Functions and Operators	String Functions	CHAR Function CHARACTERS INDEX STRREPLACE OREPLACE	V100R003C00	-

Section	Object->Keyword/ Feature		Version	Remarks
Functions and Operators	Date and Time Functions	DATE TIMESTAMP NEXT	V100R003C00 18.0.0 V100R003C00	-
Functions and Operators	Comparison and List Operators	• ^= and GT • EQ and NE • LE and GE • NOT= and LT • IN and NOT IN • IS NOT IN • LIKE ALL/NOT LIKE ALL • LIKE ANY/NOT LIKE ANY	V100R003C00	-
Functions and Operators	Table Operators	-	18.0.0	-
Functions and Operators	Functions and Operators	QUALIFY	V100R003C00	Configurable: rowstoreToColumn-store
		ALIAS	V100R003C00	Configurable: tdMigrateALIAS
		FORMAT and CAST	V100R003C00	-
		Short Keys Migration	V100R003C00	-
		Migration of Object Names Starting with $	18.0.0	Configurable: tdMigrateDollar
Functions and Operators	Query Optimization Operators	IN and NOT IN Conversion	V100R003C00	Configurable: inToExists

Section	Object->Keyword/ Feature		Version	Remarks
Migrating Tables	Migrating Tables	CREATE TABLE	V100R003C00	Configurable: session_mode
		CHARACTER SET and CASESPECIFIC	18.1.0	Configurable: tdMigrateCharset-Case
		VOLATILE	V100R003C00	Configurable: volatile and session_mode
		SET	V100R003C00	-
		MULTISET	V100R003C00	-
		TITLE	V100R003C00	-
		Indexes	V100R003C00	-
		CONSTRAINT	V100R003C00	-
		COLUMN STORE	V100R003C00	Configurable: rowstoreToColumn-store
		PARTITION	18.0.0	-
		ANALYZE	V100R003C30	-
		Support for Specified Columns	18.0.0	-
Migrating Indexes	Migrating Indexes		V100R003C00	-
Migrating Views	Migrating Views	REPLACE VIEW	V100R003C00	-
		CHECK OPTION	V100R003C00	-
		VIEW WITH RECURSIVE	V100R003C00	-
		VIEW WITH ACCESS LOCK	V100R003C00	-

Section	Object->Keyword/ Feature		Version	Remarks
COLLECT STATISTICS	COLLECT STATISTICS		V100R003C00	-
ACCESS LOCK	ACCESS LOCK		V100R003C00	Configurable: tdMigrateLOCKOption
DBC.COLUMNS	DBC.COLUMNS		18.0.0	Custom DB scripts
DBC.TABLES	DBC.TABLES		18.1.0	-
DBC.INDICES	DBC.INDICES		18.1.0	-
Data Manipulation Language (DML)	SELECT	Order of Clauses	V100R003C00 18.0.0	-
		Extended Group By Clause	V100R003C00	-
		TOP Clauses	V100R003C00	-
Data Manipulation Language (DML)	UPDATE		V100R003C00	-
Data Manipulation Language (DML)	DELETE		V100R003C00	-
Data Manipulation Language (DML)	MERGE		V100R003C00	-
Data Manipulation Language (DML)	NAMED		18.0.0	-

Section	Object->Keyword/ Feature	Version	Remarks
Type Casting and Formatting	CHAR COLUMNS and COLUMN ALIAS Expression INT DATE DAY to SECOND DECIMAL Time Interval NULL Implicit Type Casting Issues	V100R003C00	-

6.4.2 Constraints and Limitations

The restrictions on using DSC to migrate data from TD are as follows:

- DSC is used only for syntax migration and not for data migration.
- If the **SELECT** clause of a subquery contains an aggregate function when the **IN** or **NOT IN** operator is converted to **EXISTS** or **NOT EXISTS**, the migration may fail.

Teradata

- If a **case** statement containing **FORMAT** is not enclosed in parentheses, this statement will not be processed.

  Examples are as follows:

  ```
  case when column1='0' then column1='value' end (FORMAT 'YYYYMMDD')as alias1
  ```

  In this example, **case when column1='0' then column1='value' end** is not enclosed in parentheses and it will not be processed.

- If **SELECT *** and **QUALIFY** clauses are both used in an input query, the migrated query returns an additional column for the **QUALIFY** clause.

  An example is as follows:

  Teradata query

  ```
  SELECT * FROM dwQErrDtl_mc.C03_CORP_TIME_DPSIT_ACCT
  WHERE 1 = 1
  AND Data_Dt = CAST( '20150801' AS DATE FORMAT 'YYYYMMDD' )
  QUALIFY ROW_NUMBER( ) OVER( PARTITION BY Agt_Num, Agt_Modif_Num ORDER BY NULL ) = 1;
  ```

  Query after migration

  ```
  SELECT * FROM (
  SELECT *, ROW_NUMBER( ) OVER( PARTITION BY Agt_Num, Agt_Modif_Num ORDER BY NULL )
  AS ROW_NUM1
  FROM dwQErrDtl_mc.C03_CORP_TIME_DPSIT_ACCT
  WHERE 1 = 1
  AND Data_Dt = CAST( '20150801' AS DATE )
  ) Q1
  WHERE Q1.ROW_NUM1 = 1;
  ```


In the migrated query, the **ROW_NUMBER( ) OVER(PARTITION BY Agt_Num ,Agt_Modif_Num ORDER BY NULL) AS ROW_NUM1** column is returned additionally.

- Named references to a table in a query cannot be migrated from subqueries or functions.

  For example, if the input query contains a table named **foo**, DSC will not migrate any named references to the table from a subquery (**foo.fooid**) or when called from a function (**getfoo(foo.fooid)**).

```
SELECT * FROM foo
 WHERE foosubid IN (
                  SELECT foosubid
                    FROM getfoo(foo.fooid) z
                   WHERE z.fooid = foo.fooid
                   );
```

- The database with the schema name should be changed to **SET SESSION CURRENT_SCHEMA**.

TD Syntax	Syntax After Migration
DATABASE SCHTERA	SET SESSION CURRENT_SCHEMA TO SCHTERA

- The table-specific keyword **MULTISET VOLATILE** is provided in the input file, but the keyword is not supported by GaussDB(DWS). Therefore, the tool replaces it with the **LOCAL TEMPORARY/UNLOGGED** keyword during the migration process. Use the **session_mode** configuration parameter to set the default table type (SET/MULTISET) for CREATE TABLE.

6.4.3 Data Type

Data Type	Input	Output
Numeric	BIGINT	BIGINT
	BYTEINT	SMALLINT
	DECIMAL [(n[,m])]	DECIMAL [(n[,m])]
	DOUBLE PRECISION	DOUBLE PRECISION
	FLOAT	DOUBLE PRECISION
	INT / INTEGER	INTEGER
	NUMBER / NUMERIC	NUMERIC
	NUMBER(n[,m])	NUMERIC (n[,m])
	REAL	REAL
	SMALLINT	SMALLINT
Character	CHAR[(n)] / CHARACTER [(n)]	CHAR(n)
	CLOB	CLOB
	LONG VARCHAR	TEXT

Data Type	Input	Output
	VARCHAR(n) / CHAR VARYING(n) / CHARACTER VARYING(n)	VARCHAR(n)
Date/ Time	DATE	DATE
	TIME [(n)]	TIME [(n)]
	TIME [(n)] WITH TIME ZONE	TIME [(n)] WITH TIME ZONE
	TIMESTAMP [(n)]	TIMESTAMP [(n)]
	TIMESTAMP [(n)] WITH TIME ZONE	TIMESTAMP [(n)] WITH TIME ZONE
Range	PERIOD(DATE)	daterange
	PERIOD(TIME [(n)])	tsrange [(n)]
	PERIOD(TIME WITH TIME ZONE)	tstzrange
	PERIOD(TIMESTAMP [(n)])	tsrange [(n)]
	PERIOD(TIMESTAMP WITH TIME ZONE)	tstzrange
Binary	BLOB[(n)]	blob
	BYTE[(n)]	bytea
	VARBYTE[(n)]	bytea

BYTEINT

Input

```
SELECT cast(col as byteint) FROM tab;
```

Output

```
SELECT CAST( col AS SMALLINT ) FROM tab ;
```

6.4.4 Functions and Operators

6.4.4.1 Analytical Functions

Analytical functions are collectively called ordered analytical functions in Teradata, and they provide powerful analytical abilities for data mining, analysis and business intelligence.

Analytical Functions in ORDER BY

Input: Analytic function in ORDER BY clause


```
SELECT customer_id, customer_name, RANK(customer_id, customer_address DESC)
  FROM customer_t
 WHERE  customer_state = 'CA'
 ORDER BY RANK(customer_id, customer_address DESC);
```

Output:

```
SELECT customer_id, customer_name, RANK() over(order by customer_id, customer_address DESC)
  FROM customer_t
 WHERE  customer_state = 'CA'
 ORDER BY RANK() over(order by customer_id DESC, customer_address DESC) ;
```

Input: Analytic function in GROUP BY clause

```
SELECT customer_city, customer_state, postal_code
    , rank(postal_code)
    , rank() over(partition by customer_state order by postal_code)
   , rank() over(order by postal_code)
 FROM Customer_T
 GROUP BY customer_state
 ORDER BY customer_state;
```

Output:

```
SELECT customer_city, customer_state, postal_code
    , rank() over(PARTITION BY customer_state ORDER BY postal_code DESC)
    , rank() over(partition by customer_state order by postal_code)
    , rank() over(order by postal_code)
 FROM Customer_T
 ORDER BY customer_state;
```

Analytical Functions in PARTITION BY

When the input script contains a numeric value in the PARTITION BY clause, the migrated script retains the numeric value as it is.

Input: Analytic function in PARTITION BY clause (with numeric value)

```
SELECT
        Customer_id
        ,customer_name
        ,rank (
        ) over( partition BY 1 ORDER BY Customer_id )
        ,rank (customer_name)
    FROM
        Customer_t
    GROUP BY
        1
;
```

Output:

```
SELECT
        Customer_id
        ,customer_name
        ,rank (
        ) over( partition BY 1 ORDER BY Customer_id )
        ,rank (
        ) over( PARTITION BY Customer_id ORDER BY customer_name DESC )
    FROM
        Customer_t
;
```

Window Functions

Window functions perform calculations across rows of the query result. DSC supports the following Teradata window functions:

NOTE

The DSC supports only single occurrence of window function in QUALIFY clause. Multiple window functions in a QUALIFY may result in invalid migration.

CSUM

The Cumulative Sum (CSUM) function provides a running or cumulative total for a column's numeric value. It is recommended that ALIAS be used in the QUALIFY statements.

Input - CSUM with GROUP_ID

```
INSERT INTO GSIS_SUM.DW_DAT71 (
  col1
 ,PROD_GROUP
)
  SELECT
    CSUM(1, T1.col1)
    ,T1.PROD_GROUP
   FROM tab1  T1
  WHERE T1.col1 = 'ABC'
;
```

Output:

```
INSERT
    INTO
        GSIS_SUM.DW_DAT71 (
            col1
            ,PROD_GROUP
        ) SELECT
                SUM (1) over( ORDER BY T1.col1 ROWS UNBOUNDED PRECEDING )
                ,T1.PROD_GROUP
            FROM
                tab1 T1
            WHERE
                T1.col1 = 'ABC'
;
```

Input - CSUM with GROUP_ID

```
SELECT  top 10
    CSUM(1, T1.Test_GROUP)
    ,T1.col1
 FROM  $[schema}.  T1
WHERE T1.Test_GROUP = 'Test_group' group by Test_group order by Test_Group;
```

Output:

```
SELECT
     SUM (1) over( partition BY Test_group ORDER BY T1.Test_GROUP ROWS UNBOUNDED PRECEDING )
     ,T1.col1
 FROM
     $[schema}. T1
WHERE
     T1.Test_GROUP = 'Test_group'
ORDER BY
     Test_Group LIMIT 10
;
```

Input - CSUM with GROUP BY + QUALIFY

```
SELECT c1, c2, c3, CSUM(c4, c3)
 FROM tab1
QUALIFY ROW_NUMBER(c4) = 1
GROUP BY 1, 2;
```

Output:


```
SELECT c1, c2, c3, ColumnAlias1
  FROM ( SELECT c1, c2, c3
            , SUM (c4) OVER(PARTITION BY 1 ,2 ORDER BY c3 ROWS UNBOUNDED PRECEDING) AS
ColumnAlias1
            , ROW_NUMBER( ) OVER(PARTITION BY 1, 2 ORDER BY c4) AS ROW_NUM1
          FROM tab1
       ) Q1
  WHERE Q1.ROW_NUM1 = 1;
```

MDIFF

The MDIFF function calculates the moving difference for a column based on the preset query width. The query width is the specified number of rows. It is recommended that ALIAS be used in the QUALIFY statements.

Input: MDIFF with QUALIFY

```
SELECT DT_A.Acct_ID, DT_A.Trade_Date, DT_A.Stat_PBU_ID
     , CAST( MDIFF( Stat_PBU_ID_3, 1, DT_A.Trade_No ASC ) AS DECIMAL(20,0) ) AS MDIFF_Stat_PBU_ID
  FROM Trade_His DT_A
 WHERE Trade_Date >= CAST( '20170101' AS DATE FORMAT 'YYYYMMDD' )
 GROUP BY DT_A.Acct_ID, DT_A.Trade_Date
QUALIFY MDIFF_Stat_PBU_ID <> 0 OR MDIFF_Stat_PBU_ID IS NULL;
```

Output:

```
SELECT Acct_ID, Trade_Date, Stat_PBU_ID, MDIFF_Stat_PBU_ID
  FROM (SELECT DT_A.Acct_ID, DT_A.Trade_Date, DT_A.Stat_PBU_ID
       , CAST( (Stat_PBU_ID_3 - (LAG(Stat_PBU_ID_3, 1, NULL) OVER (PARTITION BY DT_A.Acct_ID,
DT_A.Trade_Date ORDER BY DT_A.Trade_No ASC)))  AS MDIFF_Stat_PBU_ID
         FROM Trade_His DT_A
        WHERE Trade_Date >= CAST( '20170101' AS DATE)
            )
WHERE MDIFF_Stat_PBU_ID <> 0 OR MDIFF_Stat_PBU_ID IS NULL;
```

RANK

RANK(col1, col2...)

Input: RANK with GROUP BY

```
SELECT  c1, c2, c3, RANK(c4, c1 DESC, c3) AS Rank1
 FROM  tab1
WHERE  ...
GROUP BY c1;
```

Output:

```
SELECT c1, c2, c3, RANK() OVER (PARTITION BY c1 ORDER BY c4, c1 DESC ,c3) AS Rank1
 FROM tab1
WHERE ...;
```

ROW_NUMBER

ROW_NUMBER(col1, col2...)

Input: ROW NUMBER with GROUP BY + QUALIFY

```
SELECT c1, c2, c3, ROW_NUMBER(c4, c3)
  FROM tab1
QUALIFY RANK(c4) = 1
 GROUP BY 1, 2;
```

Output:


```
SELECT
      c1
     ,c2
     ,c3
     ,ColumnAlias1
  FROM
      (
        SELECT
              c1
             ,c2
             ,c3
             ,ROW_NUMBER( ) over( PARTITION BY 1 ,2 ORDER BY c4 ,c3 ) AS ColumnAlias1
             ,RANK (
             ) over( PARTITION BY 1 ,2 ORDER BY c4 ) AS ROW_NUM1
          FROM
             tab1
      ) Q1
 WHERE
      Q1.ROW_NUM1 = 1
;
```

COMPRESS (specified with *****)

Input

```
ORDCADBRN VARCHAR(6) CHARACTER SET LATIN CASESPECIFIC TITLE '    ' COMPRESS '******'
```

Output:

```
ORDCADBRN VARCHAR( 6 ) /* CHARACTER SET LATIN*/ /* CASESPECIFIC*/ /*TITLE '    '*/ /* COMPRESS
'******' */
```

6.4.4.2 Math Functions

**

Input: **

```
expr1 ** expr2
```

Output:

```
expr1 ^ expr2
```

MOD

Input: MOD

```
expr1 MOD expr2
```

Output:

```
 expr1 % expr2
```

NULLIFZERO

Use the **tdMigrateNULLIFZERO** configuration parameter to configure migration of NULLIFZERO.

Input: NULLIFZERO

```
SELECT NULLIFZERO(expr1) FROM tab1
WHERE ... ;
```

Output:


```
SELECT NULLIF(expr1, 0) FROM tab1
WHERE ... ;
```

ZEROIFNULL

Use the **tdMigrateZEROIFNULL** configuration parameter to configure migration of ZEROIFNULL.

Input: ZEROIFNULL

```
SELECT ZEROIFNULL(expr1) FROM tab1
WHERE ... ;
```

Output:

```
SELECT COALESCE(expr1, 0) FROM tab1
WHERE ... ;
```

Declaring a Hexadecimal Character Literal Value

Input

```
SELECT
 (COALESCE(TRIM(BOTH FROM VTX_D_RPT_0017_WMSE12_01_01.ID),''))
 ||'7E'xc||(COALESCE(TRIM(BOTH FROM VTX_D_RPT_0017_WMSE12_01_01.Code),''))
 ||'7E'xc||(COALESCE(TRIM(BOTH FROM VTX_D_RPT_0017_WMSE12_01_01.Description),''))
 ||'7E'xc||(COALESCE(TRIM(BOTH FROM VTX_D_RPT_0017_WMSE12_01_01.Name),''))
 ||'7E'xc||(COALESCE(TRIM(BOTH FROM VTX_D_RPT_0017_WMSE12_01_01.Host_Product_Id),''))
FROM DP_VTXEDW.VTX_D_RPT_0017_WMSE12_01_01    VTX_D_RPT_0017_WMSE12_01_01
WHERE 1=1
;
```

Output:

```
SELECT
 (COALESCE(TRIM(BOTH FROM VTX_D_RPT_0017_WMSE12_01_01.ID),''))
 ||E'\x7E'||(COALESCE(TRIM(BOTH FROM VTX_D_RPT_0017_WMSE12_01_01.Code),''))
 ||E'\x7E'||(COALESCE(TRIM(BOTH FROM VTX_D_RPT_0017_WMSE12_01_01.Description),''))
 ||E'\x7E'||(COALESCE(TRIM(BOTH FROM VTX_D_RPT_0017_WMSE12_01_01.Name),''))
 ||E'\x7E'||(COALESCE(TRIM(BOTH FROM VTX_D_RPT_0017_WMSE12_01_01.Host_Product_Id),''))
FROM DP_VTXEDW.VTX_D_RPT_0017_WMSE12_01_01    VTX_D_RPT_0017_WMSE12_01_01
WHERE 1=1
;
```

Declaring a Hexadecimal Binary Literal Value

Input

```
CREATE MULTISET TABLE bvalues (IDVal INTEGER, CodeVal BYTE(2));
INSERT INTO bvalues VALUES (112193, '7879'XB) ;
SELECT IDVal, CodeVal FROM bvalues WHERE CodeVal = '7879'XB ;
```

Output:

```
CREATE TABLE bvalues (IDVal INTEGER, CodeVal BYTEA);
INSERT INTO bvalues VALUES (112193, BYTEA '\x7879') ;
SELECT IDVal, CodeVal FROM bvalues WHERE CodeVal = BYTEA '\x7879' ;
```

6.4.4.3 String Functions

CHAR Function

Input: CHAR

```
CHAR( expression1 )
```


Output:

```
LENGTH( expression1 )
```

CHARACTERS

Input: CHARACTERS

```
CHARACTERS( expression1 )
```

Output:

```
LENGTH( expression1 )
```

INDEX

Input: INDEX

```
SELECT INDEX(expr1/string, substring)
  FROM tab1
WHERE ... ;
```

Output:

```
 SELECT INSTR(expr1/string, substring)
  FROM tab1
WHERE ... ;
```

STRREPLACE

Input: STRREPLACE

```
SELECT STRREPLACE(c2, '.', '')
  FROM tab1
 WHERE ...;
```

Output:

```
SELECT REPLACE(c2, '.', '')
  FROM tab1
 WHERE ...;
```

OREPLACE

Input: OREPLACE

```
SELECT OREPLACE (c2, '.', '')
  FROM tab1
WHERE ... ;
```

Output:

```
 SELECT REPLACE(c2, '.', '')
  FROM tab1
WHERE ... ;
```

STRTOK

Input

```
LENGTH(STRTOK(STRTOK(JOB_NAME_TADD,'-',4),'_',2))
```

Output:

```
LENGTH(split_part(split_part(JOB_NAME_TADD,'-',4),'_',2))
```


6.4.4.4 Date and Time Functions

DATE

DSC supports the migration of Teradata SELECT statements that contain DATE FORMAT, using TO_CHAR to display the date in the source format. This conversion is not done if the date format is an expression (example: Start_Dt + 30) or if the WHERE statement contains an expression (Example: WHERE Start_Dt > End_Dt).

For details, see: **Type Casting to DATE without DATE Keyword**

NOTE

- Migration is supported for SELECT statements with and without column alias.
- Date formatting is not supported in the sub-levels and in inner queries. It is supported only at the outer query level.
- For date formatting, if a table is created with SCHEMA name, subsequent SELECT statements must also include the schema name. In the following example, the table TEMP_TBL in the SELECT statement will not be migrated and the table retained as it was.

```
CREATE TABLE ${SCH}.TEMP_TBL
        (C1 INTEGER
        ,C2 DATE FORMAT 'YYYY-MM-DD')
PRIMARY INDEX(C1,C2);

SELECT ${SCH}.TEMP_TBL.C2 FROM TEMP_TBL where ${SCH}.TEMP_TBL.C2 is not null;
```

Input: DATE FORMAT

```
SELECT
        CASE
            WHEN SUBSTR( CAST( CAST( SUBSTR( '20180631' ,1 ,6 ) || '01' AS DATE FORMAT 'YYYYMMDD' )
+ abc_day - 1 AS FORMAT 'YYYYMMDD' ) ,1 ,6 ) = SUBSTR( '20180631' ,1 ,6 )
            THEN 1
            ELSE 0
        END
    FROM
        tab1
;
```

Output:

```
SELECT
        CASE
            WHEN SUBSTR( TO_CHAR( CAST( SUBSTR( '20180631' ,1 ,6 ) || '01' AS DATE ) + abc_day -
1 ,'YYYYMMDD' ) ,1 ,6 ) = SUBSTR( '20180631' ,1 ,6 )
            THEN 1
            ELSE 0
        END
    FROM
        tab1
;
```

DSC supports migration of the date value. If the input DATE is followed by "YYYY-MM-DD", then the date is not changed in the output. The following examples show conversion of DATE to CURRENT_DATE.

Input: DATE

```
SELECT
       t1.c1
       ,t2.c2
  FROM
       $schema.tab1 t1
       ,$schema.tab2 t2
```



```
 WHERE
       t1.c3 ^ = t1.c3
       AND t2.c4 GT DATE
;
```

Output:

```
SELECT
       t1.c1
       ,t2.c2
  FROM
       "$schema".tab1 t1
       ,"$schema".tab2 t2
 WHERE
       t1.c3 <> t1.c3
       AND t2.c4 > CURRENT_DATE
;
```

Input: DATE with "YYYY-MM-DD"

```
ALTER TABLE
    $abc . tab1 ADD (
        col_date DATE DEFAULT DATE '2000-01-01'
    )
;
```

Output

```
ALTER TABLE
    "$abc" . tab1 ADD (
        col_date DATE DEFAULT DATE '2000-01-01'
    )
;
```

Input: DATE subtraction

```
SELECT
        CAST( T1.Buyback_Mature_Dt - CAST( '${gsTXDate}' AS DATE FORMAT 'YYYYMMDD' ) AS
CHAR( 5 ) )
    FROM
        tab1 T1
    WHERE
        T1.col1 > 10
;
```

Output:

```
SELECT
        CAST( EXTRACT( 'DAY' FROM ( T1.Buyback_Mature_Dt - CAST( '${gsTXDate}' AS DATE ) ) ) AS
CHAR( 5 ) )
    FROM
        tab1 T1
    WHERE
        T1.col1 > 10
;
```

ADD_MONTHS

Input:

```
ADD_MONTHS(CAST(substr(T1.GRANT_DATE,1,8)||'01'AS DATE FORMAT 'YYYY-MM-DD'),1)-1
```

Output:

```
mig_td_ext.ADD_MONTHS(CAST(substr(T1.GRANT_DATE,1,8)||'01'AS DATE FORMAT 'YYYY-MM-DD'),1)-1
```


TIMESTAMP

Input: TIMESTAMP

```
select CAST('20190811'||' '||'01:00:00'
AS TIMESTAMP(0)
FORMAT 'YYYYMMDDBHH:MI:SS'
) ;
```

Output:

```
SELECT TO_TIMESTAMP( '20190811' || ' ' || '01:00:00' ,'YYYYMMDD HH24:MI:SS' ) ;
```

TIME FORMAT

Input:

```
COALESCE(t3.Crt_Tm , CAST('00:00:00' AS TIME FORMAT 'HH:MI:SS'))
COALESCE(LI07_F3EABCTLP.CTLREGTIM,CAST('${NULL_TIME}' AS TIME FORMAT 'HH:MI:sS'))
trim(cast(cast(a.Ases_Orig_Tm as time format'hhmiss') as varchar(10)))
```

Output:

```
CAST('00:00:00' AS TIME FORMAT 'HH:MI:SS')
should be migrated as
SELECT CAST(TO_TIMESTAMP('00:00:00', 'HH24:MI:SS') AS TIME)
---
CAST(abc AS TIME FORMAT 'HH:MI:sS')
=>
CAST(TO_TIMESTAMP(abc, 'HH24:MI:SS') AS TIME)
---
CAST(abc AS TIME FORMAT 'HH:MI:sS')
=>
CAST(TO_TIMESTAMP(abc, 'HH24:MI:SS') AS TIME)
```

TIMESTAMP FORMAT

Input:

```
select
   a.Org_Id as Brn_Org_Id          /*        */
  ,a.Evt_Id as Vst_Srl_Nbr         /*       */
  ,a.EAC_Id as EAC_Id          /*    */
  ,cast(cast(cast(Prt_Tm as timestamp format 'YYYY-MM-DDBHH:MI:SS' ) as varchar(19) )as timestamp(0))
as Tsk_Start_Tm          /*        */
from ${BRTL_VCOR}.BRTL_BC_SLF_TMN_RTL_PRT_JNL as a          /* BC_           */
where   a.DW_Dat_Dt  = CAST('${v_Trx_Dt}' AS DATE FORMAT 'YYYY-MM-DD')  ;
```

Output:

```
SELECT
        a.Org_Id AS Brn_Org_Id /*        */
        ,a.Evt_Id AS Vst_Srl_Nbr /*       */
        ,a.EAC_Id AS EAC_Id /*    */
        ,CAST( CAST( TO_TIMESTAMP( Prt_Tm ,'YYYY-MM-DD HH24:MI:SS' ) AS VARCHAR( 19 ) ) AS
TIMESTAMP ( 0 ) ) AS Tsk_Start_Tm /*        */
    FROM ${BRTL_VCOR}.BRTL_BC_SLF_TMN_RTL_PRT_JNL AS a /* BC_           */
    WHERE
        a.DW_Dat_Dt = CAST( '${v_Trx_Dt}' AS DATE ) ;
```

TIMESTAMP(n) FORMAT

Input:

```
select
   cast('${v_Trx_Dt}' as date format 'yyyy-mm-dd') as DW_Snsh_Dt          /*      */
```



```
  ,coalesce(a.CRE_DAT,cast('0001-01-01 00:00:01' as timestamp(6) format 'yyyy-mm-ddbhh:mi:ssds(6)')) as
Crt_Tm          /*      */
  ,cast('${v_Trx_Dt}' as date format 'yyyy-mm-dd') as DW_ETL_Dt          /*      */
  ,cast(current_date as date format 'yyyy-mm-dd') as DW_Upd_Dt          /*      */
  ,current_time(0) as DW_Upd_Tm          /*      */
  ,1 as DW_Job_Seq          /*      */
from ${NDS_VIEW}.NLV65_MGM_GLDCUS_INF_NEW as a          /*    MGM    */
;
-----------
cast('0001-01-01 00:00:00' as timestamp(6) format 'yyyy-mm-ddbhh:mi:ssds(6)')
TO_TIMESTAMP('0001-01-01 00:00:00', 'yyyy-mm-dd HH24:MI:SS.US' )
----------
cast('0001-01-01 00:00:00.000000' as timestamp(6))
cast('0001-01-01 00:00:00.000000' as timestamp(6))
----------
CAST('0001-01-01 00:00:00.000000' AS TIMESTAMP(6) FORMAT 'YYYY-MM-DDBHH:MI:SS.S(6)')
TO_TIMESTAMP('0001-01-01 00:00:00.000000', 'yyyy-mm-dd HH24:MI:SS.US' )
----------
cast(LA02_USERLOG_M.LOGTIME as TIMESTAMP(6) FORMAT 'YYYY-MM-DD HH:MI:SS.S(0)' )
TO_TIMESTAMP(LA02_USERLOG_M.LOGTIME, 'YYYY-MM-DD HH24:MI:SS' )
----------
cast('0001-01-01 00:00:00' as timestamp(3) format 'yyyy-mm-ddbhh:mi:ssds(3)')
TO_TIMESTAMP('0001-01-01 00:00:00', 'yyyy-mm-dd HH24:MI:SS.MS' )
-----------
CAST( '0001-01-01 00:00:01.000000' AS TIMESTAMP ( 6 ) format 'yyyy-mm-ddbhh:mi:ssds(6)' )
TO_TIMESTAMP('0001-01-01 00:00:01.000000', 'yyyy-mm-dd HH24:MI:SS.US' )
```

Output:

```
cast('0001-01-01 00:00:00' as timestamp(6) format 'yyyy-mm-ddbhh:mi:ssds(6)')
TO_TIMESTAMP('0001-01-01 00:00:00', 'yyyy-mm-dd HH24:MI:SS.US' )
----------
cast('0001-01-01 00:00:00.000000' as timestamp(6))
cast('0001-01-01 00:00:00.000000' as timestamp(6))
----------
CAST('0001-01-01 00:00:00.000000' AS TIMESTAMP(6) FORMAT 'YYYY-MM-DDBHH:MI:SS.S(6)')
TO_TIMESTAMP('0001-01-01 00:00:00.000000', 'yyyy-mm-dd HH24:MI:SS.US' )
----------
cast(LA02_USERLOG_M.LOGTIME as TIMESTAMP(6) FORMAT 'YYYY-MM-DD HH:MI:SS.S(0)' )
TO_TIMESTAMP(LA02_USERLOG_M.LOGTIME, 'YYYY-MM-DD HH24:MI:SS' )
----------
cast('0001-01-01 00:00:00' as timestamp(3) format 'yyyy-mm-ddbhh:mi:ssds(3)')
TO_TIMESTAMP('0001-01-01 00:00:00', 'yyyy-mm-dd HH24:MI:SS.MS' )
-----------
CAST( '0001-01-01 00:00:01.000000' AS TIMESTAMP ( 6 ) format 'yyyy-mm-ddbhh:mi:ssds(6)' )
TO_TIMESTAMP('0001-01-01 00:00:01.000000', 'yyyy-mm-dd HH24:MI:SS.US' )
```

trunc(<date>, 'MM') trunc(<date>, 'YY')

Input:

```
select
   cast('${v_Trx_Dt}' as date format 'yyyy-mm-dd') as DW_Stat_Dt          /*  */
  ,coalesce(d.IAC_Id,'') as IAC_Id          /*  */
  ,coalesce(d.IAC_Mdf,'') as IAC_Mdf          /*  */
  ,coalesce(c.Rtl_Wlth_Prod_Id,'') as Rtl_Wlth_Prod_Id          /*  */
  ,coalesce(c.Ccy_Cd,'') as Ccy_Cd          /*  */
  ,0 as Lot_Bal          /*  */
  ,cast(sum(case when s2.Nvld_Dt > cast('${v_Trx_Dt}' as date format 'yyyy-mm-dd') then s2.Pos_Amt else 0
end) as decimal(18,2)) as NP_Occy_TMKV          /*          */
  ,cast(
       sum(s2.Pos_Amt *
         ((case when s2.Nvld_Dt > cast('${v_Trx_Dt}' as date format 'yyyy-mm-dd')
                  then cast('${v_Trx_Dt}' as date format 'yyyy-mm-dd') else s2.Nvld_Dt - 1 end)
         -
          (case when s2.Eft_Dt > trunc(cast('${v_Trx_Dt}' as date format 'yyyy-mm-dd'),'MM')
             then s2.Eft_Dt else trunc(cast('${v_Trx_Dt}' as date format 'yyyy-mm-dd'),'MM')
          end)
         + 1
```



```
        )
         )
/
```

Output:

```
date_trunc('month', cast('${v_Trx_Dt}' as date))
date_trunc('year', cast('${v_Trx_Dt}' as date))
```

NEXT

Input: NEXT

```
SELECT c1, c2
 FROM tab1
WHERE NEXT(c3) = CAST('2004-01-04' AS DATE FORMAT 'YYYY-MM-DD');
```

Output:

```
SELECT c1, c2
 FROM tab1
WHERE c3 + 1 = CAST('2004-01-04' AS DATE);
```

6.4.4.5 Comparison and List Operators

The following comparison and list operators are supported:

NOTE

The comparison operators LT, LE, GT, GE, EQ, and NE must not be used as TABLE alias or COLUMN alias.

^= and GT

Input: Comparison operations (^= and GT)

```
SELECT t1.c1, t2.c2
 FROM tab1 t1, tab2 t2
WHERE t1.c3 ^= t1.c3
  AND t2.c4 GT 100;
```

Output:

```
SELECT t1.c1, t2.c2
 FROM tab1 t1, tab2 t2
WHERE t1.c3 <> t1.c3
  AND t2.c4 > 100;
```

EQ and NE

Input: Comparison operations (EQ and NE)

```
SELECT t1.c1, t2.c2
 FROM tab1 t1 INNER JOIN tab2 t2
   ON t1.c2 EQ t2.c2
WHERE t1.c6 NE 1000;
```

Output:

```
SELECT t1.c1, t2.c2
 FROM tab1 t1 INNER JOIN tab2 t2
   ON t1.c2 = t2.c2
WHERE
      t1.c6 <> 1000;
```


LE and GE

Input: Comparison operations (LE and GE)

```
SELECT t1.c1, t2.c2
  FROM tab1 t1, tab2 t2
 WHERE t1.c3 LE 200
   AND t2.c4 GE 100;
```

Output:

```
 SELECT t1.c1, t2.c2
   FROM tab1 t1, tab2 t2
  WHERE t1.c3 <= 200
    AND t2.c4 >= 100;
```

NOT= and LT

Input: Comparison operations (NOT= and LT)

```
SELECT t1.c1, t2.c2
  FROM tab1 t1, tab2 t2
 WHERE t1.c3 NOT= t1.c3
   AND t2.c4 LT 100;
```

Output:

```
SELECT t1.c1, t2.c2
  FROM tab1 t1, tab2 t2
 WHERE t1.c3 <> t1.c3
   AND t2.c4 < 100;
```

IN and NOT IN

For details, see **IN and NOT IN Conversion**.

Input: IN and NOT IN

```
 SELECT c1, c2
   FROM tab1
  WHERE c1 IN 'XY';
```

Output:

```
SELECT c1, c2
  FROM tab1
 WHERE c1 = 'XY';
```

NOTE

GaussDB(DWS) does not support **IN** and **NOT IN** operators in some specific scenarios.

IS NOT IN

Input: IS NOT IN

```
SELECT c1, c2
  FROM tab1
 WHERE c1 IS NOT IN (subquery);
```

Output:

```
SELECT c1, c2
  FROM tab1
 WHERE c1 NOT IN (subquery);
```


LIKE ALL/NOT LIKE ALL

Input: LIKE ALL / NOT LIKE ALL

```
SELECT c1, c2
  FROM tab1
 WHERE c3 NOT LIKE ALL ('%STR1%', '%STR2%', '%STR3%');
```

Output:

```
SELECT c1, c2
  FROM tab1
 WHERE c3 NOT LIKE ALL (ARRAY[ '%STR1%', '%STR2%', '%STR3%' ]);
```

LIKE ANY/NOT LIKE ANY

Input: LIKE ANY / NOT LIKE ANY

```
SELECT c1, c2
  FROM tab1
 WHERE c3 LIKE ANY ('STR1%', 'STR2%', 'STR3%');
```

Output:

```
SELECT c1, c2
  FROM tab1
 WHERE c3 LIKE ANY (ARRAY[ 'STR1%', 'STR2%', 'STR3%' ]);
```

6.4.4.6 Table Operators

The functions that can be called in the FROM clause of a query are from the table operator.

Input: Table operator with RETURNS

```
SELECT *
  FROM TABLE( sales_retrieve (9005) RETURNS ( store INTEGER, item CLOB, quantity BYTEINT) ) AS ret;
```

Output:

```
SELECT *
  FROM sales_retrieve(9005) AS ret (store, item, quantity);
```

6.4.4.7 Query Optimization Operators

This section describes the syntax for migrating Teradata query optimization operators. The migration syntax determines how the keywords and features are migrated.

Use the **inToExists** parameter to configure the migration from **IN** or**NOT IN** to **EXISTS** or **NOT EXISTS**.

This parameter defaults to **FALSE**. To enable the query optimization feature, this parameter must be set to **TRUE**.

When being converted to GaussDB(DWS) SQL queries, Teradata queries containing the **IN** and **NOT IN** operators have been optimized, and **IN** and **NOT IN** have been converted to **EXISTS** and **NOT EXISTS**, respectively. The **IN** and **NOT IN** operators support single or multiple columns. DSC will migrate the **IN** or **NOT IN** statement only when it exists in the **WHERE** or **ON** clause. The following example shows the conversion from **IN** to **EXISTS**, which is also applicable to the conversion from **NOT IN** to **NOT EXISTS**.

Simple conversion from IN to EXISTS

In the following example, the keyword **IN** is provided in the input file. During the migration, DSC replaces **IN** with **EXISTS** to optimize query performance.

NOTE

- The **IN** and **NOT IN** statements with nested **IN** and **NOT IN** keywords cannot be migrated. In this case, the scripts will be invalid after migration.

```
UPDATE tab1
   SET b = 123
 WHERE b IN ('abc')
   AND b IN ( SELECT  i
              FROM tab2
              WHERE j NOT IN (SELECT m
                              FROM tab3
                             )
            )
;
```

  When an **IN** or **NOT IN** statement containing subqueries is being migrated, comments between the **IN** or **NOT IN** operator and the subqueries (see the example) cannot be migrated.

  Example:

```
SELECT *
  FROM categories
 WHERE category_name
    IN --comment
       ( SELECT category_name
         FROM categories1 )
 ORDER BY category_name;
```

- **Migrating IN or NOT IN statements whose object names contain $ and #**
  - DSC will not migrate the query if the **TABLE** name or **TABLE ALIAS** starts with **$**.

```
SELECT Customer_Name
  FROM Customer_t $A
 WHERE Customer_ID IN( SELECT Customer_ID FROM Customer_t );
```

  - If the **COLUMN** name starts with **#**, DSC may fail to migrate the query.

```
SELECT Customer_Name
  FROM Customer_t
 WHERE #Customer_ID IN( SELECT #Customer_ID FROM Customer_t );
```

Input: IN

```
SELECT ...
   FROM tab1 t
   WHERE t.col1 IN (SELECT icol1 FROM tab2 e)
 ORDER BY col1
```

Output:

```
SELECT ...
   FROM tab1 t
   WHERE EXISTS (SELECT icol1
                 FROM tab2 e
                WHERE icol1 = t.col1
               )
 ORDER BY col1;
```

Input: IN with multiple columns and Aggregate functions

```
SELECT deptno, job_id, empno, salary, bonus
   FROM emp_t
  WHERE ( deptno, job_id, CAST(salary AS NUMBER(10,2))+CAST(bonus AS NUMBER(10,2)) )
              IN ( SELECT deptno, job_id,
                   MAX(CAST(salary AS NUMBER(10,2))+CAST(bonus AS NUMBER(10,2)))
                     FROM emp_t
                     WHERE hire_dt >= CAST( '20170101' AS DATE FORMAT 'YYYYMMDD' )
```



```
                  GROUP BY deptno, job_id )
  AND hire_dt IS NOT NULL;
```

Output:

```
SELECT deptno, job_id, empno, salary, bonus
  FROM emp_t MAlias1
 WHERE EXISTS ( SELECT deptno, job_id,
                       MAX(CAST(salary AS NUMBER(10,2))+CAST(bonus AS NUMBER(10,2)))
                  FROM emp_t
                 WHERE hire_dt >= CAST( '20170101' AS DATE)
                     AND deptno  = MAlias1.deptno
                     AND job_id   = MAlias1.job_id
                 GROUP BY deptno, job_id
                HAVING MAX(CAST(salary AS NUMBER(10,2))+CAST(bonus AS NUMBER(10,2)))
                       = CAST(MAlias1.salary AS NUMBER(10,2))+CAST(MAlias1.bonus AS NUMBER(10,2)) )
  AND hire_dt IS NOT NULL;
```

6.4.4.8 QUALIFY

In general, the **QUALIFY** clause is accompanied by analytic functions (window functions) such as CSUM(), MDIFF(), ROW_NUMBER() and RANK(). This is addressed using sub-query that contains the window functions specified in the **QUALIFY** clause. Migration tools support **QUALIFY** with **MDIFF()**, **RANK()** and **ROW_NUMBER()**. **QUALIFY** is a Teradata extension and not an ANSI standard syntax. It is executed after the WHERE and GROUP BY clauses. QUALIFY must start in new line.

NOTE

Migration tools support column name and/or expressions in the ORDER BY clause only if the column name and/or expression is explicitly included in the SELECT statement as well.

Input: QUALIFY

```
SELECT
       CUSTOMER_ID
      ,CUSTOMER_NAME
  FROM
      CUSTOMER_T QUALIFY row_number( ) Over( partition BY CUSTOMER_ID ORDER BY POSTAL_CODE
DESC ) = 1
;
```

Output:

```
SELECT
        CUSTOMER_ID
        ,CUSTOMER_NAME
    FROM
        (
            SELECT
                    CUSTOMER_ID
                    ,CUSTOMER_NAME
                    ,row_number( ) Over( partition BY CUSTOMER_ID ORDER BY POSTAL_CODE DESC ) AS
ROW_NUM1
                FROM
                    CUSTOMER_T
        ) Q1
    WHERE
        Q1.ROW_NUM1 = 1
;
```

Input: QUALIFY with MDIFF and RANK

```
SELECT
         material_name
```



```
        ,unit_of_measure * standard_cost AS tot_cost
    FROM
        raw_material_t m LEFT JOIN supplies_t s
            ON s.material_id = m.material_id
        QUALIFY rank ( ) over( ORDER BY tot_cost DESC ) IN '5'
              OR mdiff( tot_cost ,3 ,material_name ) IS NULL
;
```

Output:

```
SELECT
        material_name
        ,tot_cost
    FROM
        (
            SELECT
                    material_name
                    ,unit_of_measure * standard_cost AS tot_cost
                    ,rank ( ) over( ORDER BY unit_of_measure * standard_cost DESC ) AS ROW_NUM1
                    ,unit_of_measure * standard_cost - (LAG( unit_of_measure * standard_cost ,3 ,NULL )
over( ORDER BY material_name )) AS ROW_NUM2
                FROM
                    raw_material_t m LEFT JOIN supplies_t s
                        ON s.material_id = m.material_id
        ) Q1
    WHERE
        Q1.ROW_NUM1 = '5'
        OR Q1.ROW_NUM2 IS NULL
;
```

Input: QUALIFY with ORDER BY having columns that do not exist in the SELECT list

```
SELECT Postal_Code
  FROM db_pvfc9_std.Customer_t t1
  GROUP BY Customer_Name ,Postal_Code
  QUALIFY ---comments
(  Rank ( CHAR(Customer_Address) DESC  )  ) = 1
 ORDER BY t1.Customer_Name;
```

Output:

```
SELECT Postal_Code FROM
         ( SELECT Customer_Name, Postal_Code
   , Rank () over( PARTITION BY Customer_Name, Postal_Code ORDER BY LENGTH(Customer_Address)
DESC ) AS Rank_col
             FROM db_pvfc9_std.Customer_t t1
         ) Q1
     WHERE /*comments*/
   Q1.Rank_col = 1
 ORDER BY Q1.Customer_Name;
```

Input: QUALIFY with COLUMN ALIAS - the corresponding column expression should not be added again in SELECT list.

```
SELECT material_name, unit_of_measure * standard_cost as tot_cost,
       RANK() over(order by tot_cost desc) vendor_cnt
 FROM raw_material_t m left join supplies_t s
 ON s.material_id = m.material_id
 QUALIFY vendor_cnt < 5 or MDIFF(tot_cost, 3, material_name) IS NULL;
```

Output:

```
SELECT material_name, tot_cost, vendor_cnt
  FROM  ( SELECT material_name
                , unit_of_measure * standard_cost AS tot_cost
                , rank () over (ORDER BY tot_cost DESC) vendor_cnt
                , tot_cost - ( LAG(tot_cost ,3 ,NULL) over (ORDER BY material_name) ) AS anltfn
            FROM raw_material_t m LEFT JOIN supplies_t s
```



```
                ON s.material_id = m.material_id
        ) Q1
    WHERE  Q1.vendor_cnt < 5 OR Q1.anltfn IS NULL
;
```

TITLE with QUALIFY

Input:

```
REPLACE VIEW ${STG_VIEW}.LP06_BMCLIINFP${v_Table_Suffix_Inc}
(
  CLICLINBR
, CLICHNNAM
, CLICHNSHO
, CLICLIMNE
, CLIBNKCOD
)
AS
LOCKING ${STG_DATA}.LP06_BMCLIINFP${v_Table_Suffix_Inc} FOR ACCESS
SELECT
  CLICLINBR (title '    VARCHAR(20)')
, CLICHNNAM (title '        VARCHAR(200)')
, CLICHNSHO (title '        VARCHAR(20)')
, CLICLIMNE (title '      VARCHAR(10)')
, CLIBNKCOD (title '       VARCHAR(11)')
FROM
  ${STG_DATA}.LP06_BMCLIINFP${v_Table_Suffix_Inc} s1
QUALIFY
    ROW_NUMBER() OVER(PARTITION BY  CLICLINBR ORDER BY CLICLINBR  ) = 1
;
```

Output:

```
CREATE OR REPLACE VIEW ${STG_VIEW}.LP06_BMCLIINFP${v_Table_Suffix_Inc}
(
  CLICLINBR
, CLICHNNAM
, CLICHNSHO
, CLICLIMNE
, CLIBNKCOD
)
AS
/* LOCKING ${STG_DATA}.LP06_BMCLIINFP${v_Table_Suffix_Inc} FOR ACCESS */
SELECT CLICLINBR
        , CLICHNNAM
       , CLICHNSHO
       , CLICLIMNE
      , CLIBNKCOD
 FROM (
          SELECT
                      CLICLINBR /* (title '    VARCHAR(20)') */
                    , CLICHNNAM /* (title '        VARCHAR(200)') */
                    , CLICHNSHO /* (title '        VARCHAR(20)') */
                    , CLICLIMNE /* (title '      VARCHAR(10)') */
                    , CLIBNKCOD /* (title '       VARCHAR(11)') */
                    , ROW_NUMBER() OVER(PARTITION BY  CLICLINBR ORDER BY CLICLINBR  ) AS ROWNUM1
FROM
  ${STG_DATA}.LP06_BMCLIINFP${v_Table_Suffix_Inc} s1 ) Q1
WHERE Q1.ROWNUM1 = 1
;
```

6.4.4.9 ALIAS

ALIAS is supported by all databases. In Teradata, an **ALIAS** can be referred in **SELECT** and **WHERE** clauses of the same statement where the alias is defined. Since **ALIAS** is not supported in **SELECT** and **WHERE** clauses in the target, it is replaced by the defined value/expression.

NOTE

- The comparison operators LT, LE, GT, GE, EQ, and NE must not be used as TABLE alias or COLUMN alias.
- The tool supports column ALIAS. If the ALIAS is the same as the column name, the ALIAS is specified only for that column and not for other columns in the table. In the following example, the column name **DATA_DT** conflicts with the alias **DATA_DT**, which is not supported by the tool.
  SELECT **DATA_DT**,DATA_INT AS **DATA_DT** FROM KK WHERE **DATA_DT**=DATE;

Input: ALIAS

```
SELECT
        expression1 (
            TITLE 'Expression 1'
        ) AS alias1
        ,CASE
            WHEN alias1 + Cx >= z
            THEN 1
            ELSE 0
        END AS alias2
    FROM
        tab1
    WHERE
        alias1 = y
;
```

Output: tdMigrateALIAS = FALSE

```
SELECT
        expression1 AS alias1
        ,CASE
            WHEN alias1 + Cx >= z
            THEN 1
            ELSE 0
        END AS alias2
    FROM
        tab1
    WHERE
        alias1 = y
;
```

Output: tdMigrateALIAS = TRUE

```
SELECT
        expression1 AS alias1
        ,CASE
            WHEN expression1 + Cx >= z
            THEN 1
            ELSE 0
        END AS alias2
    FROM
        tab1
    WHERE
        expression1 = y
;
```

6.4.4.10 FORMAT and CAST

In Teradata, the **FORMAT** keyword is used for formatting a column/expression. For example, FORMAT '9(n)' and 'z(n)' are addressed using LPAD with 0 and space (' ') respectively.

Data typing is done using **CAST** or direct data type [like (expression1)(CHAR(n))]. This feature is addressed using **CAST**. For details, see **Type Casting and Formatting**.

Input: FORMAT with CAST

```
SELECT
        CAST(TRIM( Agt_Num ) AS DECIMAL( 5 ,0 ) FORMAT '9(5)' )
    FROM
        C03_AGENT_BOND
;

SELECT
        CAST(CAST( Agt_Num AS INT FORMAT 'Z(17)' ) AS CHAR( 5 ) )
    FROM
        C03_AGENT_BOND
;

SELECT
        CHAR(CAST( CAST( CND_VLU AS DECIMAL( 17 ,0 ) FORMAT 'Z(17)' ) AS VARCHAR( 17 ) ) )
    FROM
        C03_AGENT_BOND
;
```

Output:

```
SELECT
        LPAD( CAST( TRIM( Agt_Num ) AS DECIMAL( 5 ,0 ) ) ,5 ,'0' ) AS Agt_Num
    FROM
        C03_AGENT_BOND
;
SELECT
CAST(CAST( Agt_Num AS INT FORMAT 'Z(17)' ) AS CHAR( 5 ) )
FROM
C03_AGENT_BOND
;

SELECT
        LENGTH( CAST( LPAD( CAST( CND_VLU AS DECIMAL( 17 ,0 ) ) ,17 ,' ' ) AS VARCHAR( 17 ) ) ) AS
CND_VLU
    FROM
        C03_AGENT_BOND
;
```

Input - FORMAT 'Z(n)9'

```
SELECT
        standard_price (FORMAT 'Z(5)9') (CHAR( 6 ))
        ,max_price (FORMAT 'ZZZZZ9') (CHAR( 6 ))
    FROM
        product_t
;
```

Output:

```
SELECT
        CAST( TO_CHAR( standard_price ,'999990' ) AS CHAR( 6 ) ) AS standard_price
        ,CAST( TO_CHAR( max_price ,'999990' ) AS CHAR( 6 ) ) AS max_price
    FROM
        product_t
;
```

Input - FORMAT 'z(m)9.9(n)'

```
SELECT
        standard_price (FORMAT 'Z(6)9.9(2)') (CHAR( 6 ))
    FROM
        product_t
;
```

Output:

```
SELECT
        CAST( TO_CHAR( standard_price ,'9999990.00' ) AS CHAR( 6 ) ) AS standard_price
    FROM
```



```
        product_t
;
```

Input - CAST AS INTEGER

```
SELECT
        CAST( standard_price AS INTEGER )
    FROM
        product_t
;
```

Output:

```
SELECT
        (standard_price)
    FROM
        product_t
;
```

Input - CAST AS INTEGER FORMAT

```
SELECT
        CAST( price11 AS INTEGER FORMAT 'Z(4)9' ) (
            CHAR( 10 )
        )
    FROM
        product_t
;
```

Output:

```
SELECT
        CAST( TO_CHAR(   ( price11 ) ,'99990' ) AS CHAR( 10 ) ) AS price11
    FROM
        product_t
;
```

NOTE

The following GaussDB(DWS) function is added to convert to integer:

```
CREATE OR REPLACE FUNCTION
/*  This function is used to support "CAST AS INTEGER" of Teradata.
    It should be created in the "mig_td_ext" schema.
*/
    ( i_param                       TEXT )
RETURN INTEGER
AS
 v_castasint    INTEGER;
BEGIN

  v_castasint := CASE WHEN i_param IS NULL
                                        THEN NULL        -- if NULL value is provided as input
                                                WHEN TRIM(i_param) IS NULL
                                                        THEN 0                -- if empty string with one
or more spaces is provided
                                                ELSE TRUNC(CAST(i_param AS NUMBER))           -- if any
numeric value is provided
                                END;

RETURN v_castasint;
END;
```

6.4.4.11 Short Keys Migration

Table 6-22 lists the short keys supported by Teradata and their equivalent syntax in GaussDB(DWS).

Table 6-22 List of short keys

Teradata Short Key	GaussDB(DWS) Syntax
SEL	SELECT
INS	INSERT
UPD	UPDATE
DEL	DELETE
CT	CREATE TABLE
CV	CREATE VIEW
BT	START TRANSACTION
ET	COMMIT

Input - BT

```
BT;
--
delete from ${BRTL_DCOR}.BRTL_CS_CUST_CID_UID_REL
where DW_Job_Seq = ${v_Group_No};

.if ERRORCODE <> 0 then .quit 12;

--
insert into ${BRTL_DCOR}.BRTL_CS_CUST_CID_UID_REL
(
   Cust_Id
  ,Cust_UID
  ,DW_Upd_Dt
  ,DW_Upd_Tm
  ,DW_Job_Seq
  ,DW_Etl_Dt
)
select
   a.Cust_Id
  ,a.Cust_UID
  ,current_date as Dw_Upd_Dt
  ,current_time(0) as DW_Upd_Tm
  ,cast(${v_Group_No} as byteint) as DW_Job_Seq
  ,cast('${v_Trx_Dt}' as date format 'yyyy-mm-dd') as DW_Etl_Dt
from ${BRTL_VCOR}.BRTL_CS_CUST_CID_UID_REL_S a
where a.DW_Snsh_Dt = cast('${v_Trx_Dt}' as date format 'yyyy-mm-dd');

.if ERRORCODE <> 0 then .quit 12;

ET;cd ..
```

Output:

```
BEGIN
--
   BEGIN
      delete from ${BRTL_DCOR}.BRTL_CS_CUST_CID_UID_REL
       where DW_Job_Seq = ${v_Group_No};
         lv_mig_errorcode = 0;
   EXCEPTION
      WHEN OTHERS THEN
         lv_mig_errorcode = -1;
   END;
```



```
  IF lv_mig_errorcode <> 0 THEN
      RAISE EXCEPTION '12';
  END IF;

--
  BEGIN
      insert into ${BRTL_DCOR}.BRTL_CS_CUST_CID_UID_REL
      (
        Cust_Id
       ,Cust_UID
       ,DW_Upd_Dt
       ,DW_Upd_Tm
       ,DW_Job_Seq
       ,DW_Etl_Dt
      )
     select
         a.Cust_Id
       ,a.Cust_UID
       ,current_date as Dw_Upd_Dt
       ,current_time(0) as DW_Upd_Tm
       ,cast(${v_Group_No} as byteint) as DW_Job_Seq
       ,cast('${v_Trx_Dt}' as date format 'yyyy-mm-dd') as DW_Etl_Dt
     from ${BRTL_VCOR}.BRTL_CS_CUST_CID_UID_REL_S a
    where a.DW_Snsh_Dt = cast('${v_Trx_Dt}' as date format 'yyyy-mm-dd');
  EXCEPTION
    WHEN OTHERS THEN
      lv_mig_errorcode = -1;
  END;

  IF lv_mig_errorcode <> 0 THEN
      RAISE EXCEPTION '12';
  END IF;

END;
```

6.4.4.12 Migration of Object Names Starting with $

This section describes the migration of object names starting with $.

The migration behavior for object names starting with $ is explained in the following table. Use the **tdMigrateDollar** configuration parameter to configure the behavior.

For details, see **IN and NOT IN Conversion**.

Table 6-23 Migration of object names starting with $

tdMigrateDollar Setting	Object Name	Migrated To
true	$V_SQL Static object name starting with $	"$V_SQL"
true	${V_SQL} Dynamic object name starting with $	${V_SQL} No change: Dynamic object names not supported

tdMigrateDollar Setting	Object Name	Migrated To
false	$V_SQL Static object name starting with $	$V_SQL No change: Configuration parameter is set to **false**.
false	${V_SQL} Dynamic object name starting with $	${V_SQL} No change: Configuration parameter is set to **false**.

NOTE

Any variable starting with $ is considered as a Value. The tool will migrate this by adding ARRAY.

Input - OBJECT STARTING WITH $

```
SELECT $C1 from p11 where $C1 NOT LIKE ANY ($sql1);
```

Output (tdMigrateDollar = TRUE)

```
SELECT
	"$C1"
FROM
	p11
WHERE
	"$C1" NOT LIKE ANY (
	ARRAY[ "$sql1" ]
	)
;
```

Output (tdMigrateDollar = FALSE)

```
SELECT
	$C1
FROM
	p11
WHERE
	$C1 NOT LIKE ANY (
	ARRAY[ $sql1 ]
	)
;
```

Input - Value starting with $ in LIKEALL/LIKE ANY

```
SELECT * FROM T1
WHERE T1.Event_Dt>=ADD_MONTHS(CAST('${OUT_DATE}' AS DATE FORMAT 'YYYYMMDD')+1,
(-1)*CAST(T7.Tm_Range_Month AS INTEGER))
  AND T1.Event_Dt<=CAST('${OUT_DATE}' AS DATE FORMAT 'YYYYMMDD')
  AND T1.Cntpty_Acct_Name NOT LIKE ALL ( SELECT Tx_Cntpty_Name_Key FROM TEMP_NAME )
  AND T1.Cntpty_Acct_Name NOT LIKE ANY ( SELECT Tx_Cntpty_Name_Key FROM TEMP_NAME )
  AND T1.Cntpty_Acct_Name LIKE ALL ( SELECT Tx_Cntpty_Name_Key FROM TEMP_NAME )
  AND T1.Cntpty_Acct_Name LIKE ANY ( SELECT Tx_Cntpty_Name_Key FROM TEMP_NAME )
  AND T1.Col1 NOT LIKE ANY ($sql1)
  AND T1.Col1 NOT LIKE ALL ($sql1)
  AND T1.Col1 LIKE ANY ($sql1)
  AND T1.Col1 LIKE ALL ($sql1);
```

Output:

```
SELECT
	*
```



```
    FROM
        T1
    WHERE
        T1.Event_Dt >= ADD_MONTHS (CAST( '${OUT_DATE}' AS DATE ) + 1 ,(- 1 ) *
CAST( T7.Tm_Range_Month AS INTEGER ))
        AND T1.Event_Dt <= CAST( '${OUT_DATE}' AS DATE )
        AND T1.Cntpty_Acct_Name NOT LIKE ALL (
            SELECT
                    Tx_Cntpty_Name_Key
                FROM
                    TEMP_NAME
        )
        AND T1.Cntpty_Acct_Name NOT LIKE ANY (
            SELECT
                    Tx_Cntpty_Name_Key
                FROM
                    TEMP_NAME
        )
        AND T1.Cntpty_Acct_Name LIKE ALL (
            SELECT
                    Tx_Cntpty_Name_Key
                FROM
                    TEMP_NAME
        )
        AND T1.Cntpty_Acct_Name LIKE ANY (
            SELECT
                    Tx_Cntpty_Name_Key
                FROM
                    TEMP_NAME
        )
        AND T1.Col1 NOT LIKE ANY (
            ARRAY[ "$sql1" ]
        )
        AND T1.Col1 NOT LIKE ALL (
            ARRAY[ "$sql1" ]
        )
        AND T1.Col1 LIKE ANY (
            ARRAY[ "$sql1" ]
        )
        AND T1.Col1 LIKE ALL (
            ARRAY[ "$sql1" ]
        )
;
```

QUALIFY, CASE, and ORDER BY

Input:

```
select
   a.Cust_UID as Cust_UID          /*   UID */
  ,a.Rtl_Usr_Id as Ini_CM          /*       */
  ,a.Cntr_Aprv_Dt as Aprv_Pass_Tm          /*       */
  ,a.Blg_Org_Id as CM_BRN_Nbr          /*       */
  ,a.Mng_Chg_Typ_Cd  as MNG_CHG_TYP_CD          /*          */
  ,case when a.Blg_Org_Id = b.BRN_Org_Id and a.Mng_Chg_Typ_Cd= 'PMD' and a.Pst_Id in
('PB0101','PB0104') then 'Y' ----         ,
       when a.Blg_Org_Id = b.BRN_Org_Id and a.Mng_Chg_Typ_Cd= 'DEVPMD' and a.Pst_Id ='PB0106'  then
'Y' ----
       when a.Blg_Org_Id = b.BRN_Org_Id and a.Mng_Chg_Typ_Cd= 'DMD' and a.Pst_Id in ('PB0201','PB0204')
then 'Y' ----         ,
      when a.Blg_Org_Id = b.BRN_Org_Id and a.Mng_Chg_Typ_Cd= 'DEVDMD' and a.Pst_Id ='PB0109'  then 'Y'
----           ,
       else ''
end  as Pst_Flg          /*      */
  ,a.Pst_Id as Pst_Id          /*      */
  ,a.BBK_Org_Id  as BBK_Org_Id          /*       */
from VT_CUID_MND_NMN_CHG_INF as a          /* VT_         */
LEFT OUTER JOIN ${BRTL_VCOR}.BRTL_EM_USR_PST_REL_INF_S as b          /* EM_          */
  on  a.Rtl_Usr_Id = b.Rtl_Usr_Id
```



```
 AND a.Blg_Org_Id  = b.BRN_Org_Id
 AND a.Pst_Id = b.Pst_Id
 AND b.Sys_Id = 'privatebanking'
 AND b.pst_sts IN ('1','0','-2') /*     1   -2   0  */
 AND b.DW_Snsh_Dt =  cast('${v_Trx_Dt}' as date format 'yyyy-mm-dd')
qualify   row_number() over(partition by a.Cust_UID,a.bbk_org_id order by
case when ( a.Mng_Chg_Typ_Cd= 'PMD'    and   a.Pst_Id in ('PB0101','PB0104')) or ( a.Mng_Chg_Typ_Cd=
'DEVPMD'    and   a.Pst_Id ='PB0106')
then 0 when  (a.Mng_Chg_Typ_Cd= 'DMD' and a.Pst_Id in ('PB0201','PB0204')) or    (a.Mng_Chg_Typ_Cd=
'DEVDMD' and a.Pst_Id ='PB0109 ')   then 0 else 1 end  asc )  = 1
;
```

Output:

```
SELECT
                Cust_UID AS Cust_UID /*   UID */
                ,Ini_CM /*        */
                ,Aprv_Pass_Tm /*        */
                ,CM_BRN_Nbr /*        */
                ,MNG_CHG_TYP_CD /*          */
                ,Pst_Flg /*      */
                ,Pst_Id AS Pst_Id /*      */
                ,BBK_Org_Id AS BBK_Org_Id /*        */
            FROM
                ( SELECT
                    a.Cust_UID AS Cust_UID /*   UID */
                    ,a.Rtl_Usr_Id AS Ini_CM /*        */
                    ,a.Cntr_Aprv_Dt AS Aprv_Pass_Tm /*        */
                    ,a.Blg_Org_Id AS CM_BRN_Nbr /*        */
                    ,a.Mng_Chg_Typ_Cd AS MNG_CHG_TYP_CD /*          */
                    ,CASE WHEN a.Blg_Org_Id = b.BRN_Org_Id AND a.Mng_Chg_Typ_Cd = 'PMD' AND a.Pst_Id
IN ( 'PB0101' ,'PB0104' )
                            THEN 'Y' /*         , */
                        WHEN a.Blg_Org_Id = b.BRN_Org_Id AND a.Mng_Chg_Typ_Cd = 'DEVPMD' AND
a.Pst_Id = 'PB0106'
                            THEN 'Y' /*  */
                        WHEN a.Blg_Org_Id = b.BRN_Org_Id AND a.Mng_Chg_Typ_Cd = 'DMD' AND a.Pst_Id
IN ( 'PB0201' ,'PB0204' )
                            THEN 'Y' /*         , */
                        WHEN a.Blg_Org_Id = b.BRN_Org_Id AND a.Mng_Chg_Typ_Cd = 'DEVDMD' AND
a.Pst_Id = 'PB0109'
                            THEN 'Y' /*             , */
                    ELSE
                        ''
                    END AS Pst_Flg /*      */
                    ,a.Pst_Id AS Pst_Id /*      */
                    ,a.BBK_Org_Id AS BBK_Org_Id /*        */
                    ,row_number( ) over( partition BY a.Cust_UID
                    ,a.bbk_org_id
                ORDER BY
                    CASE WHEN( a.Mng_Chg_Typ_Cd = 'PMD' AND Q1.Pst_Id IN ( 'PB0101' ,'PB0104' ) )
OR( Q1.Mng_Chg_Typ_Cd = 'DEVPMD' AND a.Pst_Id = 'PB0106' )
                            THEN 0
                        WHEN( a.Mng_Chg_Typ_Cd = 'DMD' AND Q1.Pst_Id IN ( 'PB0201' ,'PB0204' ) )
OR( Q1.Mng_Chg_Typ_Cd = 'DEVDMD' AND a.Pst_Id = 'PB0109 ' )
                            THEN 0
                    ELSE
                        1
                    END ASC ) AS ROW_NUM1
                FROM
                    VT_CUID_MND_NMN_CHG_INF AS a /* VT_         */
                    LEFT OUTER JOIN BRTL_VCOR.BRTL_EM_USR_PST_REL_INF_S AS b /* EM_           */
                        ON a.Rtl_Usr_Id = b.Rtl_Usr_Id
                    AND a.Blg_Org_Id = b.BRN_Org_Id
                    AND a.Pst_Id = b.Pst_Id
                    AND b.Sys_Id = 'privatebanking'
                    AND b.pst_sts IN ( '1' ,'0' ,'-2' ) /*     1   -2   0  */
                    AND b.DW_Snsh_Dt = CAST( lv_mig_v_Trx_Dt AS DATE ) ) Q1
                WHERE
                    Q1.ROW_NUM1 = 1 ;
```


6.4.5 Migrating Tables

6.4.5.1 CREATE TABLE

The Teradata **CREATE TABLE** (**short key** CT) statements are used to create new tables.

Example:

Input: CREATE TABLE

```
CT tab1 (
    id INT
);
```

Output:

```
CREATE
    TABLE
        tab1 (
            id INTEGER
        )
;
```

When **CREATE tab2 AS tab1** is executed, the structure copied from **tab1** is used to create table tab2. If the CREATE TABLE statement includes WITH DATA operator, then the data from *tab1* is also copied into *tab2*. When CREATE AS is used, the CONSTRAINT row in the source table is retained in the new table.

- If **session_mode** is set to *Teradata*, duplicate records in the target table must be deleted. This is done by adding the **MINUS** operator in the migrated scripts.
- If **session_mode** is set to *ANSI*, duplicate records are allowed in the target table.

If the source table has a PRIMARY KEY or a UNIQUE CONSTRAINT, then it will not contain any duplicate records. In this case, the MINUS operator is not required or added to remove duplicate records.

Example:

Input: CREATE TABLE AS with DATA (session_mode=Teradata)

```
CREATE TABLE tab2
   AS tab1 WITH DATA;
```

Output:

```
BEGIN
   CREATE TABLE tab2 (
         LIKE tab1 INCLUDING ALL EXCLUDING PARTITION EXCLUDING RELOPTIONS
                  );

   INSERT INTO tab2
   SELECT * FROM tab1
         MINUS SELECT * FROM tab2;
END
;
/
```

Example: Input: CREATE TABLE AS with DATA AND STATISTICS


```
CREATE SET VOLATILE TABLE tab2025
 AS ( SELECT * from tab2023 )
 WITH DATA AND STATISTICS
 PRIMARY INDEX (LOGTYPE, OPERSEQ);
```

Output:

```
CREATE LOCAL TEMPORARY TABLE tab2025
 DISTRIBUTE BY HASH ( LOGTYPE, OPERSEQ )
 AS ( SELECT * FROM tab2023 );

 ANALYZE tab2025;
```

6.4.5.2 CHARACTER SET and CASESPECIFIC

CHARACTER SET is used to specify the server character set for a character column. CASESPECIFIC specifies the case for character data comparisons and collations.

Use the **tdMigrateCharsetCase** configuration parameter to configure migration of CHARACTER SET and CASESPECIFIC. If **tdMigrateCharsetCase** is set to **false**, the tool will skip migration of the query and will log a message.

Input (tdMigrateCharsetCase=True)

```
CREATE MULTISET VOLATILE TABLE TAB1
(
  col1   INTEGER     NOT NULL
  ,col2     INTEGER     NOT NULL
  ,col3    VARCHAR(100)  NOT NULL CHARACTER SET UNICODE CASESPECIFIC )
PRIMARY INDEX (col1,col2)
ON COMMIT PRESERVE ROWS
;
```

Output:

```
CREATE LOCAL TEMPORARY TABLE TMP_RATING_SYS_PARA
(
  col1   INTEGER     NOT NULL
  ,col2     INTEGER     NOT NULL
  ,col3    VARCHAR(100)  NOT NULL /* CHARACTER SET UNICODE CASESPECIFIC  */)
)
ON COMMIT PRESERVE ROWS
 DISTRIBUTE BY HASH (col1,col2)

;
```

Input-Migration support for Character-based data type

In Teradata, the following character sets support character-based length for string data types:

- LATIN
- UNICODE
- GRAPHIC

  However, the KANJISJIS character set support byte-based length for string data types.

  For example, COLUMN_NAME VARCHAR(100) CHARACTER SET UNICODE CASESPECIFIC COLUMN_NAME VARCHAR(100) CHARACTER SET LATIN CASESPECIFIC This can store up to 100 characters (not bytes).

  In GaussDB(DWS), strings are byte-based (not character-based). VARCHAR (100) and VARCHAR2 (100) can store up to 100 byte (not characters). However, NVARCHAR2 (100) can store up to 100 characters.

So, if TD's LATIN, UNICODE and GRAPHIC character sets, VARCHAR should be migrated to NVARCHAR.

```
CREATE TABLE tab1
(
   col1 VARCHAR(10),
   COL2 CHAR(1)
);
```

Output:

```
a)when default_charset = UNICODE/GRAPHIC
CREATE
    TABLE
        tab1 (
            col1 NVARCHAR2 (10)
            ,COL2 NVARCHAR2 (1)
        ) ;

b)when default_charset = LATIN
CREATE
    TABLE
        tab1 (
            col1 VARCHAR2 (10)
            ,COL2 VARCHAR2 (1)
        ) ;
```

Input:

```
CREATE TABLE tab1
(
   col1 VARCHAR(10) CHARACTER SET UNICODE,
   COL2 CHAR(1)
);
```

Output:

```
a) when default_charset = UNICODE/GRAPHIC
CREATE
    TABLE
        tab1 (
            col1 NVARCHAR2 (10) /* CHARACTER SET UNICODE*/
            ,COL2 NVARCHAR2( 1 )
        ) ;

b) when default_charset = LATIN
CREATE
    TABLE
        tab1 (
            col1 NVARCHAR2 (10) /* CHARACTER SET UNICODE*/
            ,COL2 CHAR(1)
        ) ;
```

6.4.5.3 VOLATILE

The table-specific keyword **VOLATILE** is provided in the input file, but the keyword is not supported by GaussDB(DWS). The tool replaces it with the **LOCAL TEMPORARY** keyword during the migration process. Volatile tables are migrated as local temporary or unlogged based on the config input.

Input: CREATE VOLATILE TABLE

```
CREATE VOLATILE TABLE T1 (c1 int ,c2 int);
```

Output:

```
CREATE
   LOCAL TEMPORARY TABLE
```



```
   T1 (
       c1 INTEGER
      ,c2 INTEGER
     )
;
```

Input: CREATE VOLATILE TABLE AS WITH DATA (session_mode=Teradata)

If the source table has a PRIMARY KEY or a UNIQUE CONSTRAINT, then it will not contain any duplicate records. In this case, the MINUS operator is not required or added to remove duplicate records.

```
CREATE VOLATILE TABLE tabV1 (
       C1 INTEGER DEFAULT 99
      ,C2 INTEGER
      ,C3 INTEGER
      ,C4 NUMERIC (20,0) DEFAULT NULL (BIGINT)
      ,CONSTRAINT XX1 PRIMARY KEY ( C1, C2 )
      ) PRIMARY INDEX (C1, C3 );

CREATE TABLE tabV2 AS tabV1 WITH DATA PRIMARY INDEX (C1)
           ON COMMIT PRESERVE ROWS;
```

Output:

```
CREATE LOCAL TEMPORARY TABLE tabV1 (
       C1 INTEGER DEFAULT 99
      ,C2 INTEGER
      ,C3 INTEGER
      ,C4 NUMERIC (20,0) DEFAULT CAST( NULL AS BIGINT )
      ,CONSTRAINT XX1 PRIMARY KEY ( C1, C2 )
      ) DISTRIBUTE BY HASH (C1);

BEGIN
   CREATE TABLE tabV2 (
            LIKE tabV1 INCLUDING ALL EXCLUDING PARTITION EXCLUDING RELOPTIONS EXCLUDING
DISTRIBUTION
                  ) DISTRIBUTE BY HASH (C1);
   INSERT INTO tabV2 SELECT * FROM tabV1;
END
;
/
```

6.4.5.4 SET

SET is a unique feature in Teradata. It does not allow duplicate records. It is addressed using the **MINUS** set operator. Migration tool supports MULTISET and SET tables. SET table can be used with VOLATILE.

Input: SET TABLE

```
CREATE SET VOLATILE TABLE tab1 ... ;
INSERT INTO tab1
SELECT expr1, expr2, ...
FROM tab1, ...
WHERE .... ;
```

Output:

```
CREATE LOCAL TEMPORARY TABLE tab1
... ; INSERT INTO tab1
SELECT expr1, expr2, ...
FROM tab1, ...
WHERE ....
MINUS
SELECT * FROM tab1 ;
```


6.4.5.5 MULTISET

MULTISET is a normal table, which is supported by all the DBs. Migration tool supports MULTISET and SET tables.

MULTISET table can be used with VOLATILE.

Input: CREATE MULTISET TABLE

```
CREATE VOLATILE MULTISET TABLE T1 (c1 int ,c2 int);
```

Output:

```
CREATE
   LOCAL TEMPORARY TABLE
   T1 (
       c1 INTEGER
      ,c2 INTEGER
     )
;
```

6.4.5.6 TITLE

The keyword **TITLE** is supported for Teradata Permanent, Global Temporary and Volatile tables. In the migration process, the TITLE text is migrated as a comment.

NOTE

If the TITLE text is split across multiple lines, then in the migrated script, the line breaks (ENTER) are replaced with a space.

Input: CREATE TABLE with TITLE

```
CREATE TABLE tab1 (
  c1  NUMBER(2) TITLE 'column_a'
);
```

Output:

```
CREATE TABLE tab1 (
  c1  NUMBER(2) /* TITLE 'column_a' */
);
```

Input: TABLE with multiline TITLE

```
CREATE TABLE tab1 (
  c1  NUMBER(2) TITLE 'This is a
very long title'
);
```

Output:

```
CREATE TABLE tab1 (
  c1  NUMBER(2) /* TITLE 'This is a  very long title' */
);
```

Input: TABLE with COLUMN TITLE

DSC migrates COLUMN TITLE as a new outer query.

```
SELECT customer_id (TITLE 'cust_id')
FROM Customer_T
WHERE cust_id > 10;
```

Output:


```
SELECT
        customer_id  AS "cust_id"
    FROM
        (
            SELECT
                    customer_id
                FROM
                    Customer_T
                WHERE
                    cust_id > 10
        )
;
```

Input: TABLE with COLUMN TITLE and QUALIFY

```
SELECT ord_id
(TITLE 'Order_Id'), order_date, customer_id
  FROM order_t
WHERE Order_Id > 100
QUALIFY ROW_NUMBER() OVER(PARTITION BY customer_id ORDER BY order_date DESC) <= 5;
```

Output:

```
SELECT
        "mig_tmp_alias1" AS "Order_Id"
    FROM
        (
            SELECT
                    ord_id AS "mig_tmp_alias1"
                    ,ROW_NUMBER( ) OVER( PARTITION BY customer_id ORDER BY order_date DESC ) AS
ROW_NUM1
                FROM
                    order_t
                WHERE
                    Order_Id > 100
        ) Q1
    WHERE
        Q1.ROW_NUM1 <= 5
;
```

TITLE with ALIAS

If the TITLE is accompanied with an ALIAS, the tool will migrate it as follows:

- **TITLE with AS**: Tool will migrate it with the AS alias.
- **TITLE with NAMED:** Tool will migrate it with NAMED alias.
- **TITLE with NAMED and AS**: Tool will migrate it with AS alias.

Input: TABLE TITLE with NAMED and AS

```
SELECT  Acct_ID (TITLE 'Acc Code') (NAMED XYZ)  AS "Account Code"
       ,Acct_Name (TITLE 'Acc Name')
FROM    GT_JCB_01030_Acct_PBU
where "Account Code" > 500  group by "Account Code" ,Acct_Name ;
```

Output:

```
SELECT
        Acct_ID AS "Account Code"
        ,Acct_Name AS "Acc Name"
    FROM
        GT_JCB_01030_Acct_PBU
    WHERE
        Acct_ID > 500
    GROUP BY
        Acct_ID ,Acct_Name
;
```


NOTE

Currently the Migration tool supports the migration of the TITLE command included in the initial CREATE/ALTER statement. The subsequent references of the TITLE specified column are not supported. For example, in the CREATE TABLE statement below, the column ***eid*** with the TITLE Employee ID will be migrated to a comment but the reference of ***eid*** in the SELECT statement will be retained as it is.

Input

```
CREATE TABLE tab1 ( eid INT TITLE 'Employee ID');
SELECT eid FROM tab1;
```

Output

```
CREATE TABLE tab1 (eid INT /*TITLE 'Employee ID'*/);
SELECT eid from tab1;
```

TITLE with CREATE VIEW

Input:

```
REPLACE VIEW ${STG_VIEW}.B971_AUMSUMMARY${TABLE_SUFFIX_INC}
AS
LOCK TABLE ${STG_DATA}.B971_AUMSUMMARY${TABLE_SUFFIX_INC} FOR ACCESS
SELECT   AUM_DATE (TITLE '    ')
     ,CLNTCODE (TITLE '    ')
     ,ACCTYPE (TITLE '    ')
     ,CCY (TITLE '  ')
     ,BAL_AMT (TITLE '  ')
     ,MON_BAL_AMT (TITLE '    ')
     ,HK_CLNTCODE (TITLE '   ')
     ,MNT_DATE (TITLE '    ')
FROM ${STG_DATA}.B971_AUMSUMMARY${TABLE_SUFFIX_INC};
it should be migrated as below:
CREATE OR REPLACE VIEW ${STG_VIEW}.B971_AUMSUMMARY${TABLE_SUFFIX_INC}
AS
/*LOCK TABLE ${STG_DATA}.B971_AUMSUMMARY${TABLE_SUFFIX_INC} FOR ACCESS */
SELECT   AUM_DATE  /* (TITLE '    ') */
     ,CLNTCODE  /* (TITLE '    ') */
     ,ACCTYPE  /* (TITLE '    ') */
     ,CCY  /* (TITLE '  ') */
     ,BAL_AMT  /* (TITLE '  ') */
     ,MON_BAL_AMT  /* (TITLE '    ') */
     ,HK_CLNTCODE  /* (TITLE '   ') */
     ,MNT_DATE  /* (TITLE '    ') */
FROM ${STG_DATA}.B971_AUMSUMMARY${TABLE_SUFFIX_INC};
```

Output:

```
CREATE OR REPLACE VIEW ${STG_VIEW}.B971_AUMSUMMARY${TABLE_SUFFIX_INC}
AS
/*LOCK TABLE ${STG_DATA}.B971_AUMSUMMARY${TABLE_SUFFIX_INC} FOR ACCESS */
SELECT   AUM_DATE  /* (TITLE '    ') */
     ,CLNTCODE  /* (TITLE '    ') */
     ,ACCTYPE  /* (TITLE '    ') */
     ,CCY  /* (TITLE '  ') */
     ,BAL_AMT  /* (TITLE '  ') */
     ,MON_BAL_AMT  /* (TITLE '    ') */
     ,HK_CLNTCODE  /* (TITLE '   ') */
     ,MNT_DATE  /* (TITLE '    ') */
FROM ${STG_DATA}.B971_AUMSUMMARY${TABLE_SUFFIX_INC};
```

6.4.5.7 Indexes

The CREATE TABLE statement supports creation of an index. Migration tool supports the TABLE statement with PRIMARY INDEX and UNIQUE INDEX.

The tool will not add DISTRIBUTE BY HASH which is used to create a table with PRIMARY KEY and Non-Unique PRIMARY INDEX.

Input: CREATE TABLE with INDEX

```
CREATE SET TABLE DP_TEDW.B0381_ACCOUNT_OBTAINED_MAP,
     NO FALLBACK, NO BEFORE JOURNAL,
     NO AFTER JOURNAL, CHECKSUM = DEFAULT
 ( Ranked_Id            INTEGER  NOT NULL
 , Source_System_Code   SMALLINT NOT NULL
 , Operational_Acc_Obtained_Id VARCHAR(100)
   CHARACTER SET LATIN NOT CASESPECIFIC FORMAT 'X(50)'
 , Mapped_Id              INTEGER  NOT NULL
 )
PRIMARY INDEX B0381_ACCOUNT_OBTAINED_idx_PR ( Ranked_Id )
UNIQUE INDEX B0381_ACCT_OBT_MAP__idx_SCD ( Source_System_Code )
INDEX B0381_ACCT_OBT_MAP__idx_OPID ( Operational_Acc_Obtained_Id );
```

Output:

```
CREATE TABLE DP_TEDW.B0381_ACCOUNT_OBTAINED_MAP
  ( Ranked_Id INTEGER NOT NULL
  , Source_System_Code SMALLINT NOT NULL
  , Operational_Acc_Obtained_Id VARCHAR( 100 )
  , Mapped_Id INTEGER NOT NULL
  )
DISTRIBUTE BY HASH ( Ranked_Id );

CREATE INDEX B0381_ACCT_OBT_MAP__idx_SCD ON DP_TEDW.B0381_ACCOUNT_OBTAINED_MAP
( Source_System_Code );
CREATE INDEX B0381_ACCT_OBT_MAP__idx_OPID ON DP_TEDW.B0381_ACCOUNT_OBTAINED_MAP
( Operational_Acc_Obtained_Id );
```

NOTE

UNIQUE is removed in the index since index column list (organic_name) is not a super set of DISTRIBUTE BY column list (serial_no, organic_name).

Input - CREATE TABLE with Primary Key and Non-Unique Primary Index (DISTRIBUTE BY HASH is not added)

```
CREATE TABLE employee
 (
   EMP_NO INTEGER
 , DEPT_NO INTEGER
 , FIRST_NAME VARCHAR(20)
 , LAST_NAME CHAR(20)
 , SALARY DECIMAL(10,2)
 , ADDRESS VARCHAR(100)
 , CONSTRAINT pk_emp PRIMARY KEY ( EMP_NO )
   ) PRIMARY INDEX ( DEPT_NO ) ;
```

Output:

```
CREATE TABLE employee
 (
   EMP_NO INTEGER
 , DEPT_NO INTEGER
 , FIRST_NAME VARCHAR(20)
 , LAST_NAME CHAR(20)
 , SALARY DECIMAL(10,2)
 , ADDRESS VARCHAR(100)
 , CONSTRAINT pk_emp PRIMARY KEY ( EMP_NO )
  )
;
```


Creating a Partition Table with an Index

If **tdMigrateRANGE_N** is set to **true**,

Input:

```
CREATE SET TABLE SC.TAB , NO FALLBACK,
NO BEFORE JOURNAL,
NO AFTER JOURNAL,
CHECKSUM=DEFAULT,
DEFAULT MERGEBLOCKRATIO
(
ACCOUNT_NUM VARCHAR(255) CHARACTER SET LATIN NOT CASESPECIFIC NOT NULL
,ACCOUNT_MODIFIER_NUM CHAR(18) CHARACTER SET LATIN NOT CASESPECIFIC NOT NULL
,END_DT DATE FORMAT 'YYYY-MM-DD'
,UPD_TXF_BATCHTD INTEGER COMPRESS
)
PRIMARY INDEX XPKT0300_AGREEMENT (ACCOUNT_NUM,ACCOUNT_MODIFIER_NUM)
PARTITION BY RANGE_N(END_DT BETWEEN '2001-01-01' AND '2020-12-31' EACH INTERVAL '1' DAY, NO
RANGE ,UNKNOWN)
INDEX (UPD_TXF_BATCHTD)
;
```

Output:

```
CREATE
    TABLE
        SC.TAB (
            ACCOUNT_NUM VARCHAR( 255 ) /* CHARACTER SET LATIN*/
             /* NOT CASESPECIFIC*/              NOT NULL
            ,ACCOUNT_MODIFIER_NUM CHAR( 18 ) /* CHARACTER SET LATIN*/              /* NOT
CASESPECIFIC*/              NOT NULL
            ,END_DT DATE
            ,UPD_TXF_BATCHTD INTEGER /* COMPRESS */
        ) DISTRIBUTE BY HASH (
            ACCOUNT_NUM
            ,ACCOUNT_MODIFIER_NUM
        ) PARTITION BY RANGE (END_DT) (
            PARTITION TAB_1 start ('2001-01-01')
        END ('2020-12-31') EVERY (
            INTERVAL '1' DAY )
        ) ;
CREATE INDEX ON SC.TAB (UPD_TXF_BATCHTD) LOCAL;
```

6.4.5.8 CONSTRAINT

A table CONSTRAINT is applied to multiple columns. Migration tool supports the following constraints:

- REFERENCES constraint / FOREIGN KEY: migration currently NOT supported by tool.
- PRIMARY KEY constraint: migration supported by tool.
- UNIQUE constraint: migration supported by tool.

Input: CREATE TABLE with CONSTRAINT

```
CREATE SET TABLE DP_SEDW.T_170UT_HOLDER_ACCT, NO FALLBACK,
   NO BEFORE JOURNAL, NO AFTER JOURNAL
 ( BUSINESSDATE   VARCHAR(10)
 , SOURCESYSTEM   VARCHAR(5)
 , UPLOADCODE     VARCHAR(1)
 , HOLDER_NO      VARCHAR(7)  NOT NULL
 , POSTAL_ADD_4   VARCHAR(40)
 , EPF_IND        CHAR(1)
 , CONSTRAINT uq_t_170ut_hldr UNIQUE ( SOURCESYSTEM, UPLOADCODE,
```



```
HOLDER_NO )
    ) PRIMARY INDEX ( HOLDER_NO, SOURCESYSTEM ) ;
```

Output:

```
CREATE TABLE DP_SEDW.T_170UT_HOLDER_ACCT
  ( BUSINESSDATE      VARCHAR( 10 )
  , SOURCESYSTEM      VARCHAR( 5 )
  , UPLOADCODE        VARCHAR( 1 )
  , HOLDER_NO         VARCHAR( 7 )   NOT NULL
  , POSTAL_ADD_4      VARCHAR( 40 )
  , EPF_IND           CHAR( 1 )
  , CONSTRAINT uq_t_170ut_hldr UNIQUE ( SOURCESYSTEM, UPLOADCODE, HOLDER_NO )
       )
DISTRIBUTE BY HASH ( HOLDER_NO, SOURCESYSTEM );
```

Input:

After table creation, CONSTRAINT can be added to a table column to put some restriction at column level by using ALTER statement.

```
CREATE TABLE GCC_PLAN.T1033 ( ROLLOUT_PLAN_LINE_ID NUMBER NOT NULL,
                                        UDF_FIELD_VALUE_ID NUMBER NOT NULL) ;
ALTER TABLE GCC_PLAN.T1033
ADD CONSTRAINT UDF_FIELD_VALUE_ID_PK UNIQUE (UDF_FIELD_VALUE_ID) ;
```

Output:

```
CREATE TABLE GCC_PLAN.T1033 ( ROLLOUT_PLAN_LINE_ID NUMBER NOT NULL,
                                        UDF_FIELD_VALUE_ID NUMBER NOT NULL,
                                        CONSTRAINT UDF_FIELD_VALUE_ID_PK
                                        UNIQUE (UDF_FIELD_VALUE_ID) ;
```

NOTE

Need to put CONSTRAINT creation syntax inside table creation script after all column declaration.

6.4.5.9 COLUMN STORE

The table orientation can be converted from ROW-STORE to COLUMN store using the WITH (ORIENTATION=COLUMN) in the CREATE TABLE statement. This feature can be enabled/disabled using the **rowstoreToColumnstore** configuration parameter.

Input: CREATE TABLE with change orientation to COLUMN STORE

```
CREATE MULTISET VOLATILE TABLE tab1
    ( c1 VARCHAR(30) CHARACTER SET UNICODE
    , c2 DATE
    , ...
    )
 PRIMARY INDEX (c1, c2)
 ON COMMIT PRESERVE ROWS;
```

Output:

```
CREATE LOCAL TEMPORARY TABLE tab1
    ( c1 VARCHAR(30)
    , c2 DATE
    , ...
    ) WITH (ORIENTATION = COLUMN)
  ON COMMIT PRESERVE ROWS
  DISTRIBUTE BY HASH (c1, c2);
```


6.4.5.10 PARTITION

The tool does not support migration of partitions/subpartitions and the partition/subpartition keywords are commented in the migrated scripts:

- Range partition/subpartition
- List partition/subpartition
- Hash partition/subpartition

Scenario 1: Assume that the configuration parameters (**tdMigrateCASE_N** and **tdMigrateRANGE_N**) are set to **comment** and **range** respectively.

The following is a Teradata CREATE TABLE script with nested partitions.

Input - PARTITION BY RANGE_N

```
CREATE TABLE tab1
    ( entry_id          integer   not null
        , oper_id           integer   not null
        , source_system_cd   varchar(5)
        , entry_dt          date
        , file_id           integer
         , load_id           integer
        , contract_id        varchar(50)
        , contract_type_cd   varchar(50)
        )
PRIMARY INDEX (entry_id, oper_id, source_system_cd)
PARTITION BY ( CASE_N( source_system_cd = '00000'
                                        , source_system_cd = '00002'
                                        , source_system_cd = '00006'
                                        , source_system_cd = '00018'
                                        , NO CASE )
                       , RANGE_N(  entry_dt BETWEEN DATE '2012-01-01' AND DATE '2025-12-31' EACH
INTERVAL '1' DAY, NO RANGE )
                       );
```

Output:

```
CREATE TABLE tab1
    ( entry_id          integer   not null
        , oper_id           integer   not null
        , source_system_cd   varchar(5)
        , entry_dt          date
        , file_id           integer
         , load_id           integer
        , contract_id        varchar(50)
        , contract_type_cd   varchar(50)
        )
DISTRIBUTE BY HASH (entry_id, oper_id, source_system_cd)
PARTITION BY RANGE (entry_dt) ( PARTITION tab1_p1 START (CAST('2012-01-01' AS DATE))
                                                  END (CAST('2025-12-31' AS DATE))
                                                  EVERY (INTERVAL '1' DAY) );
```

Scenario 2: Assume that the configuration parameters (**tdMigrateCASE_N** and **tdMigrateRANGE_N**) are set to **comment** or **range** respectively.

The following is a Teradata CREATE TABLE script with nested partitions.

Input:

```
CREATE TABLE tab2
    ( entry_id          integer   not null
        , oper_id           integer   not null
        , source_system_cd   varchar(5)
        , entry_dt          date
        , file_id           integer
```



```
        , load_id           integer
        , contract_id       varchar(50)
        , contract_type_cd  varchar(50)
        )
PRIMARY INDEX (entry_id, oper_id, source_system_cd)
PARTITION BY ( RANGE_N(  entry_dt BETWEEN DATE '2012-01-01' AND DATE '2025-12-31' EACH
INTERVAL '1' DAY, NO RANGE )
                        , CASE_N( source_system_cd = '00000'
                                , source_system_cd = '00002'
                                , source_system_cd = '00006'
                                , source_system_cd = '00018'
                                , NO CASE )
                        );
```

Output:

```
CREATE TABLE tab2
    ( entry_id          integer   not null
        , oper_id           integer   not null
        , source_system_cd  varchar(5)
        , entry_dt          date
        , file_id           integer
        , load_id           integer
        , contract_id       varchar(50)
        , contract_type_cd  varchar(50)
        )
DISTRIBUTE BY HASH (entry_id, oper_id, source_system_cd)
PARTITION BY RANGE (entry_dt) ( PARTITION tab2_p1 START (CAST('2012-01-01' AS DATE))
                                                  END (CAST('2025-12-31' AS DATE))
                                                  EVERY (INTERVAL '1' DAY) );
```

Scenario 3: Assume that the configuration parameters (**tdMigrateCASE_N** and **tdMigrateRANGE_N**) are set to values other than **comment** or **range** respectively.

Partition syntax will not be commented and the remaining syntax will be migrated.

Input:

```
CREATE TABLE tab1
    ( entry_id          integer   not null
        , oper_id           integer   not null
        , source_system_cd  varchar(5)
        , entry_dt          date
        , file_id           integer
        , load_id           integer
        , contract_id       varchar(50)
        , contract_type_cd  varchar(50)
        )
PRIMARY INDEX (entry_id, oper_id, source_system_cd)
PARTITION BY ( CASE_N( source_system_cd = '00000'
                                , source_system_cd = '00002'
                                , source_system_cd = '00006'
                                , source_system_cd = '00018'
                                , NO CASE )
                        , RANGE_N(  entry_dt BETWEEN DATE '2012-01-01' AND DATE '2025-12-31' EACH
INTERVAL '1' DAY, NO RANGE )
                        );
```

Output:

```
CREATE TABLE tab2
    ( entry_id          integer   not null
        , oper_id           integer   not null
        , source_system_cd  varchar(5)
        , entry_dt          date
        , file_id           integer
        , load_id           integer
        , contract_id       varchar(50)
```



```
        , contract_type_cd   varchar(50)
        )
DISTRIBUTE BY HASH (entry_id, oper_id, source_system_cd)
/* PARTITION BY ( CASE_N( source_system_cd = '00000'
                                        , source_system_cd = '00002'
                                        , source_system_cd = '00006'
                                        , source_system_cd = '00018'
                                        , NO CASE )
                          , RANGE_N(  entry_dt BETWEEN DATE '2012-01-01' AND DATE '2025-12-31' EACH
INTERVAL '1' DAY, NO RANGE )
                          ) */
;
```

Scenario 4: Assume that the configuration parameters (**tdMigrateCASE_N** and **tdMigrateRANGE_N**) are set to any value.

The following is another TD create table script with RANGE_N partition (without nested partitions).

Input:

```
CREATE TABLE tab4
    ( entry_id          integer   not null
        , oper_id           integer   not null
        , source_system_cd   varchar(5)
        , entry_dt          date
        , file_id           integer
         , load_id           integer
        , contract_id        varchar(50)
        , contract_type_cd   varchar(50)
        )
PRIMARY INDEX (entry_id, oper_id, source_system_cd)
PARTITION BY (RANGE_N(  entry_dt BETWEEN DATE '2012-01-01' AND DATE '2025-12-31' EACH INTERVAL
'1' DAY, NO RANGE )
                    );
```

Output:

```
CREATE TABLE tab4
    ( entry_id          integer   not null
        , oper_id           integer   not null
        , source_system_cd   varchar(5)
        , entry_dt          date
        , file_id           integer
         , load_id           integer
        , contract_id        varchar(50)
        , contract_type_cd   varchar(50)
        )
DISTRIBUTE BY HASH (entry_id, oper_id, source_system_cd)
PARTITION BY RANGE (entry_dt) ( PARTITION tab4_p1 START (CAST('2012-01-01' AS DATE))
                                                  END (CAST('2025-12-31' AS DATE))
                                                  EVERY (INTERVAL '1' DAY) );
```

Scenario 5: Assume that the configuration parameters (**tdMigrateCASE_N** and **tdMigrateRANGE_N**) are set to **comment** or **range** respectively.

The following is another Teradata script for creating a table with a CASE_N partition (without nested partitions).

Input:

```
CREATE TABLE tab5
    ( entry_id          integer   not null
        , oper_id           integer   not null
        , source_system_cd   varchar(5)
        , entry_dt          date
        , file_id           integer
         , load_id           integer
```



```
        , contract_id        varchar(50)
        , contract_type_cd   varchar(50)
        )
PRIMARY INDEX (entry_id, oper_id, source_system_cd)
PARTITION BY ( CASE_N( source_system_cd = '00000'
                                        , source_system_cd = '00002'
                                        , source_system_cd = '00006'
                                        , source_system_cd = '00018'
                                        , NO CASE )
                );
```

Output:

```
CREATE TABLE tab5
    ( entry_id           integer   not null
        , oper_id            integer   not null
        , source_system_cd   varchar(5)
        , entry_dt           date
        , file_id            integer
         , load_id           integer
        , contract_id        varchar(50)
        , contract_type_cd   varchar(50)
        )
DISTRIBUTE BY HASH (entry_id, oper_id, source_system_cd)
/* PARTITION BY ( CASE_N( source_system_cd = '00000'
                                        , source_system_cd = '00002'
                                        , source_system_cd = '00006'
                                        , source_system_cd = '00018'
                                        , NO CASE )
                ) */
;
```

Partition of RANGE_N in the String Columns

Input:

```
CREATE SET TABLE SC.TAB , NO FALLBACK,
NO BEFORE JOURNAL,
NO AFTER JOURNAL,
CHECKSUM=DEFAULT,
DEFAULT MERGEBLOCKRATIO
(
ACCOUNT_NUM VARCHAR(255) CHARACTER SET LATIN NOT CASESPECIFIC NOT NULL
,ACCOUNT_MODIFIER_NUM CHAR(18) CHARACTER SET LATIN NOT CASESPECIFIC NOT NULL
,DATA_SOURCE_ID CHAR(10) CHARACTER SET LATIN NOT CASESPECIFIC
,END_DT DATE FORMAT 'YYYY-MM-DD'
,UPD_TXF_BATCHTD INTEGER COMPRESS
)
PRIMARY INDEX XPKT0300_AGREEMENT (ACCOUNT_NUM,ACCOUNT_MODIFIER_NUM)
PARTITION BY ( RANGE_N(DATA_SOURCE_ID BETWEEN
'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z' AND 'ZZ', NO
RANGE ,UNKNOWN) ,CASE_N(END_DT IS NULL , NO CASE , UNKNOWN))
;
```

Output:

```
CREATE
    TABLE
        SC.TAB (
            ACCOUNT_NUM VARCHAR( 255 ) /* CHARACTER SET LATIN*/       /* NOT
CASESPECIFIC*/               NOT NULL
            ,ACCOUNT_MODIFIER_NUM CHAR( 18 ) /* CHARACTER SET LATIN*/              /* NOT
CASESPECIFIC*/               NOT NULL
            ,DATA_SOURCE_ID CHAR( 10 ) /* CHARACTER SET LATIN*/              /* NOT CASESPECIFIC*/
            ,END_DT DATE
            ,UPD_TXF_BATCHTD INTEGER /* COMPRESS */
        ) DISTRIBUTE BY HASH (
            ACCOUNT_NUM
            ,ACCOUNT_MODIFIER_NUM
```



```
        )/* PARTITION BY (
            RANGE_N (
                DATA_SOURCE_ID BETWEEN 'A'
                ,'B'
                ,'C'
                ,'D'
                ,'E'
                ,'F'
                ,'G'
                ,'H'
                ,'I'
                ,'J'
                ,'K'
                ,'L'
                ,'M'
                ,'N'
                ,'O'
                ,'P'
                ,'Q'
                ,'R'
                ,'S'
                ,'T'
                ,'U'
                ,'V'
                ,'W'
                ,'X'
                ,'Y'
                ,'Z' AND 'ZZ'
                ,NO RANGE
                ,UNKNOWN
            )
            ,*/
/* CASE_N(END_DT IS NULL , NO CASE , UNKNOWN))  */
            ;
```

RANGE_N with different partition INTERVAL

Input:

```
CREATE MULTISET TABLE tab1
    ( TICD                VARCHAR(10)
    , TCIT                VARCHAR(10)
    , TCCM                 VARCHAR(50)
    , DW_Stat_Dt          DATE
   )
PRIMARY INDEX ( TICD )
PARTITION BY RANGE_N
  ( DW_Stat_Dt BETWEEN DATE '0001-01-01' AND DATE '0001-01-04' EACH INTERVAL '1' DAY,
    DATE '0001-01-05' AND DATE '1899-12-31',
    DATE '1900-01-01' AND DATE '1900-01-01',
    DATE '1900-01-02' AND DATE '1999-12-31',
    DATE '2000-01-01' AND DATE '2009-12-31' EACH INTERVAL '1' YEAR,
    DATE '2010-01-01' AND DATE '2021-12-31' EACH INTERVAL '1' DAY,
    DATE '9999-12-31' AND DATE '9999-12-31',
    NO RANGE );
```

Output:

```
CREATE TABLE tab1
    ( TICD                VARCHAR( 10 )
    , TCIT                VARCHAR( 10 )
    , TCCM                 VARCHAR( 50 )
    , DW_Stat_Dt          DATE
    )
DISTRIBUTE BY HASH (TICD)
PARTITION BY RANGE (DW_Stat_Dt)
  ( PARTITION tab1_0 START (DATE '0001-01-01') END (DATE '0001-01-04') EVERY (INTERVAL '1' DAY),
    PARTITION tab1_1 START (DATE '0001-01-04') END (DATE '1899-12-31'),
    PARTITION tab1_2 START (DATE '1899-12-31') END (DATE '1900-01-01'),
```



```
    PARTITION tab1_ 3 START (DATE '1900-01-01') END (DATE '1999-12-31'),
    PARTITION tab1_4 START (DATE '1999-12-31') END (DATE '2009-12-31') EVERY (INTERVAL '1' YEAR) ,
    PARTITION tab1_5 START (DATE '2009-12-31') END (DATE '2021-12-31') EVERY (INTERVAL '1' DAY) ,
    PARTITION tab1_6 START (DATE '2021-12-31') END (DATE '9999-12-31')
 );
```

RANGE_N with * for start-date

Input:

```
CREATE MULTISET TABLE Orders5 (
  StoreNo SMALLINT,
  OrderNo INTEGER,
  OrderDate DATE,
  OrderTotal INTEGER
)
PRIMARY INDEX(OrderNo)
PARTITION BY RANGE_N  (
  OrderDate BETWEEN DATE * AND DATE '2016-12-31' EACH INTERVAL '1' YEAR,
    DATE '2017-01-01'  EACH INTERVAL '1' MONTH,
    DATE '2020-01-01' AND DATE '2020-12-31' EACH INTERVAL '1' DAY
);
```

Output:

```
CREATE TABLE Orders5 (
  StoreNo SMALLINT,
  OrderNo INTEGER,
  OrderDate DATE,
  OrderTotal INTEGER
)
DISTRIBUTE BY HASH (OrderNo)
PARTITION BY RANGE (OrderDate)
  ( PARTITION Orders5_0 START (DATE '0001-01-01') END (DATE '2016-12-31') EVERY (INTERVAL '1' YEAR),
PARTITION Orders5_1 START (DATE '2016-12-31') END (DATE '2020-01-01') EVERY (INTERVAL '1' MONTH),
PARTITION Orders5_2 START (DATE '2020-01-01') END (DATE '2020-12-31') EVERY (INTERVAL '1' DAY)
);
```

RANGE_N with * for end-date

Input:

```
CREATE SET TABLE Orders4 (
  StoreNo SMALLINT,
  OrderNo INTEGER,
  OrderDate DATE,
  OrderTotal INTEGER
)
PRIMARY INDEX(OrderNo)
PARTITION BY RANGE_N  (
  OrderDate BETWEEN DATE '2010-01-01' AND '2016-12-31' EACH INTERVAL '1' YEAR,
    DATE '2017-01-01' EACH INTERVAL '1' MONTH,
    DATE '2019-01-01' AND *
);
```

Output:

```
CREATE TABLE Orders4 (
  StoreNo SMALLINT,
  OrderNo INTEGER,
  OrderDate DATE,
  OrderTotal INTEGER
)
DISTRIBUTE BY HASH (OrderNo)
PARTITION BY RANGE (OrderDate)
  ( PARTITION Orders4_0 START (DATE '2010-01-01') END (DATE '2016-12-31') EVERY (INTERVAL '1' YEAR),
PARTITION Orders4_1 START (DATE '2016-12-31') END (DATE '2020-01-01') EVERY (INTERVAL '1' MONTH) ,
```



```
PARTITION Orders4_2 START (DATE '2020-01-01') END (MAXVALUE)
);
```

RANGE_N with comma separated values

Input:

```
CREATE TABLE orders10
    (storeid INTEGER NOT NULL
    ,productid INTEGER NOT NULL
    ,orderdate DATE NOT NULL
    ,totalorders INTEGER NOT NULL)
    PRIMARY INDEX (storeid, productid)
     PARTITION BY ( RANGE_N(totalorders BETWEEN *, 100, 1000 AND *) );
```

Output:

```
CREATE TABLE orders10
    (storeid INTEGER NOT NULL
    ,productid INTEGER NOT NULL
    ,orderdate DATE NOT NULL
    ,totalorders INTEGER NOT NULL)
DISTRIBUTE BY HASH (storeid, productid)
PARTITION BY RANGE (totalorders)
   ( PARTITION Orders10_0 END (100),
     PARTITION Orders10_1 END (1000),
     PARTITION Orders10_2 END (MAXVALUE)
   );
```

6.4.5.11 ANALYZE

The **ANALYZE** statement of Teradata is used to migrate tables.

Input: CREATE TABLE with INDEX

```
CREATE TABLE EMP27 AS emp21 WITH DATA
PRIMARY INDEX (EMPNO) ON COMMIT PRESERVE ROWS;
```

Output:

```
Begin
CREATE TABLE EMP27
( LIKE emp21 INCLUDING ALL EXCLUDING PARTITION EXCLUDING RELOPTIONS EXCLUDING
DISTRIBUTION )
DISTRIBUTE BY HASH ( EMPNO ) ;
INSERT INTO EMP27
select * from emp21 ;
end ;
/
ANALYZE Emp27 (EmpNo);
```

6.4.5.12 Support for Specified Columns

Migration tool supports queries that specify number of columns (not all columns specified) during INSERT. This can happen when the input INSERT statement does not contain all the columns mentioned in the input CREATE statement. During migration, the columns are added with any default values specified.

NOTE

- This feature is supported if **session_mode** is **Teradata**.
- The SELECT statement for the INSERT-INTO-SELECT must not include the following:
  - Set operators
  - MERGE, TOP with PERCENT, TOP PERCENT with TIES

Input - TABLE with all columns of CREATE are not specified in the INSERT statement

```
CREATE
    VOLATILE TABLE
        Convert_Data3
        ,NO LOG (
            zoneno CHAR( 6 )
            ,brno CHAR( 6 )
            ,currtype CHAR( 4 )
            ,Commuteno CHAR( 4 )
            ,Subcode CHAR( 12 )
            ,accdate DATE format 'YYYY-MM-DD' NOT NULL
            ,acctime INTEGER
            ,quoteno CHAR( 1 )
            ,quotedate DATE FORMAT 'YYYY-MM-DD'
            ,lddrbaL DECIMAL( 18 ,0 ) DEFAULT 0
            ,ldcrbal DECIMAL( 18 ,0 )
            ,tddramt DECIMAL( 18 ,0 ) DEFAULT 25
            ,tdcramt DECIMAL( 18 ,0 )
            ,tddrbal DECIMAL( 18 ,2 )
            ,tdcrbal DECIMAL( 18 ,2 )
        ) PRIMARY INDEX (
            BRNO
            ,CURRTYPE
            ,SUBCODE
        )
            ON COMMIT PRESERVE ROWS
;

INSERT
    INTO
        Convert_Data3 (
            zoneno
            ,brno
            ,currtype
            ,commuteno
            ,subcode
            ,accdate
            ,acctime
            ,quoteno
            ,quotedate
            ,tddrbal
            ,tdcrbal
        ) SELECT
                A.zoneno
                ,A.brno
                ,'014' currtype
                ,'2' commuteno
                ,A.subcode
                ,A.Accdate
                ,A.Acctime
                ,'2' quoteno
                ,B.workdate quoteDate
                ,CAST( ( CAST( SUM ( CAST( A.tddrbal AS FLOAT ) * CAST( B.USCVRATE AS FLOAT ) ) AS
FLOAT ) ) AS DEC ( 18 ,2 ) ) AS tddrbal
                ,CAST( ( CAST( SUM ( CAST( A.tdcrbal AS FLOAT ) * CAST( B.USCVRATE AS FLOAT ) ) AS
FLOAT ) ) AS DEC ( 18 ,2 ) ) AS tdcrbal
            FROM
                table2 A
;
```


Output:

```
CREATE
    LOCAL TEMPORARY TABLE
        Convert_Data3 (
            zoneno CHAR( 6 )
            ,brno CHAR( 6 )
            ,currtype CHAR( 4 )
            ,Commuteno CHAR( 4 )
            ,Subcode CHAR( 12 )
            ,accdate DATE NOT NULL
            ,acctime INTEGER
            ,quoteno CHAR( 1 )
            ,quotedate DATE
            ,lddrbaL DECIMAL( 18 ,0 ) DEFAULT 0
            ,ldcrbal DECIMAL( 18 ,0 )
            ,tddramt DECIMAL( 18 ,0 ) DEFAULT 25
            ,tdcramt DECIMAL( 18 ,0 )
            ,tddrbal DECIMAL( 18 ,2 )
            ,tdcrbal DECIMAL( 18 ,2 )
        )
            ON COMMIT PRESERVE ROWS DISTRIBUTE BY HASH (
            BRNO
            ,CURRTYPE
            ,SUBCODE
        )
;

INSERT
    INTO
        Convert_Data3 (
            lddrbaL
            ,ldcrbal
            ,tddramt
            ,tdcramt
            ,zoneno
            ,brno
            ,currtype
            ,commuteno
            ,subcode
            ,accdate
            ,acctime
            ,quoteno
            ,quotedate
            ,tddrbal
            ,tdcrbal
        ) SELECT
                0
                ,NULL
                ,25
                ,NULL
                ,A.zoneno
                ,A.brno
                ,'014' currtype
                ,'2' commuteno
                ,A.subcode
                ,A.Accdate
                ,A.Acctime
                ,'2' quoteno
                ,B.workdate quoteDate
                ,CAST( ( CAST( SUM ( CAST( A.tddrbal AS FLOAT ) * CAST( B.USCVRATE AS FLOAT ) ) AS
FLOAT ) ) AS DECIMAL( 18 ,2 ) ) AS tddrbal
                ,CAST( ( CAST( SUM ( CAST( A.tdcrbal AS FLOAT ) * CAST( B.USCVRATE AS FLOAT ) ) AS
FLOAT ) ) AS DECIMAL( 18 ,2 ) ) AS tdcrbal
            FROM
                table2 A MINUS SELECT
                        lddrbaL
                        ,ldcrbal
                        ,tddramt
                        ,tdcramt
```



```
                    ,zoneno
                    ,brno
                    ,currtype
                    ,commuteno
                    ,subcode
                    ,accdate
                    ,acctime
                    ,quoteno
                    ,quotedate
                    ,tddrbal
                    ,tdcrbal
                FROM
                    CONVERT_DATA3
;
```

6.4.6 Migrating Indexes

The sequence of **CREATE INDEX** columns and table names in Teradata is different from that in GaussDB(DWS). Use the configuration parameter **distributeByHash** to configure how the data is distributed across the cluster nodes. The tool will not add DISTRIBUTE BY HASH which is used to create a table with Primary Key and Non-Unique Primary Index.

Input - Primary key is not superset of primary index and only one column is matched

```
CREATE TABLE      good_5 (
             column_1 INTEGER NOT NULL PRIMARY KEY
            ,column_2 INTEGER
            ,column_3 INTEGER NOT NULL
            ,column_4 INTEGER
        ) PRIMARY INDEX (column _1,column_2);
```

Output:

```
CREATE TABLE      good_5 (
             column_1 INTEGER NOT NULL PRIMARY KEY
            ,column_2 INTEGER
            ,column_3 INTEGER NOT NULL
            ,column_4 INTEGER
        )
;
```

Input - Primary key is not superset of primary index and no column is matched

```
CREATE SET  TABLE  DP_SEDW.T_170UT_HOLDER_ACCT
         ,NO FALLBACK
         ,NO BEFORE JOURNAL
         ,NO AFTER JOURNAL (
            BUSINESSDATE VARCHAR( 10 )
            ,SOURCESYSTEM VARCHAR( 5 )
            ,UPLOADCODE VARCHAR( 1 )
            ,HOLDER_NO VARCHAR( 7 ) NOT NULL
            ,POSTAL_ADD_4 VARCHAR( 40 )
            ,EPF_IND CHAR( 1 )
            ,PRIMARY KEY (  UPLOADCODE  ,HOLDER_NO   )
) PRIMARY INDEX ( SOURCESYSTEM,EPF_IND  );
```

Output:

```
CREATE TABLE   DP_SEDW.T_170UT_HOLDER_ACCT (
            BUSINESSDATE VARCHAR( 10 )
            ,SOURCESYSTEM VARCHAR( 5 )
            ,UPLOADCODE VARCHAR( 1 )
            ,HOLDER_NO VARCHAR( 7 ) NOT NULL
            ,POSTAL_ADD_4 VARCHAR( 40 )
```



```
            ,EPF_IND CHAR( 1 )
            ,PRIMARY KEY (UPLOADCODE ,HOLDER_NO   ) );
```

Input - No primary key and unique index has index name

```
CREATE SET TABLE "DP_TEDW"."T0409_INTERNAL_ORG_GRP_FUNCT",
    NO FALLBACK, NO BEFORE JOURNAL,
    NO AFTER JOURNAL
    ( Organization_Party_Id       INTEGER              NOT NULL
    , Function_Code               SMALLINT             NOT NULL
    , Intern_Funct_Strt_Date      DATE FORMAT 'YYYY-MM-DD'  NOT NULL
    , Intern_Funct_End_Date       DATE FORMAT 'YYYY-MM-DD'
    )
PRIMARY INDEX ( Organization_Party_Id )
UNIQUE INDEX ux_t0409_intr_fn_1 ( Function_Code, Intern_Funct_Strt_Date )
UNIQUE INDEX ( Organization_Party_Id, Intern_Funct_Strt_Date );
```

Output:

```
CREATE TABLE "DP_TEDW"."T0409_INTERNAL_ORG_GRP_FUNCT"
        ( Organization_Party_Id       INTEGER           NOT NULL
        , Function_Code               SMALLINT          NOT NULL
        , Intern_Funct_Strt_Date      DATE              NOT NULL
        , Intern_Funct_End_Date       DATE
        )
DISTRIBUTE BY HASH ( Organization_Party_Id );
CREATE INDEX ux_t0409_intr_fn_1 ON "DP_TEDW"."T0409_INTERNAL_ORG_GRP_FUNCT" ( Function_Code,
Intern_Funct_Strt_Date );
CREATE UNIQUE INDEX ON "DP_TEDW"."T0409_INTERNAL_ORG_GRP_FUNCT" ( Organization_Party_Id,
Intern_Funct_Strt_Date );
```

Input - CREATE TABLE with Primary Key and Non-Unique Primary Index (DISTRIBUTE BY HASH is not added)

```
CREATE TABLE employee
 (
   EMP_NO INTEGER
 , DEPT_NO INTEGER
 , FIRST_NAME VARCHAR(20)
 , LAST_NAME CHAR(20)
 , SALARY DECIMAL(10,2)
 , ADDRESS VARCHAR(100)
 , CONSTRAINT pk_emp PRIMARY KEY ( EMP_NO )
   ) PRIMARY INDEX ( DEPT_NO ) ;
```

Output:

```
CREATE TABLE employee
 (
   EMP_NO INTEGER
 , DEPT_NO INTEGER
 , FIRST_NAME VARCHAR(20)
 , LAST_NAME CHAR(20)
 , SALARY DECIMAL(10,2)
 , ADDRESS VARCHAR(100)
 , CONSTRAINT pk_emp PRIMARY KEY ( EMP_NO )
  )
;
```

6.4.7 Migrating Views

CREATE VIEW (**short key** CV) is used together with **SELECT** to create a view.

The keyword **VIEW** is supported by both Teradata and GaussDB(DWS), but the **SELECT** statements are enclosed in double quotation marks during the migration. For details, see the following figures.

Use the **tdMigrateVIEWCHECKOPTIO....** configuration parameter to configure migration of views containing the WITH CHECK OPTION keyword. If

tdmigrateVIEWCHECKOPTION is set to **false**, the tool will skip migration of the query and will log a message.

If the CREATE VIEW includes the LOCK keyword, then the VIEW query will be migrated based on the value of **tdMigrateLOCKoption**.

Input - CREATE VIEW

```
CREATE VIEW DP_STEDW.MY_PARAM
AS
SELECT RUNDATE FROM  DP_STEDW.DATE_TBL   WHERE   dummy = 1;
```

Output:

```
CREATE OR REPLACE VIEW DP_STEDW.MY_PARAM
AS
SELECT RUNDATE
  FROM  DP_STEDW.DATE_TBL
        WHERE   dummy = 1;
```

Input: CREATE VIEW WITH FORCE KEYWORD

```
CREATE
OR REPLACE FORCE VIEW IS2010_APP_INFO (
  APP_ID, APP_SHORTNAME, APP_CHNAME,
  APP_ENNAME
) AS
select
  t.app_id,
  t.app_shortname,
  t.app_chname,
  t.app_enname
from
  newdrms.seas_app_info t
WHERE
  t.app_status <> '2';
```

Output:

```
CREATE
OR REPLACE
/*FORCE*/
VIEW IS2010_APP_INFO (
    APP_ID,
    APP_SHORTNAME,
    APP_CHNAME,
    APP_ENNAME ) AS
SELECT
    t.app_id,
    t.app_shortname,
    t.app_chname,
    t.app_enname
FROM
    newdrms.seas_app_info t
WHERE
t.app_status <> '2';
```

REPLACE VIEW

In Teradata, the **REPLACE VIEW** statement is used to create a view or rebuild the existing view. DSC converts the **REPLACE VIEW** statement to the **CREATE OR REPLACE VIEW** statement that is compatible with GaussDB(DWS).

Input - REPLACE VIEW

```
REPLACE VIEW DP_STEDW.MY_PARAM AS SELECT
        RUNDATE
```



```
    FROM
        DP_STEDW.DATE_TBL
    WHERE
        dummy = 1
;
```

Output:

```
CREATE
OR REPLACE VIEW DP_STEDW.MY_PARAM AS (
    SELECT
            RUNDATE
        FROM
            DP_STEDW.DATE_TBL
        WHERE
            dummy = 1
)
;
```

Input - REPLACE RECURSIVE VIEW

```
Replace RECURSIVE VIEW reachable_from (
emp_id,emp_name,DEPTH)
AS (
SELECT root.emp_id,root.emp_name,0 AS DEPTH
FROM emp AS root
WHERE root.mgr_id IS NULL);
```

Output:

```
CREATE OR REPLACE VIEW reachable_from AS (
WITH RECURSIVE reachable_from (
emp_id,emp_name,DEPTH)
AS (
SELECT root.emp_id,root.emp_name,0 AS DEPTH
FROM emp AS root
WHERE root.mgr_id IS NULL
) SELECT * FROM reachable_from);
```

REPLACE FUNCTION

Input:

```
REPLACE FUNCTION up_load1.RPT_016_BUS_DATE()
RETURNS DATE
LANGUAGE SQL
CONTAINS SQL
DETERMINISTIC
SQL SECURITY DEFINER
COLLATION INVOKER
INLINE TYPE 1
RETURN DATE'2017-08-22';
```

Output:

```
CREATE OR REPLACE FUNCTION up_load1.RPT_016_BUS_DATE()
RETURNS DATE
LANGUAGE SQL
IMMUTABLE
SECURITY DEFINER
AS
$$
SELECT CAST('2017-08-20' AS DATE)
$$
;
```


CHECK OPTION

Use the **tdMigrateVIEWCHECKOPTIO....** configuration parameter to configure migration of views containing the WITH CHECK OPTION keyword.

If a view with **CHECK OPTION** is present in the source, then the **CHECK OPTION** is commented from the target database.

Input - VIEW with CHECK OPTION

```
CV  mgr15 AS SEL *
FROM
   employee
WHERE
   manager_id = 15 WITH CHECK OPTION
;
```

Output (tdMigrateVIEWCHECKOPTION=True)

```
CREATE
    OR REPLACE VIEW mgr15 AS (
        SELECT
                *
            FROM
                employee
            WHERE
                manager_id = 15 /*WITH CHECK OPTION */
    )
;
```

Output (tdMigrateVIEWCHECKOPTION=False)

```
CV  mgr15 AS SEL *
FROM
   employee
WHERE
   manager_id = 15 WITH CHECK OPTION
;
```

VIEW WITH RECURSIVE

GaussDB(DWS) does not support the Teradata keyword **RECURSIVE VIEW**. Therefore the keyword is replaced with **VIEW WITH RECURSIVE** keyword as shown in the following figures.

Figure 6-3 Input view-CREATE RECURSIVE VIEW

```
CREATE
    RECURSIVE VIEW emp_hier (
        emp_id
        ,mgr_id
        ,LEVEL
    ) AS (
        SELECT
                a.emp_id
                ,a.mgr_id
                ,0
            FROM
                employee a
            WHERE
                a.emp_id = 123
```


Figure 6-4 Output view

```
CREATE
    VIEW emp_hier AS (
        WITH RECURSIVE emp_hier (
            emp_id
            ,mgr_id
            ,LEVEL
        ) AS (
            SELECT
                    a.emp_id
                    ,a.mgr_id
                    ,0
                FROM
                    employee a
                WHERE
                    a.emp_id = 123
```

VIEW WITH ACCESS LOCK

Use the **tdMigrateLOCKOption** configuration parameter to configure migration of query containing the LOCK keyword. If **tdMigrateLOCKOption** is set to **false**, the tool will skip migration of the query and will log a message.

Input - VIEW with ACCESS LOCK

```
CREATE OR REPLACE VIEW DP_SVMEDW.S_LCR_909_001_LCRLOAN
 AS
 LOCK TABLE DP_STEDW.S_LCR_909_001_LCRLOAN FOR ACCESS  FOR ACCESS
 ( SELECT RUN_ID, PRODUCT_ID, CURRENCY
          , CASHFLOW, ENTITY, LCR
          , TIME_BUCKET, MT, Ctl_Id
          , File_Id, Business_Date
     FROM DP_STEDW.S_LCR_909_001_LCRLOAN ) ;
```

Output:

```
CREATE OR REPLACE VIEW DP_SVMEDW.S_LCR_909_001_LCRLOAN
 AS
/* LOCK TABLE DP_STEDW.S_LCR_909_001_LCRLOAN FOR ACCESS */
 ( SELECT RUN_ID, PRODUCT_ID, CURRENCY
          , CASHFLOW, ENTITY, LCR
          , TIME_BUCKET, MT, Ctl_Id
          , File_Id, Business_Date
     FROM DP_STEDW.S_LCR_909_001_LCRLOAN ) ;
```

dbc.columnsV

Input:

```
SELECT A.ColumnName
 AS V_COLS             ,A.columnname  || ' ' ||CASE WHEN columnType in ('CF','CV')
THEN CASE WHEN columnType='CV' THEN 'VAR' ELSE ''             END||'CHAR('||TRIM(columnlength
(INT))||                ') CHARACTER SET LATIN'||                  CASE WHEN UpperCaseFlag='N'
THEN ' NOT' ELSE ''
END || ' CASESPECIFIC'
WHEN columnType='DA' THEN 'DATE'
WHEN columnType='TS' THEN 'TIMESTAMP(' || TRIM(DecimalFractionalDigits)||')'
WHEN columnType='AT' THEN 'TIME('|| TRIM(DecimalFractionalDigits)||')'                                WHEN
columnType='I' THEN 'INTEGER'
 WHEN columnType='I1' THEN 'BYTEINT'
WHEN columnType='I2' THEN 'SMALLINT'
```



```
WHEN columnType='I8' THEN 'BIGINT'
WHEN columnType='D' THEN 'DECIMAL('||TRIM(DecimalTotalDigits)||','||
TRIM(DecimalFractionalDigits)||')'                    ELSE 'Unknown'
END||CASE WHEN Nullable='Y'
THEN '' ELSE ' NOT NULL' END||'0A'XC
AS V_ColT
,D.ColumnName
AS V_PICol -- Obtain the primary index of the target table.
FROM dbc.columnsV A LEFT JOIN dbc.IndicesV B  ON A.columnName = B.columnName AND B.IndexType
IN ('Q','P') AND B.DatabaseName = '${V_TDDLDB}'  AND B.tablename='${TARGET_TABLE}' WHERE
A.databasename='${V_TDDLDB}' AND A.tablename = '${TARGET_TABLE}' AND A.columnname NOT IN
( 'ETL_JOB_NAME'                                                                    ,'ETL_TX_DATE' ,
'ETL_PROC_DATE'
                    )ORDER BY A.columnid;
```

Output:

```
D DECLARE lv_mig_V_COLS    TEXT;          lv_mig_V_ColT         TEXT;          lv_mig_V_PICol        TEXT; BEGIN
SELECT STRING_AGG(A.ColumnName, ',')                  , STRING_AGG(A.columnname  || ' ' ||CASE WHEN
columnType in ('CF','CV')                             THEN CASE WHEN columnType='CV' THEN 'VAR' ELSE
''             END||'CHAR('||TRIM(mig_td_ext.mig_fn_castasint(columnlength))||                ') /*CHARACTER SET
LATIN*/'||                  CASE WHEN UpperCaseFlag='N'                    THEN ' NOT' ELSE ''                    END
|| ' /*CASESPECIFIC*/'                              WHEN columnType='DA' THEN 'DATE'
WHEN columnType='TS' THEN 'TIMESTAMP(' || TRIM(DecimalFractionalDigits)||')'
WHEN columnType='AT' THEN 'TIME('|| TRIM(DecimalFractionalDigits)||')'                                  WHEN
columnType='I' THEN 'INTEGER'                         WHEN columnType='I1' THEN
'BYTEINT'                           WHEN columnType='I2' THEN 'SMALLINT'                                WHEN
columnType='I8' THEN 'BIGINT'                         WHEN columnType='D' THEN 'DECIMAL('||
TRIM(DecimalTotalDigits)||','||TRIM(DecimalFractionalDigits)||')'                                  ELSE
'Unknown'                           END||CASE WHEN Nullable='Y'        THEN '' ELSE ' NOT NULL' END||E'\x0A',
',')                , STRING_AGG(B.ColumnName, ',')               INTO lv_mig_V_COLS, lv_mig_V_ColT,
lv_mig_V_PICol FROM mig_td_ext.vw_td_dbc_columnsV A LEFT JOIN mig_td_ext.vw_td_dbc_IndicesV B  ON
A.columnName = B.columnName AND B.IndexType IN ('Q','P') AND B.DatabaseName = 'public'  AND
B.tablename='emp2' WHERE A.databasename='public' AND A.tablename = 'emp2'; -- ORDER BY
A.columnid; END; /
```

6.4.8 COLLECT STATISTICS

COLLECT STAT is used in Teradata for collecting optimizer statistics, which will be used for query performance. GaussDB(DWS) uses the ANALYZE statement to replace COLLECT STAT.

Learn more information in **ANALYZE**.

Input - COLLECT STATISTICS

```
COLLECT STAT tab1 COLUMN (c1, c2);
```

Output:

```
ANALYZE tab1 (c1, c2);
```

Input - COLLECT STATISTICS

```
COLLECT STATISTICS
  COLUMN (customer_id,customer_name)
, COLUMN (postal_code)
, COLUMN (customer_address)
ON customer_t;
```

Output:

```
ANALYZE customer_t (
    customer_id
    ,customer_name
    ,postal_code
    ,customer_address
)
;
```


Input - COLLECT STATISTICS with COLUMN

```
COLLECT STATISTICS
  COLUMN (
  Order_Date
  -- ,o_orderID
/*COLLECT
STATISTICS*/
  ,Order_ID
     )
ON order_t;
```

Output:

```
ANALYZE order_t (
    Order_Date
     ,Order_ID
)
;
```

Input - COLLECT STATISTICS with Schema Name

```
COLLECT STATS COLUMN (
     empno
     ,ename
)
     ON ${schemaname}."usrTab1"
;
```

Output:

```
ANALYZE ${schemaname}."usrTab1"
(
     empno
     ,ename
)
;
```

COLLECT STATISTICS

Collect statistics based on sampling percentage.

Input:

```
COLLECT STATISTICS
 USING SAMPLE 5.00 PERCENT
 COLUMN ( CDR_TYPE_KEY ) ,
 COLUMN ( PARTITION ) ,
 COLUMN ( SRC ) ,
 COLUMN ( PARTITION,SBSCRPN_KEY )
 ON DT_SDM.FCT_OTGO_NTWK_ACTVY_DAILY ;
```

Output:

```
SET
default_statistics_target = 5.00 ;
ANALYZE DT_SDM.FCT_OTGO_NTWK_ACTVY_DAILY (CDR_TYPE_KEY) ;
ANALYZE DT_SDM.FCT_OTGO_NTWK_ACTVY_DAILY (PARTITION) ;
ANALYZE DT_SDM.FCT_OTGO_NTWK_ACTVY_DAILY (SRC) ;
ANALYZE DT_SDM.FCT_OTGO_NTWK_ACTVY_DAILY (PARTITION,SBSCRPN_KEY) ;
 RESET default_statistics_target ;
```

6.4.9 ACCESS LOCK

ACCESS LOCK allows you to read the data from a table that may have been locked for the READ or WRITE.

Use the **tdMigrateLOCKoption** configuration parameter to configure migration of query containing the LOCK keyword. If **tdMigrateLOCKOption** is set to **false**, the tool will skip migration of the query and will log a message.

Input - ACCESS LOCK (tdMigrateLOCKOption=True)

```
LOCKING TABLE tab1 FOR ACCESS
 INSERT INTO tab2
SELECT …
 FROM …
  WHERE ...;
```

Output:

```
/* LOCKING TABLE tab1 FOR ACCESS */
INSERT INTO tab2
SELECT …
 FROM …
 WHERE ...;
```

6.4.10 DBC.COLUMNS

DBC.COLUMNS view is a table containing information about table and view columns, stored procedures, or macro parameters. The view includes the following column names: **DatabaseName**, **TableName**, **ColumnName**, **ColumnFormat**, **ColumnTitle**, **ColumnType**, and **DefaultValue**. In GaussDB(DWS), this table is equivalent to the **information_schema.columns** table.

NOTE

This feature requires one time execution of the custom script file *DSC/scripts/teradata/db_scripts/**mig_fn_get_datatype_short_name.sql***.

For more information about the steps to execute the file, refer to **Operating Environment** and **Prerequisites** respectively.

The DSC migrates the following dbc.columns to their corresponding information_schema columns.

Table 6-24 Migration of dbc.columns to information_schema columns

dbc.columns	information_schema.columns
ColumnName	Column_Name
ColumnType	mig_fn_get_datatype_short_name (data_Type)
ColumnLength	character_maximum_length
DecimalTotalDigits	numeric_precision
DecimalFractionalDigits	numeric_scale
databasename	table_schema
tablename	table_name
ColumnId	ordinal_position

The following assumptions are made when migrating dbc.columns:

- The FROM clause will contain only the dbc.columns TABLE NAME
- COLUMN NAME can be in the form of *column_name* or *schema_name.table_name.column_name*.

Migration of dbc.columns is not supported for the following cases:

- If the FROM clause has an ALIAS for dbc.columns table name (dbc.columns alias).
- If dbc.columns is combined with other tables (FROM dbc.columns alias1, table1 alias2 OR dbc.columns alias1 join table1 alias2).

NOTE

- If the input SELECT statement includes dbc.column COLUMN NAMES directly, then the tool will migrate the input column names as an ALIAS. For example, the input column name **DecimalFractionalDigits** is migrated to *numeric_scale* with an ALIAS **DecimalFractionalDigits**.

  Example:

  Input:

  ```
  SEL
        columnid
       ,DecimalFractionalDigits
      FROM
       dbc.columns
  ;
  ```

  Output:

  ```
  SELECT
        ordinal_position columnid
       ,numeric_scale DecimalFractionalDigits
      FROM
       information_schema.columns
  ;
  ```

- For table names and schema names, the DSC will convert all string values to lowercase. To maintain case-sensitivity, the table/schema names should be within double quotes. In the following input example, "Test" will not be converted to lower case.

  ```
  SELECT
          TableName
      FROM
          dbc . columns
      WHERE
          dbc.columns.databasename = '"Test"';
  ```

Input: dbc.columns table with all supported columns

```
SELECT
'$AUTO_DB_IP'
,objectdatabasename
,objecttablename
,'$TX_DATE_10'
,''
,'0'
,FirstStepTime
,FirstRespTime
,RowCount
,cast(RowCount*sum(case when T2.ColumnType ='CV' then T2.ColumnLength/3 else  T2.ColumnLength end)
as decimal(38,0))
,'3'
,''
,'BAK_CLR_DATA'
,'2'
,''
```



```
FROM TMP_clr_information T1
inner join dbc.columns T2
on T1.objectdatabasename =T2.DatabaseName
and T1.objecttablename =T2.TableName
where  T2.DatabaseName not in (
sel child from dbc.children
where parent='$FCRM_DB'
)
group by 1,2,3,4,5,6,7,8,9,11,12,13,14,15;
```

Output:

```
SELECT
        '$AUTO_DB_IP'
        ,objectdatabasename
        ,objecttablename
        ,'$TX_DATE_10'
        ,''
        ,'0'
        ,FirstStepTime
        ,FirstRespTime
        ,RowCount
        ,CAST( RowCount * SUM ( CASE WHEN mig_fn_get_datatype_short_name ( T2.data_Type ) = 'CV'
THEN T2.character_maximum_length / 3 ELSE T2.character_maximum_length END ) AS
DECIMAL( 38 ,0 ) )
        ,'3'
        ,''
        ,'BAK_CLR_DATA'
        ,'2'
        ,''
    FROM
        TMP_clr_information T1 INNER JOIN information_schema.columns T2
            ON T1.objectdatabasename = T2.table_schema
        AND T1.objecttablename = T2.table_name
    WHERE
        NOT EXISTS (
            SELECT
                    child
                FROM
                    dbc.children
                WHERE
                    child = T2.table_schema
                    AND( parent = '$FCRM_DB' )
        )
    GROUP BY
        1 ,2 ,3 ,4 ,5 ,6 ,7 ,8 ,9 ,11 ,12 ,13 ,14 ,15
;
```

Input: dbc.columns table with TABLE NAME

```
SELECT
        TRIM( ColumnName )
        ,UPPER( dbc.columns.ColumnType )
    FROM
        dbc . columns
    WHERE
        dbc.columns.databasename = '"Test"'
    ORDER BY
        dbc.columns.ColumnId
;
```

Output:

```
SELECT
        TRIM( Column_Name )
        ,UPPER( mig_fn_get_datatype_short_name ( information_schema.columns.data_Type ) )
    FROM
        information_schema.columns
    WHERE
        information_schema.columns.table_schema = CASE
```



```
            WHEN TRIM( '"Test"' ) LIKE '"%'
            THEN REPLACE( SUBSTR( '"Test"' ,2 ,LENGTH( '"Test"' ) - 2 ) ,'""' ,'"' )
            ELSE LOWER( '"Test"' )
        END
    ORDER BY
        information_schema.columns.ordinal_position
;
```

6.4.11 DBC.TABLES

The DSC migrates dbc.tables to their corresponding mig_td_ext.vw_td_dbc_tables.

Example: databasename is migrated as mig_td_ext.vw_td_dbc_tables.schemaname.

Input:

```
sel databasename,tablename FROM dbc.tables
WHERE tablekind='T' and trim(databasename) = '<dbname>'
AND
( NOT(TRIM(tablename) LIKE ANY  (<excludelist>))
);
```

Output:

```
SELECT
        mig_td_ext.vw_td_dbc_tables.schemaname
                        , mig_td_ext.vw_td_dbc_tables.tablename

    FROM
        mig_td_ext.vw_td_dbc_tables
    WHERE
                        mig_td_ext.vw_td_dbc_tables.tablekind =  'T'
        AND TRIM(mig_td_ext.vw_td_dbc_tables.schemaname) = '<dbname>'
        AND( NOT( TRIM(mig_td_ext.vw_td_dbc_tables.tablename) LIKE ANY ( ARRAY[ < excludelist > ] ) ) )
;
```

6.4.12 DBC.INDICES

DSC migrates dbc.indices to the corresponding mig_td_ext.vw_td_dbc_indices.

Example: databasename is migrated as mig_td_ext.vw_td_dbc_tables.schemaname.

Input:

```
sel databasename,tablename FROM dbc.indices
WHERE tablekind='T' and trim(databasename) = '<dbname>'
AND
( NOT(TRIM(tablename) LIKE ANY  (<excludelist>))
) AND indextype IN ( 'Q','P');
```

Output:

```
SELECT
        mig_td_ext.vw_td_dbc_indices.schemaname

, mig_td_ext.vw_td_dbc_indices.tablename

    FROM
        mig_td_ext.vw_td_dbc_indices
    WHERE

mig_td_ext.vw_td_dbc_indices.tablekind =  'T'

AND TRIM(mig_td_ext.vw_td_dbc_indices.schemaname) =
'<dbname>'

AND( NOT( TRIM(mig_td_ext.vw_td_dbc_indices.tablename) LIKE ANY (
```



```
ARRAY[ < excludelist > ] ) ) )
;
```

NOTE

In dbc.indices implementation, the query should contain "AND indextype IN ('Q','P')". If the query does not contain "AND indextype IN ('Q','P')", then the query is not migrated and DSC logs the following error message:

"Query/statement is not supported as indextype should be mentioned with values 'P' and 'Q'."

dbc.sessioninfoV

Input:

```
select username,clientsystemuserid,clientipaddress,clientprogramname
  from dbc.sessioninfoV
 where sessionno = 140167641814784;
```

Output:

```
select usename AS username, NULL::TEXT AS clientsystemuserid
  , client_addr AS clientipaddress, application_name AS clientprogramname
  from pg_catalog.pg_stat_activity
 WHERE pid = 140167641814784;
```

dbc.sessioninfo

Input:

```
SELECT  username
,clientsystemuserid
 ,clientipaddress
,clientprogramname
 FROM
dbc.sessioninfo
WHERE
sessionno = lv_mig_session ;
```

Output:

```
select usename AS username, NULL::TEXT AS clientsystemuserid
  , client_addr AS clientipaddress, application_name AS clientprogramname
  from pg_catalog.pg_stat_activity
 WHERE pid = lv_mig_session;
```

Teradata "SET QUERY_BAND" with "FOR SESSION"

Input:

```
set query_band = 'AppName=${AUTO_SYS};JobName=${AUTO_JOB};TxDate=${TX_DATE};ScriptName=$
{script_name};' for session ;
```

Output:

```
set query_band = 'AppName=${AUTO_SYS};JobName=${AUTO_JOB};TxDate=${TX_DATE};ScriptName=$
{script_name};' /* for session */;
```

SESSION

Input:

```
select Session ;
should be migrated as below:
SELECT pg_backend_pid();
```


Output:

```
SELECT pg_backend_pid();
```

6.4.13 SHOW STATS VALUES SEQUENCED

This command displays the COLLECT STATISTICS statement with statistics. Gauss does not have an equivalent for this. Since it does not affect the functionality, this command can be commented.

Input:

```
SHOW STATS VALUES SEQUENCED on "temp"."table"
```

Output:

```
/*SHOW STATS VALUES SEQUENCED on "temp"."table"*/
```

6.4.14 COMMENT

The figure below illustrates the process of simplifying **COMMENT** statements. A complex statement is entered and then simplified through migration.

Input:

```
CREATE MULTISET TABLE PDAT.t_tbl_comment
   (
      Data_Dt  DATE           NOT NULL
     ,Data_Src  VARCHAR(4)    NOT NULL
     ,List_Make_Stat  CHAR(1)
   ) PRIMARY INDEX(Data_Dt,Data_Src) ;
COMMENT ON PDAT.t_tbl_comment IS 'comment test on table';
---------------
REPLACE VIEW  PVW.v_vw_comment AS
LOCKING ROW FOR ACCESS
SELECT Data_Dt, Data_Src, List_Make_Stat
FROM PDAT.t_tbl_comment;
COMMENT ON PVW.v_vw_comment IS 'comment test on view';
COMMENT ON PVW.v_vw_comment.Data_Dt IS 'comment test on view column';
```

Output:

```
COMMENT ON TABLE PDAT.t_tbl_comment IS 'comment test on table';
COMMENT ON VIEW PVW.v_vw_comment IS 'comment test on view';
COMMENT ON COLUMN PVW.v_vw_comment.Data_Dt IS 'comment test on view column';
```

6.4.15 Data Manipulation Language (DML)

6.4.15.1 INSERT

The Teradata **INSERT** (**short key** INS) statement is used to insert records into a table. DSC supports the **INSERT** statement.

The **INSERT INTO TABLE table_name** syntax is used in Teradata SQL, but is not supported by GaussDB(DWS). GaussDB(DWS) supports only **INSERT INTO table_name**. Therefore, remove the keyword **TABLE** when using DSC.

Input

```
INSERT TABLE tab1
SELECT col1, col2
  FROM tab2
 WHERE col3 > 0;
```


Output

```
INSERT INTO tab1
SELECT col1, col2
  FROM tab2
 WHERE col3 > 0;
```

6.4.15.2 SELECT

ANALYZE

The Teradata **SELECT** command (**short key** SEL) is used to specify the table columns from which data is to be retrieved.

ANALYZE is used in GaussDB(DWS) for collecting optimizer statistics, which is used for improving query performance.

Input: ANALYZE with INSERT

```
INSERT INTO employee(empno,ename)  Values (1,'John');
COLLECT STAT on employee;
```

Output

```
INSERT INTO employee( empno, ename)
SELECT 1 ,'John';
ANALYZE employee;
```

Input: ANALYZE with UPDATE

```
UPD employee SET ename = 'Jane'
       WHERE ename = 'John';
COLLECT STAT on employee;
```

Output

```
UPDATE employee SET ename = 'Jane'
 WHERE ename = 'John';
ANALYZE employee;
```

Input: ANALYZE with DELETE

```
DEL FROM employee WHERE ID > 10;
COLLECT STAT on employee;
```

Output

```
DELETE FROM employee WHERE ID > 10;
ANALYZE employee;
```

Order of Clauses

For Teradata migration of **SELECT** statements, all the clauses (**FROM**, **WHERE**, **HAVING** and **GROUP BY**) can be listed in any order. If the **FROM** clause of a statement contains a QUALIFY clause that is used as **ALIAS**, DSC does not migrate the statement.

Use the **tdMigrateALIAS** configuration parameter to configure migration of ALIAS.

Input: Order of Clauses

```
SELECT expr1 AS alias1
     , expr2 AS alias2
     , expr3 AS alias3
```



```
    , MAX( expr4 ), ...
  FROM tab1 T1 INNER JOIN tab2 T2
    ON T1.c1 = T2.c2 ...
   AND T3.c5 = '010'
   AND ...
 WHERE T1.c7 = '000'
   AND ...
HAVING alias1 <> 'IC'
       AND alias2 <> 'IC'
       AND alias3 <> ''
 GROUP BY 1, 2, 3 ;
```

Output

```
SELECT
        expr1 AS "alias1"
        ,expr2 AS "alias2"
        ,expr3 AS "alias3"
        ,MAX( expr4 )
        ,...
    FROM
        tab1 T1 INNER JOIN tab2 T2
            ON T1.c1 = T2.c2 ...
        AND T3.c5 = '010'
        AND ...
    WHERE
        T1.c7 = '000'
        AND ...
    GROUP BY
        1 ,2 ,3
    HAVING
        alias1 <> 'IC'
        AND alias2 <> 'IC'
        AND alias3 <> '' ;
```

Input: Order of Clauses

```
SELECT
        TOP 10 *
    GROUP BY
        DeptNo
    WHERE
        empID < 100
FROM
        tbl_employee;
```

Output

```
SELECT
        *
    FROM
        tbl_employee
    WHERE
        empID < 100
    GROUP BY
        DeptNo LIMIT 10
;
```

> **NOTE**
>
> If the input script contains **QUALIFY** as an **ALIAS** before the **FROM** clause, DSC will not migrate the statement and copy the input statement verbatim.

Input: Order of Clauses with QUALIFY as an ALIAS before the FROM clause

```
SELECT
        *
    FROM
        table1
    WHERE
```



```
        abc = (
            SELECT
                    col1 AS qualify
                FROM
                    TABLE
                    WHERE
                        col1 = 5
        )
;
```

Output

```
SELECT
        *
    FROM
        table1
    WHERE
        abc = (
            SELECT
                    col1 AS qualify
                FROM
                    TABLE
                    WHERE
                        col1 = 5
        )
;
```

Extended Group By Clause

The **GROUP BY** clause can be specified if you want the database to group the selected rows based on the value of expr(s). If this clause contains **CUBE**, **ROLLUP** or **GROUPING SETS** extensions, the database produces super-aggregate groupings in addition to the regular groupings. These features are not available in GaussDB(DWS), but similar functions can be enabled using the **UNION ALL** operator.

Use the **extendedGroupByClause** configuration parameter to configure migration of the extended **GROUP BY** clause.

Input: Extended Group By Clause - CUBE

```
SELECT expr1 AS alias1
     , expr2 AS alias2
     , expr3 AS alias3
     , MAX( expr4 ), ...
  FROM tab1 T1 INNER JOIN tab2 T2
    ON T1.c1 = T2.c2 ...
   AND T3.c5 = '010'
   AND ...
 WHERE T1.c7 = '000'
   AND ...
HAVING alias1 <> 'IC'
       AND alias2 <> 'IC'
       AND alias3 <> ''
 GROUP BY 1, 2, 3 ;
```

Output

```
SELECT
        expr1 AS "alias1"
        ,expr2 AS "alias2"
        ,expr3 AS "alias3"
        ,MAX( expr4 )
        ,...
    FROM
        tab1 T1 INNER JOIN tab2 T2
            ON T1.c1 = T2.c2 ...
```



```
        AND T3.c5 = '010'
        AND ...
    WHERE
        T1.c7 = '000'
        AND ...
    GROUP BY
        1 ,2 ,3
    HAVING
        alias1 <> 'IC'
        AND alias2 <> 'IC'
        AND alias3 <> '' ;
```

Input: Extended Group By Clause - ROLLUP

```
SELECT d.dname, e.job, MAX(e.sal)
  FROM emp e RIGHT OUTER JOIN dept d
    ON e.deptno=d.deptno
WHERE e.job IS NOT NULL
GROUP BY ROLLUP (d.dname, e.job);
```

Output

```
SELECT dname, job, ColumnAlias1
  FROM ( SELECT MAX(e.sal) AS ColumnAlias1, d.dname, e.job
           FROM emp e RIGHT OUTER JOIN dept d
             ON e.deptno = d.deptno
          WHERE e.job IS NOT NULL
          GROUP BY d.dname ,e.job
          UNION ALL
         SELECT MAX(e.sal) AS ColumnAlias1, d.dname, NULL AS
                job
           FROM emp e RIGHT OUTER JOIN dept d
             ON e.deptno = d.deptno
          WHERE e.job IS NOT NULL
          GROUP BY d.dname
          UNION ALL
         SELECT MAX( e.sal ) AS ColumnAlias1, NULL AS dname,
                NULL AS job
           FROM emp e RIGHT OUTER JOIN dept d
             ON e.deptno = d.deptno
          WHERE e.job IS NOT NULL
       );
```

Input: Extended Group By Clause - GROUPING SETS

```
SELECT d.dname, e.job, MAX(e.sal)
FROM emp e RIGHT OUTER JOIN dept d
ON e.deptno=d.deptno
WHERE e.job IS NOT NULL
GROUP BY GROUPING SETS(d.dname, e.job);
```

Output

```
SELECT dname, job, ColumnAlias1
  FROM ( SELECT MAX(e.sal) AS ColumnAlias1
           , d.dname, NULL AS job
           FROM emp e RIGHT OUTER JOIN dept d
             ON e.deptno = d.deptno
          WHERE e.job IS NOT NULL
          GROUP BY d.dname
          UNION ALL
         SELECT MAX(e.sal) AS ColumnAlias1
           , NULL AS dname, e.job
           FROM emp e RIGHT OUTER JOIN dept d
             ON e.deptno = d.deptno
          WHERE e.job IS NOT NULL
          GROUP BY e.job
       );
```


SELECT AS

GaussDB(DWS) variable names are case insensitive, while Teradata variable names are case sensitive. To ensure that the Teradata script is correct before and after the migration, retain the case of the original variable name in the variable definition of the **SELECT** statement. The converted variable is defined in the **AS** *Variable name*.

Input

```
SELECT  TRIM('${JOB_NAME}')                                                    AS JOB_NAME
      ,CASE WHEN LENGTH(trim(STRTOK('${JOB_NAME}','-',4)))=2
           THEN trim(STRTOK('${JOB_NAME}','-',4))
            ELSE ''
            END                                                          AS EDW_BANK_NM
      ,TRIM('${TX_DATE}')                                                 AS TX_DATE
      ,USER                                                            AS ETL_USER
      ,CAST( CURRENT_TIMESTAMP(0) AS VARCHAR(19))                                    AS CURR_STIME
      ,'${ETL_DATA}'                                                  AS ETL_DATA
      ,'T61_INDV_CUST_ACCT_ORG_AUM'                                          AS TARGET_TABLE
      ,'CAST(''8999-12-31'' AS DATE)'                                       AS MAXDATE
;
.IF ERRORCODE <> 0 THEN .QUIT 12
```

Output

```
SELECT
         TRIM( '${job_name}' ) AS "JOB_NAME"
         ,CASE
                  WHEN LENGTH( TRIM( split_part ( '${job_name}' ,'-' ,4 ) ) ) = 2 THEN TRIM( split_part ( '$
{job_name}' ,'-' ,4 ) )
              ELSE ''
         END AS "EDW_BANK_NM"
         ,TRIM( '${tx_date}' ) AS "TX_DATE"
         ,USER AS "ETL_USER"
         ,CAST( CURRENT_TIMESTAMP( 0 ) AS VARCHAR( 19 ) ) AS "CURR_STIME"
         ,'${etl_data}' AS "ETL_DATA"
         ,'T61_INDV_CUST_ACCT_ORG_AUM' AS "TARGET_TABLE"
         ,'CAST(''8999-12-31'' AS DATE)' AS "MAXDATE" ;
\if ${ERROR} != 'false'
 \q 12
\endif

;
```

Definition nested with **AS** expression is implemented by splitting multiple statements.

Input

```
SELECT  TRIM('${JOB_NAME}')                                                    AS JOB_NAME
      ,'CAST(''0001-01-02'' AS DATE)'                                       AS ILLDATE
      ,'T61_INDV_CUST_HOLD_PROD_IND_AUM'                                           AS
TARGET_TABLE
      ,0                                                               AS NULLNUMBER
      ,'CAST(''00:00:00.999'' AS TIME(3))'                                  AS NULLTIME
      ,'CAST(''0001-01-01 00:00:00.000000'' AS TIMESTAMP(6))'                       AS NULLTIMESTAMP
      ,'VT_'||TARGET_TABLE                                             AS VT_TABLE
      ,'V'||SUBSTR(TARGET_TABLE,2,CHAR(TARGET_TABLE)-1)                            AS
TARGET_TABLE_V
      ,'${GDM_DETAIL_DDL}'                                                AS V_TDDLDB
      ,'${GDM_DETAIL_VIEW}'                                               AS V_TARGETDB
      ,'${UDF}'                                                        AS V_PUB_UDF

;
.IF ERRORCODE <> 0 THEN .QUIT 12
```

Output

```
SELECT
        TRIM( '${job_name}' ) AS "JOB_NAME"
        ,'CAST(''0001-01-02'' AS DATE)' AS "ILLDATE"
        ,'T61_INDV_CUST_HOLD_PROD_IND_AUM' AS "TARGET_TABLE"
        ,0 AS "NULLNUMBER"
        ,'CAST(''00:00:00.999'' AS TIME(3))' AS "NULLTIME"
        ,'CAST(''0001-01-01 00:00:00.000000'' AS TIMESTAMP(6))' AS "NULLTIMESTAMP"
        ,'${gdm_detail_ddl}' AS "V_TDDLDB"
        ,'${gdm_detail_view}' AS "V_TARGETDB"
        ,'${udf}' AS "V_PUB_UDF" ;
SELECT
        'VT_' || '${TARGET_TABLE}' AS "VT_TABLE" ;
SELECT
        'V' || SUBSTR( '${TARGET_TABLE}' ,2 ,LENGTH( '${TARGET_TABLE}' ) - 1 ) AS "TARGET_TABLE_V" ;
\if ${ERROR} != 'false'
 \q 12
\endif

;
```

TOP Clauses

DSC also supports the migration of **TOP** statements with dynamic parameters. The **TOP** clauses of Teradata are migrated to the **LIMIT** clauses in GaussDB(DWS).

NOTE

- When migrating a statement with a **TOP** clause that includes **WITH TIES**, it is necessary to include the **ORDER BY** clause as well. Otherwise, the tool will be unable to migrate the statement, and it will be copied as it is.
- When using TOP with dynamic parameters:
  - The input dynamic parameters should be in the following form:
    ```
    TOP :<parameter_name>
    ```
    The following characters are allowed: lowercase letters (a-z), uppercase letters (A-Z), digits (0-9), and underscores (_).

Input: SELECT...TOP

```
SELECT TOP 1 c1, COUNT (*) cnt
  FROM tab1
 GROUP BY c1
 ORDER BY cnt;
```

Output

```
SELECT c1, COUNT( * ) cnt
  FROM tab1
 GROUP BY c1
 ORDER BY cnt
 LIMIT 1;
```

Input: SELECT...TOP PERCENT

```
SELECT TOP 10 PERCENT c1, c2
  FROM employee
 WHERE ...
 ORDER BY c2 DESC;
```

Output

```
WITH top_percent AS (
      SELECT c1, c2
        FROM employee
       WHERE ...
       ORDER BY c2 DESC
                  )
```



```
SELECT *
  FROM top_percent
 LIMIT (SELECT CEIL(COUNT( * ) * 10 / 100)
         FROM top_percent);
```

Input: SELECT...TOP with dynamic parameters

```
SELECT
          TOP :Limit WITH TIES c1
         ,SUM (c2) sc2
    FROM
        tab1
    WHERE
        c3 > 10
    GROUP BY
        c1
    ORDER BY
        c1
;
```

Output

```
WITH top_ties AS (
    SELECT
             c1
            ,SUM (c2) sc2
            ,rank (
            ) OVER( ORDER BY c1 ) AS TOP_RNK
        FROM
            tab1
        WHERE
            c3 > 10
        GROUP BY
            c1
) SELECT
        c1
        ,sc2
    FROM
        top_ties
    WHERE
        TOP_RNK <= :Limit
    ORDER BY
        TOP_RNK
;
```

Input: SELECT...TOP with dynamic parameters and TIES

```
 SELECT
          TOP :Limit WITH TIES Customer_ID
  FROM
        Customer_t
  ORDER BY
        Customer_ID
;
```

Output

```
WITH top_ties AS (
    SELECT
            Customer_ID
            ,rank (
            ) OVER( order by Customer_id) AS TOP_RNK
        FROM
            Customer_t
) SELECT
        Customer_ID
    FROM
        top_ties
    WHERE
        TOP_RNK <= :Limit
```



```
    ORDER BY
        TOP_RNK
;
```

Input: SELECT...TOP PERCENT with dynamic parameters

```
SELECT
        TOP :Input_Limit PERCENT WITH TIES c1
        ,SUM (c2) sc2
    FROM
        tab1
    GROUP BY
        c1
    ORDER BY
        c1
;
```

Output

```
WITH top_percent_ties AS (
    SELECT
            c1
            ,SUM (c2) sc2
            ,rank (
            ) OVER( ORDER BY c1 ) AS TOP_RNK
        FROM
            tab1
        GROUP BY
            c1
) SELECT
        c1
        ,sc2
    FROM
        top_percent_ties
    WHERE
        TOP_RNK <= (
            SELECT
                    CEIL(COUNT( * ) * :Input_Limit / 100)
                FROM
                    top_percent_ties
        )
    ORDER BY
        TOP_RNK
;
```

SAMPLE clauses

The **SAMPLE** clause of Teradata is migrated to the **LIMIT** clause in GaussDB(DWS).

NOTE

The tool only supports single positive integers in the **SAMPLE** clause.

Input: SELECT...SAMPLE

```
SELECT c1, c2, c3
  FROM tab1
 WHERE c1 > 1000
SAMPLE 1;
```

Output

```
SELECT c1, c2, c3
  FROM tab1
 WHERE c1 > 1000
 LIMIT 1;
```


6.4.15.3 UPDATE

DSC supports migration of the **UPDATE** (**short key** UPD) statements.

Input: UPDATE with TABLE ALIAS

```
UPDATE T1
  FROM tab1 T1, tab2 T2
   SET c1 = T2.c1
     , c2 = T2.c2
 WHERE T1.c3 = T2.c3;
```

Output

```
UPDATE tab1 T1
   SET c1 = T2.c1
     , c2 = T2.c2
  FROM tab2 T2
 WHERE T1.c3 = T2.c3;
```

Input: UPDATE with TABLE ALIAS using a sub query

```
UPDATE t1
  FROM tab1 t1, ( SELECT c1, c2 FROM tab2
                  WHERE c2 > 100 ) t2
   SET c1 = t2.c1
 WHERE t1.c2 = t2.c2;
```

Output

```
 UPDATE tab1 t1
   SET c1 = t2.c1
  FROM ( SELECT c1, c2 FROM tab2
         WHERE c2 > 100 ) t2
 WHERE t1.c2 = t2.c2;
```

Input: UPDATE with ANALYZE

```
UPD employee SET ename = 'Jane'
       WHERE ename = 'John';
COLLECT STAT on employee;
```

Output

```
UPDATE employee SET ename = 'Jane'
 WHERE ename = 'John';
ANALYZE employee;
```

6.4.15.4 DELETE

DELETE (**short key** abbreviated as **DEL**) is an ANSI-compliant SQL syntax operator used to delete existing records from a table. DSC supports the Teradata **DELETE** statement and its short key **DEL**. The **DELETE** statement that does not contain the **WHERE** clause is migrated to **TRUNCATE** in GaussDB(DWS). Use the **deleteToTruncate** parameter to enable or disable this behavior.

Input: DELETE

```
DEL FROM tab1
 WHERE a =10;
```

Output

```
DELETE FROM tab1
 WHERE a =10;
```


Input: DELETE without WHERE - Migrated to TRUNCATE if deletetoTruncate=TRUE

```
DELETE FROM ${schemaname} . "tablename" ALL;
```

Output

```
TRUNCATE
    TABLE
        ${schemaname} . "tablename";
```

In DELETE, the same table is used in DELETE and FROM clauses with / without WHERE clause

Input

```
DELETE DP_TMP.M_P_TX_SCV_REMAINING_PARTY
FROM DP_TMP.M_P_TX_SCV_REMAINING_PARTY ALL ;
---
DELETE DP_VMCTLFW.CTLFW_Process_Id
FROM DP_VMCTLFW.CTLFW_Process_Id
WHERE (Process_Name =  :_spVV2 )
AND  (Process_Id  NOT IN (SELECT MAX(Process_Id )(NAMED Process_Id )
                           FROM DP_VMCTLFW.CTLFW_Process_Id
                          WHERE Process_Name =  :_spVV2 )
    );
---
DELETE CPID
FROM DP_VMCTLFW.CTLFW_Process_Id AS CPID
WHERE (Process_Name =  :_spVV2 )
AND  (Process_Id  NOT IN (SELECT MAX(Process_Id )(NAMED Process_Id )
                           FROM DP_VMCTLFW.CTLFW_Process_Id
                          WHERE Process_Name =  :_spVV2 )
    );
```

Output

```
DELETE FROM DP_TMP.M_P_TX_SCV_REMAINING_PARTY;
---
DELETE FROM DP_VMCTLFW.CTLFW_Process_Id
WHERE (Process_Name =  :_spVV2 )
AND  (Process_Id  NOT IN (SELECT MAX(Process_Id )(NAMED Process_Id )
                           FROM DP_VMCTLFW.CTLFW_Process_Id
                          WHERE Process_Name =  :_spVV2 )
    );
---
DELETE FROM DP_VMCTLFW.CTLFW_Process_Id AS CPID
WHERE (Process_Name =  :_spVV2 )
AND  (Process_Id  NOT IN (SELECT MAX(Process_Id )(NAMED Process_Id )
                           FROM DP_VMCTLFW.CTLFW_Process_Id
                          WHERE Process_Name =  :_spVV2 )
    );
```

DELETE table_alias FROM table

Input

```
SQL_Detail10124.sql
delete a
  from ${BRTL_DCOR}.BRTL_CS_POT_CUST_UMPAY_INF_S as a
 where a.DW_Snsh_Dt = cast('${v_Trx_Dt}' as date format 'yyyy-mm-dd')
 and a.DW_Job_Seq = 1 ;
was migrated as below:
     DELETE FROM
         BRTL_DCOR.BRTL_CS_POT_CUST_UMPAY_INF_S AS a
             USING
     WHERE a.DW_Snsh_Dt = CAST( lv_mig_v_Trx_Dt AS DATE )
         AND a.DW_Job_Seq = 1 ;
```



```
SQL_Detail10449.sql
delete a
  from ${BRTL_DCOR}.BRTL_EM_YISHITONG_USR_INF as a
 where a.DW_Job_Seq = 1 ;
was migrated as below:
      DELETE FROM
          BRTL_DCOR.BRTL_EM_YISHITONG_USR_INF AS a
              USING
      WHERE a.DW_Job_Seq = 1 ;
SQL_Detail5742.sql
delete a
  from ${BRTL_DCOR}.BRTL_PD_FP_NAV_ADT_INF as a;
was migrated as
      DELETE a
 FROM
      BRTL_DCOR.BRTL_PD_FP_NAV_ADT_INF AS a ;
```

Output

```
SQL_Detail10124.sql
delete from ${BRTL_DCOR}.BRTL_CS_POT_CUST_UMPAY_INF_S as a
 where a.DW_Snsh_Dt = cast('${v_Trx_Dt}' as date format 'yyyy-mm-dd')
 and a.DW_Job_Seq = 1 ;
SQL_Detail10449.sql
delete from ${BRTL_DCOR}.BRTL_EM_YISHITONG_USR_INF as a
 where a.DW_Job_Seq = 1 ;
SQL_Detail5742.sql
delete from ${BRTL_DCOR}.BRTL_PD_FP_NAV_ADT_INF as a;
```

6.4.15.5 MERGE

MERGE is an ANSI-standard SQL syntax operator used to select rows from one or more sources for updating or inserting into a table or view. The conditions to update or insert to the target table or view can be specified.

Input: MERGE

```
MERGE INTO tab1 A
using ( SELECT c1, c2, ... FROM tab2 WHERE ...) AS B
ON A.c1 = B.c1
 WHEN MATCHED THEN
   UPDATE SET c2 = c2
          , c3 = c3
  WHEN NOT MATCHED THEN
INSERT VALUES (B.c1, B.c2, B.c3);
```

Output

```
WITH B AS (
    SELECT
            c1
            ,c2
            ,...
        FROM
            tab2
        WHERE
            ...
)
,UPD_REC AS (
    UPDATE
            tab1 A
        SET
            c2 = c2
            ,c3 = c3
        FROM
            B
        WHERE
            A.c1 = B.c1 returning A. *
```



```
)
INSERT
    INTO
        tab1 SELECT
                B.c1
                ,B.c2
                ,B.c3
             FROM
                B
            WHERE
                NOT EXISTS (
                    SELECT
                        1
                     FROM
                        UPD_REC A
                     WHERE
                        A.c1 = B.c1
                    )
;
```

6.4.15.6 NAMED

NAMED is used in Teradata to assign a temporary name to an expression or column. The **NAMED** statements for expression names are migrated to **AS** in GaussDB(DWS). The **NAMED** statements for column names are retained in the same syntax.

Input: NAMED Expression migrated to AS

```
SELECT Name, ((Salary + (YrsExp * 200))/12) (NAMED Projection)
  FROM Employee
 WHERE DeptNo = 600 AND Projection < 2500;
```

Output

```
SELECT Name, ((Salary + (YrsExp * 200))/12) AS  Projection
  FROM Employee
 WHERE DeptNo = 600 AND ((Salary + (YrsExp * 200))/12)  < 2500;
```

Input: NAMED AS for Column Name

```
SELECT product_id AS id
  FROM emp where pid=2 or id=2;
```

Output

```
SELECT product_id (NAMED "pid") AS id
  FROM emp where product_id=2 or product_id=2;
```

Input: NAMED( ) for Column Name

```
INSERT INTO Neg100 (NAMED,ID,Dept) VALUES ('TEST',1,'IT');
```

Output

```
INSERT INTO Neg100 (NAMED,ID,Dept) SELECT 'TEST',1, 'IT';
```

Input: NAMED alias with TITLE alias without AS

```
SELECT dept_name (NAMED alias1) (TITLE alias2 )
  FROM employee
 WHERE dept_name like 'Quality';
```

Output

```
SELECT dept_name
    AS alias1
```



```
 FROM employee
WHERE dept_name like 'Quality';
```

Input: NAMED alias with TITLE alias with AS

The DSC will skip the NAMED alias and TITLE alias and use only the AS alias.

```
SELECT sale_name (Named alias1 ) (Title alias2)
   AS alias3
 FROM employee
WHERE sname = 'Stock' OR sname ='Sales';
```

Output

```
SELECT sale_name
   AS alias3
 FROM employee
WHERE sname = 'Stock' OR sname ='Sales';
```

Input: NAMED with TITLE

NAMED and TITLE used together, separated by comma(,) within brackets().

```
SELECT customer_id (NAMED cust_id, TITLE 'Customer Id')
FROM Customer_T
WHERE cust_id > 10;
```

Output

```
SELECT cust_id AS "Customer Id"
FROM   (SELECT customer_id AS cust_id
            FROM   customer_t
            WHERE  cust_id > 10);
```

6.4.15.7 ACTIVITYCOUNT

ACTIVITYCOUNT

Input

It is a status variable that returns the number of rows affected by an SQL DML statement in an embedded SQL.

```
SEL tablename
FROM dbc.tables
WHERE databasename ='tera_db'
  AND tablename='tab1';

.IF ACTIVITYCOUNT > 0 THEN .GOTO NXTREPORT;
CREATE MULTISET TABLE tera_db.tab1
      , NO FALLBACK
      , NO BEFORE JOURNAL
      , NO AFTER JOURNAL
      , CHECKSUM = DEFAULT
        (
                Tx_Zone_Num CHAR( 4 )
              , Tx_Org_Num  VARCHAR( 30 )
        )
        PRIMARY INDEX
        (
                Tx_Org_Num
        )
        INDEX
        (
                Tx_Teller_Id
        )
;
```



```
.LABEL NXTREPORT
DEL FROM tera_db.tab1;
```

Output

```
DECLARE v_verify TEXT ;
v_no_data_found NUMBER ( 1 ) ;

BEGIN
    BEGIN
        v_no_data_found := 0 ;

        SELECT
                mig_td_ext.vw_td_dbc_tables.tablename INTO v_verify
            FROM
                mig_td_ext.vw_td_dbc_tables
            WHERE
                mig_td_ext.vw_td_dbc_tables.schemaname = 'tera_db'
                AND mig_td_ext.vw_td_dbc_tables.tablename = 'tab1' ;

            EXCEPTION
                WHEN NO_DATA_FOUND THEN
                v_no_data_found := 1 ;

    END ;

    IF
        v_no_data_found = 1 THEN
            CREATE TABLE tera_db.tab1 (
                Tx_Zone_Num CHAR( 4 )
                ,Tx_Org_Num VARCHAR( 30 )
            ) DISTRIBUTE BY HASH ( Tx_Org_Num ) ;

    CREATE
        INDEX
            ON tera_db.tab1 ( Tx_Teller_Id ) ;

    END IF ;

    DELETE FROM
        tera_db.tab1 ;

END ;
/
```

6.4.15.8 TIMESTAMP

Input - TIMESTAMP with FORMAT

The FORMAT phrase sets the format for a specific TIME or TIMESTAMP column or value. A FORMAT phrase overrides the system format.

```
SELECT 'StartDTTM' as a
          ,CURRENT_TIMESTAMP (FORMAT 'HH:MI:SSBMMMBDD,BYYYY');
```

Output

```
SELECT 'StartDTTM' AS a
          ,TO_CHAR( CURRENT_TIMESTAMP ,'HH:MI:SS MON DD, YYYY' ) ;
```

TIMESTAMP Typecasting

Input

```
COALESCE( a.Snd_Tm ,TIMESTAMP '0001-01-01 00:00:00' )
```

Output


```
COALESCE( a.Snd_Tm , CAST('0001-01-01 00:00:00' AS TIMESTAMP) )
```

6.4.16 Type Casting and Formatting

This section contains the migration syntax for migrating Teradata type casting and formatting syntax. The migration syntax determines how the keywords and features are migrated.

In Teradata, the FORMAT keyword is used for formatting a column/expression. FORMAT '9(n)' and 'z(n)' are addressed using LPAD with 0 and space (' ') respectively. Data typing can be done using CAST or direct data type [like (expression1)(CHAR(n))]. This feature is addressed using CAST.

The following type casting and formatting statements are supported by the DSC:

- **CHAR**
- **COLUMNS and COLUMN ALIAS**
- **Expression**
- **INT**
- **DATE**
- **DAY to SECOND**
- **DECIMAL**
- **Time Interval**
- **NULL**
- **Implicit Type Casting Issues**

CHAR

Input - Data type casting for CHAR

```
(expression1)(CHAR(n))
```

Output

```
CAST( (expression1) AS CHAR(n) )
```

COLUMNS and COLUMN ALIAS

Input - Type casting and formatting of a column should ensure the column name is the same as the column alias

```
SELECT Product_Line_ID, MAX(Standard_Price)
  FROM ( SELECT A.Product_Description, A.Product_Line_ID
   , A.Standard_Price(DECIMAL(18),FORMAT '9(18)')(CHAR(18))
       FROM product_t A
     WHERE Product_Line_ID in (1, 2)
  ) AS tabAls
 GROUP BY Product_Line_ID;
```

Output

```
SELECT Product_Line_ID, MAX( Standard_Price )
   FROM  ( SELECT A.Product_Description, A.Product_Line_ID
        , CAST( LPAD( CAST(A.Standard_Price AS DECIMAL( 18 ,0 )), 18, '0' ) AS CHAR( 18 ) ) AS
Standard_Price
                FROM product_t A
               WHERE Product_Line_ID IN( 1 ,2 )
```



```
        ) AS tabAls
 GROUP BY Product_Line_ID;
```

Expression

Input - Type casting and formatting of an expression

```
SELECT product_id, standard_price*100.00(DECIMAL (17),FORMAT '9(17)' )(CHAR(17) ) AS order_amt
  FROM db_pvfc9_std.Product_t
 WHERE product_line_id is not null;
```

Output

```
SELECT product_id, CAST(LPAD(CAST(standard_price*100.00 AS DECIMAL(17)), 17, '0') AS CHAR(17)) AS
order_amt
  FROM db_pvfc9_std.Product_t
  WHERE product_line_id is not null;
```

INT

Input - Data type casting for INT

```
SELECT
        CAST( col1 AS INT ) (
            FORMAT '9(5)'
        )
    FROM
        table1;
```

Output

```
SELECT
        LPAD( CAST( col1 AS INT ) ,5 ,'0' )
    FROM
        table1;
```

Input - Data type casting for INT

```
SELECT
        CAST( col1 AS INT ) (
            FORMAT '999999'
        )
    FROM
        table1;
```

Output

```
SELECT
        LPAD( CAST( col1 AS INT ) ,6 ,'0' )
    FROM
        table1;
```

Input - Data type casting for INT

```
SELECT
        CAST( expression1 AS INT FORMAT '9(10)' )
    FROM
        table1;
```

Output

```
SELECT
        LPAD( CAST( expression1 AS INT ) ,10 ,'0' )
    FROM
        table1;
```

Input - Data type casting for INT


```
SELECT
        CAST( expression1 AS INT FORMAT '9999' )
    FROM
        table1;
```

Output

```
SELECT
        LPAD( CAST( expression1 AS INT ) ,4 ,'0' )
    FROM
        table1;
```

DATE

In Teradata, when casting DATE from one format to another format, AS FORMAT is used. Migration tools will add TO_CHAR function to retain the specified input format.

For details, see **Date and Time Functions**.

Input - Data type casting without DATE keyword

```
SELECT
     CAST( CAST( '2013-02-12' AS DATE FORMAT 'YYYY/MM/DD' ) AS FORMAT 'DD/MM/YY' )
;
```

Output

```
SELECT
     TO_CHAR( CAST( '2013-02-12' AS DATE ) ,'DD/MM/YY' )
;
```

DAY to SECOND

Input - Data type casting DAY to SECOND

```
SELECT CAST(T1.Draw_Gold_Dt || ' ' ||T1.Draw_Gold_Tm as Timestamp)
- CAST(T1.Tx_Dt || ' '|| T1.Tx_Tm as Timestamp)  DAY(4) To SECOND  from db_pvfc9_std.draw_tab T1;
```

Output

```
SELECT
        CAST(( CAST( T1.Draw_Gold_Dt || ' ' || T1.Draw_Gold_Tm AS TIMESTAMP ) - CAST(T1.Tx_Dt || ' ' ||
T1.Tx_Tm AS TIMESTAMP ) ) AS INTERVAL DAY ( 4 ) TO SECOND )
    FROM
        db_pvfc9_std.draw_tab T1
;
```

DECIMAL

Input - Data type casting for DECIMAL

```
SELECT
        standard_price (
            DECIMAL( 17 )
            ,FORMAT '9(17)'
        ) (
            CHAR( 17 )
        )
    FROM
        db_pvfc9_std.Product_t;
```

Output

```
SELECT
        CAST( LPAD( CAST( standard_price AS DECIMAL( 17 ,0 ) ) ,17 ,'0' ) AS CHAR( 17 ) )
```



```
    FROM
        db_pvfc9_std.Product_t;
```

Input - Data type casting for DECIMAL

```
SELECT
        standard_price (
            DECIMAL( 17 ,0 )
            ,FORMAT '9(17)'
        ) (
            VARCHAR( 17 )
        )
    FROM
        db_pvfc9_std.Product_t;
```

Output

```
SELECT
        CAST( LPAD( CAST( standard_price AS DECIMAL( 17 ,0 ) ) ,17 ,'0' ) AS VARCHAR( 17 ) )
    FROM
        db_pvfc9_std.Product_t;
```

Input - Data type casting for DECIMAL

```
SELECT
        customer_id (
            DECIMAL( 17 )
        ) (
            FORMAT '9(17)'
        ) (
            VARCHAR( 17 )
        )
    FROM
        db_pvfc9_std.Customer_t;
```

Output

```
SELECT
        CAST( LPAD( CAST( customer_id AS DECIMAL( 17 ,0 ) ) ,17 ,'0' ) AS VARCHAR( 17 ) )
    FROM
        db_pvfc9_std.Customer_t;
```

Time Interval

Type casting to time intervals is supported in DDL and DML. It is supported within SELECT and can be used in subqueries of VIEW, MERGE, and INSERT.

Input - Data type casting to time intervals

```
SELECT TIME '06:00:00.00' HOUR TO SECOND;
```

Output

```
SELECT TIME '06:00:00.00';
```

Input - Data type casting to time intervals with TOP

```
SELECT TOP 3 * FROM dwQErrDtl_mc.C03_CORP_AGENT_INSURE
WHERE Data_Dt > (SELECT TIME '06:00:00.00' HOUR TO SECOND);
```

Output

```
SELECT  * FROM dwQErrDtl_mc.C03_CORP_AGENT_INSURE WHERE  Data_Dt > (SELECT TIME
'06:00:00.00')   limit 3;
```


NULL

DSC will migrate an expression in the form NULL(data_type) to CAST(NULL AS replacement_data_type).

Input - Data type casting for NULL

```
NULL(VARCHAR(n))
```

Output

```
CAST(NULL AS VARCHAR(n))
```

Implicit Type Casting Issues

Input - Implicit TYPE CASTING ISSUES

```
SELECT Data_Type,Start_Dt,End_Dt
 FROM (
  SELECT Data_Type,Start_Dt,End_Dt
  FROM (
   SELECT '101' AS Data_Type,CAST('${TX_DATE}' AS DATE FORMAT 'YYYYMMDD')-1 AS Start_Dt,CAST('$
{TX_DATE}' AS DATE FORMAT 'YYYYMMDD') AS End_Dt
  ) TT
  UNION ALL
  SELECT '201' AS Data_Type,CAST('${TX_DATE}' AS DATE FORMAT 'YYYYMMDD')-7 AS Start_Dt,CAST('$
{TX_DATE}' AS DATE FORMAT 'YYYYMMDD') AS End_Dt
  FROM Sys_Calendar.CALENDAR
  WHERE calendar_date = CAST('${TX_DATE}' AS DATE FORMAT 'YYYYMMDD')
  AND Day_Of_Week = 1
  UNION ALL
  SELECT Data_Type,Start_Dt,End_Dt
  FROM (
   SELECT '401' AS Data_Type,CAST('${TX_PRIMONTH_END}' AS DATE FORMAT 'YYYYMMDD') AS
Start_Dt,CAST('${TX_DATE}' AS DATE FORMAT 'YYYYMMDD') AS End_Dt
   ) TT
  WHERE CAST('${TX_DATE}' AS DATE FORMAT 'YYYYMMDD')=CAST('${TX_MONTH_END}' AS DATE
FORMAT 'YYYYMMDD')
  UNION ALL
  SELECT Data_Type,Start_Dt,End_Dt
  FROM (
   SELECT '501' AS Data_Type,CAST('${TX_PRIQUARTER_END}' AS DATE FORMAT 'YYYYMMDD') AS
Start_Dt,CAST('${TX_DATE}' AS DATE FORMAT 'YYYYMMDD') AS End_Dt
   ) TT
  WHERE CAST('${TX_DATE}' AS DATE FORMAT 'YYYYMMDD')=CAST('${TX_QUARTER_END}' AS DATE
FORMAT 'YYYYMMDD')
  UNION ALL
  SELECT Data_Type,Start_Dt,End_Dt
  FROM (
   SELECT '701' AS Data_Type,CAST('${TX_PRIYEAR_END}' AS DATE FORMAT 'YYYYMMDD') AS
Start_Dt,CAST('${TX_DATE}' AS DATE FORMAT 'YYYYMMDD') AS End_Dt
   ) TT
  WHERE CAST('${TX_DATE}' AS DATE FORMAT 'YYYYMMDD')=CAST('${TX_YEAR_END}' AS DATE FORMAT
'YYYYMMDD')
 ) T1
 ;
```

Output

```
SELECT Data_Type,Start_Dt,End_Dt
 FROM (
  SELECT Data_Type,Start_Dt,End_Dt
  FROM (
   SELECT CAST('101' AS TEXT) AS Data_Type,CAST('${TX_DATE}' AS DATE FORMAT 'YYYYMMDD')-1 AS
Start_Dt,CAST('${TX_DATE}' AS DATE FORMAT 'YYYYMMDD') AS End_Dt
  ) TT
  UNION ALL
  SELECT CAST('201' AS TEXT) AS Data_Type,CAST('${TX_DATE}' AS DATE FORMAT 'YYYYMMDD')-7 AS
```



```
Start_Dt,CAST('${TX_DATE}' AS DATE FORMAT 'YYYYMMDD') AS End_Dt
  FROM Sys_Calendar.CALENDAR
  WHERE calendar_date = CAST('${TX_DATE}' AS DATE FORMAT 'YYYYMMDD')
  AND Day_Of_Week = 1
  UNION ALL
  SELECT Data_Type,Start_Dt,End_Dt
  FROM (
   SELECT CAST('401' AS TEXT) AS Data_Type,CAST('${TX_PRIMONTH_END}' AS DATE FORMAT
'YYYYMMDD') AS Start_Dt,CAST('${TX_DATE}' AS DATE FORMAT 'YYYYMMDD') AS End_Dt
   ) TT
  WHERE CAST('${TX_DATE}' AS DATE FORMAT 'YYYYMMDD')=CAST('${TX_MONTH_END}' AS DATE
FORMAT 'YYYYMMDD')
  UNION ALL
  SELECT Data_Type,Start_Dt,End_Dt
  FROM (
   SELECT CAST('501' AS TEXT) AS Data_Type,CAST('${TX_PRIQUARTER_END}' AS DATE FORMAT
'YYYYMMDD') AS Start_Dt,CAST('${TX_DATE}' AS DATE FORMAT 'YYYYMMDD') AS End_Dt
   ) TT
  WHERE CAST('${TX_DATE}' AS DATE FORMAT 'YYYYMMDD')=CAST('${TX_QUARTER_END}' AS DATE
FORMAT 'YYYYMMDD')
  UNION ALL
  SELECT Data_Type,Start_Dt,End_Dt
  FROM (
   SELECT CAST('701' AS TEXT) AS Data_Type,CAST('${TX_PRIYEAR_END}' AS DATE FORMAT 'YYYYMMDD')
AS Start_Dt,CAST('${TX_DATE}' AS DATE FORMAT 'YYYYMMDD') AS End_Dt
   ) TT
  WHERE CAST('${TX_DATE}' AS DATE FORMAT 'YYYYMMDD')=CAST('${TX_YEAR_END}' AS DATE FORMAT
'YYYYMMDD')
 ) T1
 ;
```

Hexadecimal Character Literals

Input	Output
'CASE WHEN Nullable='Y' THEN '' ELSE ' NOT NULL' END || '0A'XC	CASE WHEN Nullable='Y' THEN '' ELSE ' NOT NULL' END || E'\x0A'

Hexadecimal Character literal value

Input	Output

'SELECT CASE WHEN Nullable='Y' THEN ''
ELSE NOT NULL
END || '0A'XC AS SP_DATA_DT
FROM tbl_table;
.IF ERRORCODE <> 0 THEN .QUIT 12
	DECLARE lv_mig_errorcode NUMBER (4) ;
	lv_mig_SP_DATA_DT TEXT ;
	BEGIN
	BEGIN
	SELECT
	STRING_AGG (
	CASE
	WHEN Nullable = 'Y' THEN ''
	ELSE NOT NULL
	END || E'\x0A' /* ????????? */
	,''
	) INTO
	lv_mig_SP_DATA_DT
	FROM
	tbl_table ;
	lv_mig_errorcode := 0 ;
	EXCEPTION
	WHEN OTHERS THEN lv_mig_errorcode := - 1 ;
	END ;
	IF lv_mig_errorcode <> 0 THEN RAISE EXCEPTION
	'12' ;
	END IF ;
	END ;
	/

TRIM (Including INT Type Casting)

Input	Output
TRIM(columnlength (INT))	TRIM(mig_td_ext.mig_fn_castasint(columnlength))

6.4.17 BTEQ Utility Command

GaussDB(DWS) provides gsql meta-commands that can be used to replace common BTEQ tool commands. The mostly used replacements are as follows:

.QUIT | .EXIT | .RETURN

The meta-command **\q [***value***]** can be used to exit the gsql program, and **value** specifies the exit code. The .QUIT, .EXIT, and .RETURN commands can replace each other with the **\q** command.

Input	Output
.QUIT 0	\q 0
.EXIT	\q
.RETURN	\q

.LABEL and .GOTO

The Teradata command .LABEL is used to create tags and is usually used together with .GOTO. .GOTO skips all intermediate BTEQ commands and SQL statements, instructs you to reach the specified label position, and performs corresponding restoration processing.

gsql meta-command **\goto LABEL...** and **\label LABEL** can be replaced with each other with no constraints.

Input	Output
.IF CHECK_PK='' THEN .GOTO NOCHECK ${CHECK_PK}; .LABEL NOCHECK .QUIT 0	\if ${CHECK_PK} == '' \goto NOCHECK \endif ${CHECK_PK} \label NOCHECK \q 0

.RUN FILE

Executing the SQL request of a specified file can be implemented by the gsql meta-command **\i**.

Input	Output
.RUN FILE=sampfile	\i 'sampfile'

.EXPORT FILE

Exporting the SQL statement execution result to a specified file can be implemented by the gsql meta-command **\o**.

Input	Output
.EXPORT REPORT FILE=resultfile	\o 'resultfile'

..SET ..END

To set a variable to specified value, both a single-line command or a multi-line command ended with ..END can be used. You can run the SELECT command to query a specified variable name or run the **\set** command to convert the variable name.

Input	Output
.SET SEPARATOR ' | '	\set SEPARATOR '|'
..SET NAME 'Jack' ..END	SELECT 'Jack' AS "NAME";

.IF

.IF is one of the important process control commands used to execute parts of an input script. It can be a single-branch command, which is used in pairs with the THEN clause. It can also be a multi-branch command. The multi-branch IF structure allows multiple nested layers, but each layer must start with the IF command and end with the ENDIF command.

The flow control meta-commands **\if...** \else ... **\endif** can be used to replace this BTEQ command.

Input	Output
.IF ERRORCODE <> 0 THEN .QUIT 12;	\if ${ERROR} != 'false' \q 12 \endif

..FOR

In the loop control command, when the loop condition is met, the script of the loop body can be executed continuously until .QUIT exits the loop. GaussDB(DWS) provides **\for...** \loop ... \exit-for ... **\end-for** structure command implements the loop logic.

Input	Output
..IF ACTIVITYCOUNT > 0 THEN ..FOR SEL SqlStr AS V_SqlStr FROM ${ ETL_DATA}.TB_DWDATA_UPDATE WHERE JobName = '${JOB_NAME}' AND TXDATE = ${ TX_DATE} - 19000000 ORDER BY ExcuteSeq ASC; ..DO ${V_SqlStr} .IF ERRORCODE<>0 THEN .QUIT 12 ..END-FOR ..END-IF	\if ${ERROR} != 'false' \q 12 \endif \if ${ACTIVITYCOUNT} != 0 \for SELECT SqlStr AS V_SqlStr FROM ${ETL_DATA}.TB_DWDATA_UPDATE WHERE JobName = '${JOB_NAME}' AND TXDATE = to_date($ {TX_DATE} ,'yyyymmdd') ORDER BY ExcuteSeq ASC \loop ${V_SqlStr} ; \if ${ERROR} != 'false' \q 12 \endif \end-for \endif

6.4.18 Teradata Formats

Date in the YYYYMMDD Format

Input	Output
SELECT 1 FROM tb_dt_fmtyyyymmdd WHERE JobName ='${JOB_NAME}' AND TXDATE = ${TX_DATE} - 19000000;	SELECT 1 FROM tb_dt_fmtyyyymmdd WHERE JobName ='${JOB_NAME}' AND TXDATE = TO_DATE(${TX_DATE}, 'YYYYMMDD');

Date in the YYYYDDD Format

Input	Output
REPLACE VIEW SC.VIEW_1 (col_1) LOCKING TABLE sc.tab FOR ACCESS AS SEL --tgt.col_1 is date type CAST(CAST(TGT.col_1 AS DATE FORMAT 'YYYYDDD') AS CHAR(7)) AS col_1 FROM sc.tab TGT ;	CREATE OR REPLACE VIEW SC.VIEW_1 (col_1) /*LOCKING TABLE sc.tab FOR ACCESS */ AS (SELECT /* tgt.col_1 is date type */ CAST(TO_DATE(TGT.col_1, 'YYYYDDD') AS CHAR(7)) AS col_1 FROM sc.tab TGT) ;

Column Names Starting with

Input	Output
REPLACE VIEW SC.VIEW_1 (,col_1 ,#_col_2 ,#_col_3) LOCKING TABLE sc.tab FOR ACCESS AS SEL Tgt.col1 ,Tgt.#_col_2 ,Tgt.#_col_3 FROM sc.tab TGT ;	CREATE OR REPLACE VIEW SC.VIEW_1 (,col_1 ,"#_COL_2" ,"#_COL_3") /*LOCKING TABLE sc.tab FOR ACCESS */ AS (SELECT Tgt.col1 ,Tgt."#_COL_2" ,Tgt."#_COL_3" FROM sc.tab TGT) ;

Database Operations First During Type Casting

Input	Output
REPLACE VIEW SC.VIEW_1 (col_1) LOCKING TABLE sc.tab FOR ACCESS AS SEL (COALESCE(TRIM(TGT.col_1),'')) || '_' || (COALESCE(TRIM(TGT.col_1),'')) (CHAR(22)) AS col_1 FROM sc.tab TGT ;	CREATE OR REPLACE VIEW SC.VIEW_1 (col_1) /*LOCKING TABLE sc.tab FOR ACCESS */ AS (SELECT CAST((COALESCE(TRIM(TGT.col_1) ,'')) || '_' || (COALESCE(TRIM(TGT.col_1) ,'')) AS CHAR(22)) AS col_1 FROM sc.tab TGT) ;

6.4.19 System Views

DSC migrates the system views **dbc.columnsV** and **dbc.IndicesV**, and the resulting output is displayed below.

Input:

```
SELECT A.ColumnName
AS V_COLS
     ,A.columnname  || ' ' ||CASE WHEN columnType in ('CF','CV')
                                  THEN CASE WHEN columnType='CV' THEN 'VAR' ELSE ''
            END||'CHAR('||TRIM(columnlength (INT))||
                ') CHARACTER SET LATIN'||
                  CASE WHEN UpperCaseFlag='N'
                     THEN ' NOT' ELSE ''
                  END || ' CASESPECIFIC'
                              WHEN columnType='DA' THEN 'DATE'
                              WHEN columnType='TS' THEN 'TIMESTAMP(' || TRIM(DecimalFractionalDigits)||')'
                              WHEN columnType='AT' THEN 'TIME('|| TRIM(DecimalFractionalDigits)||')'
                              WHEN columnType='I' THEN 'INTEGER'
                              WHEN columnType='I1' THEN 'BYTEINT'
                              WHEN columnType='I2' THEN 'SMALLINT'
                              WHEN columnType='I8' THEN 'BIGINT'
                              WHEN columnType='D' THEN 'DECIMAL('||TRIM(DecimalTotalDigits)||','||
TRIM(DecimalFractionalDigits)||')'
                              ELSE 'Unknown'
                         END||CASE WHEN Nullable='Y'
       THEN '' ELSE ' NOT NULL' END||'0A'XC
AS V_ColT           -      ,B.ColumnName
AS V_PICol
 FROM dbc.columnsV A LEFT JOIN dbc.IndicesV B
   ON A.columnName = B.columnName AND B.IndexType IN ('Q','P')
  AND B.DatabaseName = '${V_TDDLDB}'  AND B.tablename='${TARGET_TABLE}'
WHERE A.databasename='${V_TDDLDB}' AND A.tablename = '${TARGET_TABLE}'
  AND A.columnname NOT IN
( 'ETL_JOB_NAME'                                                                                ,'ETL_TX_DATE'
                        ,'ETL_PROC_DATE'
                        )
 ORDER BY A.columnid;
```

Output:

```
DECLARE lv_mig_V_COLS   TEXT;
        lv_mig_V_ColT       TEXT;
        lv_mig_V_PICol      TEXT;
BEGIN
SELECT STRING_AGG(A.ColumnName, ',')
```



```
    , STRING_AGG(A.columnname  || ' ' ||CASE WHEN columnType in ('CF','CV')
                                  THEN CASE WHEN columnType='CV' THEN 'VAR' ELSE ''
           END||'CHAR('||TRIM(mig_td_ext.mig_fn_castasint(columnlength))||
              ') /*CHARACTER SET LATIN*/'||
                CASE WHEN UpperCaseFlag='N'
                  THEN ' NOT' ELSE ''
                END || ' /*CASESPECIFIC*/'
                            WHEN columnType='DA' THEN 'DATE'
                            WHEN columnType='TS' THEN 'TIMESTAMP(' || TRIM(DecimalFractionalDigits)||')'
                            WHEN columnType='AT' THEN 'TIME('|| TRIM(DecimalFractionalDigits)||')'
                            WHEN columnType='I' THEN 'INTEGER'
                            WHEN columnType='I1' THEN 'BYTEINT'
                            WHEN columnType='I2' THEN 'SMALLINT'
                            WHEN columnType='I8' THEN 'BIGINT'
                            WHEN columnType='D' THEN 'DECIMAL('||TRIM(DecimalTotalDigits)||','||
TRIM(DecimalFractionalDigits)||')'
                            ELSE 'Unknown'
                          END||CASE WHEN Nullable='Y'
       THEN '' ELSE ' NOT NULL' END||E'\x0A', ',')                , STRING_AGG(B.ColumnName, ',')
INTO lv_mig_V_COLS, lv_mig_V_ColT, lv_mig_V_PICol
FROM mig_td_ext.vw_td_dbc_columnsV A LEFT JOIN mig_td_ext.vw_td_dbc_IndicesV B
  ON A.columnName = B.columnName AND B.IndexType IN ('Q','P')
 AND B.DatabaseName = 'public'  AND B.tablename='emp2'
WHERE A.databasename='public' AND A.tablename = 'emp2';
-- ORDER BY A.columnid;
END;
/
```

6.5 MySQL Syntax Migrating

6.5.1 Supported Keywords and Features

Table 6-25 lists the MySQL keywords and features that can be migrated.

- The **Version** column contains the earliest versions that support the keywords or features.
- The **Remarks** column contains the configuration parameters to customize how the migration tool migrates the corresponding keywords or features.

Table 6-25

Section	Object->Keyword/ Feature	Version	Remarks
Data Types	Numeric Types	8.0.0	You can specify the month when the coupon will expire. table.orientation table.type table.compress.mode table.compress.row table.compress.column table.compress.level
	Date/Time Types	8.0.0	
	String Types	8.0.0	
	Spatial Data Types	8.0.0	
	LOB Types	8.0.0	
	Set Types	8.0.0	
	Boolean	8.0.0	
	Binary Types	8.0.0	

Section	Object->Keyword/ Feature	Version	Remarks
	JSON	8.0.0	
Functions and Expressions	Type Mapping	8.0.0	-

Section	Object->Keyword/ Feature		Version	Remarks
Table (Optional Parameters and Operations)	Table (Optional Parameters and Operations)	• AUTO_INCREMENT • AVG_ROW_LENGTH • CHARSET • CHECKSUM • COLLATE • COMMENT • CONNECTION • DELAY_KEY_WRITE • DIRECTORY • ENGINE • KEY_BLOCK_SIZE • INSERT_METHOD • MAX_ROWS • MIN_ROWS • PACK_KEYS • PASSWORD • ROW_FORMAT • STATS_AUTO_RECALC • STATS_PERSISTENT • STATS_SAMPLE_PAGES • UNION • LIKE Table Cloning	8.0.0 8.0.0	- -

Section	Object->Keyword/ Feature		Version	Remarks
		• Adding and Deleting a Column • MODIFY: Modifying a Column • CHANGE: Changing a Column • SET|DROP COLUMN DEFAULT VALUE • DROP (Table Deletion) • TRUNCATE (Table Deletion) • LOCK • RENAME (Table Renaming)		
	Indexes	• Unique Indexes • Normal and Prefix Indexes • Hash index • B-tree Indexes • Spatial Indexes • Full-Text Indexes • Deleting an Index • Renaming an Index	8.0.0 8.0.0 8.0.0 8.0.0 8.0.0 8.0.0 8.0.0 8.0.0	-
	Comment		8.0.0	-

Section	Object->Keyword/ Feature		Version	Remarks
	Databases		8.0.0	-
Data Manipulation Language (DML)	SELECT	• Quotation Marks • INTERVAL • Division Expressions • Group By Conversion • ROLLUP	8.0.0	-
			8.0.0	-
			8.0.0	-
			8.0.0	-
	INSERT	• HIGH_PRIORITY • LOW_PRIORITY • PARTITION • DELAYED • IGNORE • VALUES • ON DUPLICATE KEY UPDATE • SET	8.0.0 8.0.0 8.0.0 8.0.0 8.0.0 8.0.0 8.0.0 8.0.0	-
	UPDATE	• LOW_PRIORITY • ORDER BY • LIMIT • IGNORE	8.0.0 8.0.0 8.0.0 8.0.0	-
	REPLACE	• LOW_PRIORITY • PARTITION • DELAYED • VALUES • SET	8.0.0 8.0.0 8.0.0 8.0.0 8.0.0 8.0.0	-

Section	Object->Keyword/ Feature		Version	Remarks
Transaction Management and Database Management	Transaction Management	• TRANSACTION • LOCK	8.0.0 8.0.0	-
	Database Management	• SET CHARACTER	8.0.0	-

6.5.2 Data Types

6.5.2.1 Numeric Types

Overview

A data type is a basic data attribute. Occupied storage space and allowed operations vary according to data types. In a database, data is stored in tables, in which a data type is specified for each column. Data in the column must be of its allowed data type. The following table lists an example of converting MySQL numeric types to GaussDB(DWS) numeric types.

Type Comparison

Table 6-26 Numeric type mapping

MySQL Numeric Type	MySQL INPUT	GaussDB(DWS) Output
DEC	DEC DEC[(M[,D])] [UNSIGNED] [ZEROFILL]	DECIMAL DECIMAL[(M[,D])]
DECIMAL	DECIMAL[(M[,D])] [UNSIGNED] [ZEROFILL]	DECIMAL[(M[,D])]
DOUBLE PRECISION	DOUBLE PRECISION DOUBLE PRECISION [(M[,D])] [UNSIGNED] [ZEROFILL]	DOUBLE PRECISION DOUBLE PRECISION
DOUBLE	DOUBLE[(M[,D])] [UNSIGNED] [ZEROFILL]	DOUBLE PRECISION
FIXED	FIXED FIXED[(M[,D])] [UNSIGNED] [ZEROFILL]	DECIMAL DECIMAL[(M[,D])]

MySQL Numeric Type	MySQL INPUT	GaussDB(DWS) Output
FLOAT	FLOAT FLOAT [(M[,D])] [UNSIGNED] [ZEROFILL] FLOAT(p) [UNSIGNED] [ZEROFILL]	REAL REAL REAL
INT	INT INT(p) [UNSIGNED] [ZEROFILL]	INTEGER INTEGER(p)
INTEGER	INTEGER INTEGER(p) [UNSIGNED] [ZEROFILL]	INTEGER INTEGER(p)
MEDIUMINT	MEDIUMINT MEDIUMINT(p) [UNSIGNED] [ZEROFILL]	INTEGER INTEGER(p)
NUMERIC	NUMERIC NUMERIC [(M[,D])] [UNSIGNED] [ZEROFILL]	DECIMAL DECIMAL[(M[,D])]
REAL	REAL[(M[,D])]	REAL/DOUBLE PRECISION
SMALLINT	SMALLINT SMALLINT(p) [UNSIGNED] [ZEROFILL]	SMALLINT
TINYINT	TINYINT TINYINT(n) TINYINT(n) ZEROFILL TINYINT(n) UNSIGNED ZEROFILL	SMALLINT SMALLINT SMALLINT TINYINT

NOTE

- When the TINYINT type is converted, if it is unsigned (UNSIGNED), it is converted to TINYINT. Otherwise, it is converted to SMALLINT.
- The REAL type is converted to DOUBLE PRECISION by default. If **table.database.realAsFlag** in the **features-mysql.properties** configuration file is set to **true** (**false** by default), the type is converted to REAL.

Input: TINYINT

```
CREATE TABLE IF NOT EXISTS `runoob_dataType_test`(
   `dataType_1` TINYINT,
   `dataType_2` TINYINT(0),
   `dataType_3` TINYINT(255),
   `dataType_4` TINYINT(255) UNSIGNED ZEROFILL,
```



```
  `dataType_5` TINYINT(255) ZEROFILL
);
```

Output

```
CREATE TABLE IF NOT EXISTS "public"."runoob_datatype_test"
(
  "datatype_1" SMALLINT,
  "datatype_2" SMALLINT,
  "datatype_3" SMALLINT,
  "datatype_4" TINYINT,
  "datatype_5" SMALLINT
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_1");
```

6.5.2.2 Date/Time Types

Overview

This section describes the following date and time types: DATETIME, TIME, TIMESTAMP, and YEAR. GaussDB(DWS) does not support these types, and DSC will convert them.

Type Mapping

Table 6-27 Date/Time type mapping

MySQL Date/ Time Type	MySQL INPUT	GaussDB(DWS) OUTPUT
DATETIME	DATETIME[(fsp)]	TIMESTAMP[(fsp)] WITHOUT TIME ZONE
TIME	TIME[(fsp)]	TIME[(fsp)] WITHOUT TIME ZONE
TIMESTAMP	TIMESTAMP[(fsp)]	TIMESTAMP[(fsp)] WITH TIME ZONE
YEAR	YEAR[(4)]	SMALLINT(4)

NOTE

The value of *fsp* must be in the range [0, 6]. Value **0** indicates no decimal. If this parameter is omitted, the default precision will be 0.

Input: DATETIME

```
CREATE TABLE IF NOT EXISTS `runoob_dataType_test`(
  `dataType_1` DATETIME,
  `dataType_2` DATETIME(0),
  `dataType_3` DATETIME(6),
  `dataType_4` DATETIME DEFAULT NULL,
```



```
    `dataType_5` DATETIME DEFAULT '2018-10-12 15:27:33.999999'
);
```

Output

```
CREATE TABLE IF NOT EXISTS "public"."runoob_datatype_test"
(
  "datatype_1" TIMESTAMP WITHOUT TIME ZONE,
  "datatype_2" TIMESTAMP(0) WITHOUT TIME ZONE,
  "datatype_3" TIMESTAMP(6) WITHOUT TIME ZONE,
  "datatype_4" TIMESTAMP WITHOUT TIME ZONE DEFAULT NULL,
  "datatype_5" TIMESTAMP WITHOUT TIME ZONE DEFAULT '2018-10-12 15:27:33.999999'
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_1");
```

Input: TIME

```
CREATE TABLE IF NOT EXISTS `runoob_dataType_test`(
    `dataType_1` TIME DEFAULT '20:58:10',
    `dataType_2` TIME(3) DEFAULT '20:58:10',
    `dataType_3` TIME(6) DEFAULT '20:58:10',
    `dataType_4` TIME(6) DEFAULT '2018-10-11 20:00:00',
    `dataType_5` TIME(6) DEFAULT '20:58:10.01234',
    `dataType_6` TIME(6) DEFAULT '2018-10-11 20:00:00.01234',
    `dataType_7` TIME DEFAULT NULL,
    `dataType_8` TIME(6) DEFAULT NULL,
    PRIMARY KEY (dataType_1)
);
```

Output

```
CREATE TABLE IF NOT EXISTS "public"."runoob_datatype_test"
(
  "datatype_1" TIME WITHOUT TIME ZONE DEFAULT '20:58:10',
  "datatype_2" TIME(3) WITHOUT TIME ZONE DEFAULT '20:58:10',
  "datatype_3" TIME(6) WITHOUT TIME ZONE DEFAULT '20:58:10',
  "datatype_4" TIME(6) WITHOUT TIME ZONE DEFAULT '2018-10-11 20:00:00',
  "datatype_5" TIME(6) WITHOUT TIME ZONE DEFAULT '20:58:10.01234',
  "datatype_6" TIME(6) WITHOUT TIME ZONE DEFAULT '2018-10-11 20:00:00.01234',
  "datatype_7" TIME WITHOUT TIME ZONE DEFAULT NULL,
  "datatype_8" TIME(6) WITHOUT TIME ZONE DEFAULT NULL,
  PRIMARY KEY ("datatype_1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_1");
```

Input: TIMESTAMP

```
CREATE TABLE IF NOT EXISTS `runoob_dataType_test`(
    `dataType_1` TIMESTAMP,
    `dateType_4` TIMESTAMP DEFAULT '2018-10-12 15:27:33',
    `dateType_5` TIMESTAMP DEFAULT '2018-10-12 15:27:33.999999',
    `dateType_6` TIMESTAMP DEFAULT '2018-10-12 15:27:33',
    `dateType_7` TIMESTAMP DEFAULT '2018-10-12 15:27:33',
    `dataType_8` TIMESTAMP(0) DEFAULT '2018-10-12 15:27:33',
    `dateType_9` TIMESTAMP(6) DEFAULT '2018-10-12 15:27:33'
);
```

Output

```
CREATE TABLE IF NOT EXISTS "public"."runoob_datatype_test"
(
  "datatype_1" TIMESTAMP WITH TIME ZONE,
  "datetype_4" TIMESTAMP WITH TIME ZONE DEFAULT '2018-10-12 15:27:33',
  "datetype_5" TIMESTAMP WITH TIME ZONE DEFAULT '2018-10-12 15:27:33.999999',
  "datetype_6" TIMESTAMP WITH TIME ZONE DEFAULT '2018-10-12 15:27:33',
  "datetype_7" TIMESTAMP WITH TIME ZONE DEFAULT '2018-10-12 15:27:33',
```



```
  "datatype_8" TIMESTAMP(0) WITH TIME ZONE DEFAULT '2018-10-12 15:27:33',
  "datetype_9" TIMESTAMP(6) WITH TIME ZONE DEFAULT '2018-10-12 15:27:33'
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_1");
```

Input: YEAR

```
CREATE TABLE IF NOT EXISTS `runoob_dataType_test`(
    `dataType_1` YEAR,
    `dataType_2` YEAR(4),
    `dataType_3` YEAR DEFAULT '2018',
    `dataType_4` TIME DEFAULT NULL
);
```

Output

```
CREATE TABLE IF NOT EXISTS "public"."runoob_datatype_test"
(
  "datatype_1" SMALLINT,
  "datatype_2" SMALLINT,
  "datatype_3" VARCHAR(4) DEFAULT '2018',
  "datatype_4" TIME WITHOUT TIME ZONE DEFAULT NULL
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_1");
```

6.5.2.3 String Types

Overview

MySQL interprets length specifications in character column definitions in character units. This applies to the CHAR, VARCHAR, and TEXT types. DSC supports the following type conversions:

Type Mapping

Table 6-28 String type mapping

MySQL String Type	MySQL INPUT	GaussDB(DWS) OUTPUT
CHAR	CHAR[(0)] CHAR[(n)]	CHAR[(1)] CHAR[(4n)]
CHARACTER	CHARACTER[(0)] CHARACTER[(n)]	CHAR[(1)] CHAR[(4n)]
NCHAR	NCHAR[(0)] NCHAR[(n)]	CHAR[(1)] CHAR[(4n)]
LONGTEXT	LONGTEXT	TEXT
MEDIUMTEXT	MEDIUMTEXT	TEXT
TEXT	TEXT	TEXT
TINYTEXT	TINYTEXT	TEXT

MySQL String Type	MySQL INPUT	GaussDB(DWS) OUTPUT
VARCHAR	VARCHAR[(0)] VARCHAR[(n)]	VARCHAR[(1)] VARCHAR[(4n)]
NVARCHAR	NVARCHAR[(0)] NVARCHAR[(n)]	VARCHAR[(1)] VARCHAR[(4n)]
CHARACTE VARYING	CHARACTE VARYING	VARCHAR

NOTE

- During CHAR/CHARACTER/NCHAR conversion, if the precision is less than or equal to 0, it is converted to CHAR(1). If the precision is greater than 0, it is converted to a precision level four times that of the CHAR type.
- During VARCHAR/NVARCHAR conversion, if the precision is less than or equal to 0, it is converted to VARCHAR(1). If the precision is greater than 0, it is converted to a precision four times that of the VARCHAR type.

Input: CHAR

In MySQL, the size of a CHAR column is set to a fixed length specified at table creation, ranging from 0 to 255. Stored CHAR values are right-padded with spaces to meet this specified length.

```
CREATE TABLE IF NOT EXISTS `runoob_dataType_test`(
   `dataType_1` CHAR NOT NULL,
   `dataType_2` CHAR(0) NOT NULL,
   `dataType_3` CHAR(255) NOT NULL
);
```

Output

```
CREATE TABLE IF NOT EXISTS "public"."runoob_datatype_test"
(
  "datatype_1" CHAR NOT NULL,
  "datatype_2" CHAR(1) NOT NULL,
  "datatype_3" CHAR(1020) NOT NULL
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_1");
```

Input: [LONG|MEDIUM|TINY]TEXT

```
CREATE TABLE IF NOT EXISTS `runoob_dataType_test`(
    `dataType_1` LONGTEXT,
    `dataType_2` MEDIUMTEXT,
    `dataType_3` TEXT,
    `dataType_4` TINYTEXT
);
```

Output

```
CREATE TABLE IF NOT EXISTS "public"."runoob_datatype_test"
(
  "datatype_1" TEXT,
  "datatype_2" TEXT,
  "datatype_3" TEXT,
```



```
  "datatype_4" TEXT
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_1");
```

Input: VARCHAR

In MySQL, values in VARCHAR columns are variable-length strings. The length can be any value from 0 to 65,535.

```
CREATE TABLE IF NOT EXISTS `runoob_dataType_test`(
   `dataType_1` VARCHAR(0),
   `dataType_2` VARCHAR(1845)
);
```

Output

```
CREATE TABLE IF NOT EXISTS "public"."runoob_datatype_test"
(
  "datatype_1" VARCHAR(1),
  "datatype_2" VARCHAR(7380)
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_1");
```

6.5.2.4 Spatial Data Types

Overview

MySQL has spatial data types corresponding to the OpenGIS class. DSC supports the following type conversions:

Type Mapping

Table 6-29 Spatial type mapping

MySQL Spatial Type	MySQL INPUT	GaussDB(DWS) OUTPUT
GEOMETRY	GEOMETRY	GEOMETRY
POINT	POINT	POINT
LINESTRING	LINESTRING	POLYGON
POLYGON	POLYGON	POLYGON
MULTIPOINT	MULTIPOINT	BOX
MULTILINESTRING	MULTILINESTRING	BOX
MULTIPOLYGON	MULTIPOLYGON	POLYGON
GEOMETRYCOLLECTION	GEOMETRYCOLLECTION	GEOMETRYCOLLECTION

NOTE

- GEOMETRY can store geometry values of any type. The other single-value types (POINT, LINESTRING, and POLYGON) restrict their values to a particular geometry type.
- GEOMETRYCOLLECTION can store a collection of objects of any type. The other collection types (**MULTIPOINT**, **MULTILINESTRING**, **MULTIPOLYGON**, and **GEOMETRYCOLLECTION**) restrict collection members to those having a particular geometry type.

- **Input**

```
CREATE TABLE `t_geo_test2`  (
  `id` int(11) NOT NULL,
  `name` varchar(255),
  `geometry_1` geometry NOT NULL,
  `point_1` point NOT NULL,
  `linestring_1` linestring NOT NULL,
  `polygon_1` polygon NOT NULL,
  `multipoint_1` multipoint NOT NULL,
  `multilinestring_1` multilinestring NOT NULL,
  `multipolygon_1` multipolygon NOT NULL,
  `geometrycollection_1` geometrycollection NOT NULL,
  PRIMARY KEY (`id`) USING BTREE
) ENGINE = InnoDB;
```

Output

```
CREATE TABLE "public"."t_geo_test2"
(
  "id" INTEGER(11) NOT NULL,
  "name" VARCHAR(255),
  "geometry_1" GEOMETRY NOT NULL,
  "point_1" POINT NOT NULL,
  "linestring_1" POLYGON NOT NULL,
  "polygon_1" POLYGON NOT NULL,
  "multipoint_1" BOX NOT NULL,
  "multilinestring_1" BOX NOT NULL,
  "multipolygon_1" POLYGON NOT NULL,
  "geometrycollection_1" GEOMETRYCOLLECTION NOT NULL,
  PRIMARY KEY ("id")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("id");
```

6.5.2.5 LOB Types

Overview

A BLOB is a binary large object that can hold a variable amount of data. The four BLOB types are TINYBLOB, BLOB, MEDIUMBLOB, and LONGBLOB. The only difference between these four types is the maximum length of the values they can contain. DSC supports the following type conversions:

NOTE

The BLOB type can store images. Column storage does not support BLOB.

Type Mapping

Table 6-30 LOB type mapping

MySQL LOB Type	MySQL INPUT	GaussDB(DWS) OUTPUT
TINYBLOB	TINYBLOB	BLOB
BLOB	BLOB	BLOB
MEDIUMBLOB	MEDIUMBLOB	BLOB
LONGBLOB	LONGBLOB	BLOB

Input: [TINY|MEDIUM|LONG]BLOB

```
CREATE TABLE IF NOT EXISTS `runoob_dataType_test`(
   `dataType_1` BIGINT,
   `dataType_2` TINYBLOB,
   `dataType_3` BLOB,
   `dataType_4` MEDIUMBLOB,
   `dataType_5` LONGBLOB
);
```

Output

```
CREATE TABLE IF NOT EXISTS "public"."runoob_datatype_test"
(
  "datatype_1" BIGINT,
  "datatype_2" BLOB,
  "datatype_3" BLOB,
  "datatype_4" BLOB,
  "datatype_5" BLOB
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_1");
```

6.5.2.6 Set Types

Overview

1. In MySQL, an ENUM is a string object with a value chosen from a list of permitted values that are enumerated explicitly in the column specification at table creation.
2. A SET is a string object that can have zero or more values, each of which must be chosen from a list of permitted values specified when the table is created.

Type Mapping

Table 6-31 Set type mapping

MySQL Set Type	MySQL INPUT	GaussDB(DWS) OUTPUT
ENUM	ENUM	VARCHAR
SET	SET	VARCHAR

NOTE

- The ENUM type is transformed into the VARCHAR type, with a precision quadruple the length of the longest enumeration. The CHECK() function validates the accuracy of the entered values.
- The SET type is converted to the VARCHAR type. The precision is four times the sum of the length of each enumerated value field and the number of separators.

Input: ENUM

```
CREATE TABLE IF NOT EXISTS `runoob_dataType_test`(
    id   int(2) PRIMARY KEY,
   `dataType_17` ENUM('dws-1', 'dws-2', 'dws-3')
);
```

Output

```
CREATE TABLE IF NOT EXISTS "public"."runoob_datatype_test"
(
  "id" INTEGER(2) PRIMARY KEY,
  "datatype_17" VARCHAR(20) CHECK (dataType_17 IN('dws-1','dws-2','dws-3','', null))
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("id");
```

Input: SET

```
CREATE TABLE IF NOT EXISTS `runoob_tbl_test`(
   `dataType_18` SET('dws-1', 'dws-2', 'dws-3')
);
```

Output

```
CREATE TABLE IF NOT EXISTS "public"."runoob_tbl_test"
(
  "datatype_18" VARCHAR(68)
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_18");
```

6.5.2.7 Boolean

Overview

MySQL supports both BOOL and BOOLEAN. DSC supports the following type conversions:

Type Mapping

Input: BOOL/BOOLEAN

```
CREATE TABLE IF NOT EXISTS `runoob_dataType_test`(
   `dataType_1` INT,
   `dataType_2` BOOL,
   `dataType_3` BOOLEAN
);
```

Output

```
CREATE TABLE IF NOT EXISTS "public"."runoob_datatype_test"
(
  "datatype_1" INTEGER,
  "datatype_2" BOOLEAN,
  "datatype_3" BOOLEAN
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_1");
```

6.5.2.8 Binary Types

Overview

- In MySQL, the BIT data type is used to store bit values, ranging from 1 to 64.
- MySQL BINARY and VARBINARY types are similar to CHAR and VARCHAR, except that they contain binary strings rather than non-binary strings.

Type Mapping

Table 6-32 Binary type mapping

MySQL Binary Type	MySQL INPUT	GaussDB(DWS) OUTPUT
BIT[(M)]	BIT[(M)]	BIT[(M)]
BINARY[(M)]	BINARY[(M)]	BYTEA
CHAR BYTE[(M)]	BINARY[(M)]	BYTEA
VARBINARY[(M)]	VARBINARY[(M)]	BYTEA

Input: BIT

```
CREATE TABLE IF NOT EXISTS `runoob_dataType_test`(
   `dataType_1` INT,
   `dataType_2` BIT(1),
   `dataType_3` BIT(64)
);
```

Output

```
CREATE TABLE IF NOT EXISTS "public"."runoob_datatype_test"
(
  "datatype_1" INTEGER,
  "datatype_2" BIT(1),
```



```
  "datatype_3" BIT(64)
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_1");
```

Input: [VAR]BINARY

```
CREATE TABLE IF NOT EXISTS `runoob_dataType_test`(
    `dataType_1` INT,
    `dataType_2` BINARY,
    `dataType_3` BINARY(0),
    `dataType_4` BINARY(255),
    `dataType_5` VARBINARY(0),
    `dataType_6` VARBINARY(6553)
);
```

Output

```
CREATE TABLE IF NOT EXISTS "public"."runoob_datatype_test"
(
  "datatype_1" INTEGER,
  "datatype_2" BYTEA,
  "datatype_3" BYTEA,
  "datatype_4" BYTEA,
  "datatype_5" BYTEA,
  "datatype_6" BYTEA
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_1");
```

6.5.2.9 JSON Types

Overview

The JSON data type can be used to store JavaScript Object Notation (JSON) data. DSC supports the following type conversions:

Type Mapping

Input Example (JSON)

```
CREATE TABLE IF NOT EXISTS `runoob_dataType_test`(
    `dataType_1` INT,
    `dataType_2` VARCHAR,
    `dataType_3` JSON
);
ALTER TABLE `runoob_dataType_test` ADD COLUMN `dataType_4` JSON NOT NULL;
ALTER TABLE `runoob_dataType_test` CHANGE COLUMN `dataType_4` `dataType_5` JSON NOT NULL;
```

Output

```
CREATE TABLE IF NOT EXISTS "public"."runoob_datatype_test"
(
  "datatype_1" INTEGER,
  "datatype_2" VARCHAR,
  "datatype_3" JSONB
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_1");
ALTER TABLE
  "public"."runoob_datatype_test"
ADD
  COLUMN "datatype_4" JSONB;
```



```
ALTER TABLE
 "public"."runoob_datatype_test" CHANGE COLUMN "datatype_4" "datatype_5" JSONB;
```

6.5.3 Functions and Expressions

Overview

The functions and expressions in MySQL do not exist in GaussDB(DWS) or are different from those in GaussDB(DWS). DSC migrates the functions and expressions based on the supported types of GaussDB(DWS). (compatible with ADB for MySQL)

Type Mapping

Table 6-33 Type mapping

MySQL/ADB Function	Description	MySQL INPUT	GaussDB(DWS) OUTPUT
CONVERT	Converts a value to a specified data type or character set.	CONVERT(A, B)	CAST(A AS B)
CURDATE	Specifies the current date. It is synonymous with CURDATE().	CURDATE/CURDATE()	CURRENT_DATE ()
GET_JSON_OBJECT	Parses a field in a JSON string.	GET_JSON_OBJECT(column, '$.obj.arg') GET_JSON_OBJECT(column, '$[i]')	JSON_EXTRACT(column, 'obj', 'arg') JSON_ARRAY_ELEMENT(column, 0)
JSON_EXTRACT	Queries the value of a field in JSON.	JSON_EXTRACT(column, '$.obj')	JSON_EXTRACT(column, 'obj')
REGEXP	Fuzzy match	REGEXP A NOT REGEXP A	~ A !~ A
UUID	Generates a unique value (sequence number).	UUID	SYS_GUID

MySQL/ADB Function	Description	MySQL INPUT	GaussDB(DWS) OUTPUT
SPLIT()[]	Splits **string** on **delimiter** and returns the **field**th column (counting from text of the first appeared delimiter).	SPLIT(string text, delimiter text)[field int]	SPLIT_PART(string text, delimiter text,field int)
RAND	Obtains a random number ranging from 0.0 to 1.0.	RAND()	RANDOM()
SLICE	Concatenates two or more strings, placing a separator between each one. The separator is specified by the first argument.	SLICE()	CONCAT_WS(sep text, str"any" [, str"any" [, ...]])
TRY_CAST	Converts **x** into the type specified by **y**.	TRY_CAST(X AS Y)	CAST(X AS Y)
CAST	Converts **x** into the type specified by **y**. The second argument of the cast function in the ADB can be forcibly converted to the string or double-precision data type. In GaussDB(DWS), it is converted to the varchar or double precision type.	CAST(X AS Y)	CAST(X AS Y)

Input example: CONVERT

```
SELECT CONVERT (IFNULL (BUSINESS_ID, 0) , DECIMAL(18, 2)) FROM ACCOUNT;
```

Output

```
SELECT cast (ifnull (business_id, 0) AS decimal(18, 2))FROM account;
```

Input example: CURDATE

```
SELECT CURDATE;
SELECT CURDATE();
```

Output

```
SELECT CURRENT_DATE();
SELECT CURRENT_DATE();
```

Input example: GET_JSON_OBJECT

```
SELECT GET_JSON_OBJECT(COL_JSON, '$.STORE.BICYCLE.PRICE');

SELECT GET_JSON_OBJECT(COL_JSON, '$.STORE.FRUIT[0]');
```

Output

```
SELECT
  JSON_EXTRACT_PATH(COL_JSON, 'STORE', 'BICYCLE', 'PRICE');
SELECT
  JSON_ARRAY_ELEMENT(JSON_EXTRACT_PATH(COL_JSON, 'STORE', 'FRUIT'), 0);
```

Input example: JSON_EXTRACT

```
SELECT JSON_EXTRACT(EVENT_ATTR,'$.TOPIC_ID');
```

Output

```
SELECT JSON_EXTRACT_PATH(EVENT_ATTR, 'TOPIC_ID');
```

Input example: REGEXP

```
SELECT * FROM USERS WHERE NAME NOT REGEXP '^ Wang';
SELECT * FROM USERS WHERE TEL REGEXP '[^4-5]{11}';
```

Output

```
SELECT * FROM USERS WHERE NAME !~ '^ Wang';
SELECT * FROM USERS WHERE TEL ~ '[^4-5]{11}';
```

Input example: UUID

```
SELECT CURDATE(str1), UUID(str2, str3) FROM T1;
SELECT A FROM B WHERE uuid() > 2;
```

Output

```
SELECT current_date (str1),sys_guid (str2, str3) FROM T1;
SELECT A FROM  B WHERE sys_guid () > 2;
```

Input example: SPLIT()[]

```
SELECT split('a-b-c-d-e', '-')[4];
```

Output

```
SELECT split_part('a-b-c-d-e', '-', 4);
```


Input Example RAND

```
SELECT rand();
```

Output

```
SELECT random ();
```

Input Example: SLICE

```
SELECT slice(split('2021_08_01','_'),1,3) from dual;
```

Output

```
SELECT
  concat_ws(
    split_part('2021_08_01', '_', 1),
    split_part('2021_08_01', '_', 2),
    split_part('2021_08_01', '_', 3)
  )
FROM
  dual;
```

Input example: TRY_CAST

```
select * from ods_pub where try_cast(pay_time AS timestamp) >= 1;
select try_cast(pay_time as timestamp) from obs_pub;
```

Output

```
SELECT * FROM ods_pub WHERE cast (pay_time AS timestamp) >= 1;
SELECT cast (pay_time as timestamp) FROM obs_pub;
```

Input Example: CAST

```
select cast(ifnull(c1, 0) as string) from t1;
select cast(ifnull(c1, 0) as varchar) from t1;
select cast(ifnull(c1, 0) as double) from t1;
select cast(ifnull(c1, 0) as int) from t1;
```

Output

```
SELECT cast (ifnull (c1, 0) as varchar) FROM t1;
SELECT cast (ifnull (c1, 0) as varchar) FROM t1;
SELECT cast (ifnull (c1, 0) as double precision) FROM t1;
SELECT cast (ifnull (c1, 0) as int) FROM t1;
```

6.5.4 Table (Optional Parameters and Operations)

This section describes the migration syntax of tables (optional parameters and operations). The migration syntax determines how the keywords and features are migrated. GaussDB(DWS) does not support table migration. Currently, all table migration methods are temporary.

6.5.4.1 ALGORITHM

MySQL has extended the support for **ALTER TABLE... ALGORITHM=INSTANT**: Users can add columns instantly and delete columns instantly anywhere in a table, and evaluate row size limits when adding a column.

This is not supported by GaussDB(DWS) and is deleted by DSC during the migration.

Input

```
ALTER TABLE runoob_alter_test ALGORITHM=DEFAULT;
ALTER TABLE runoob_alter_test ALGORITHM=INPLACE;
ALTER TABLE runoob_alter_test ALGORITHM=COPY;
ALTER TABLE runoob_alter_test ADD COLUMN COL_18 VARCHAR(64)  DEFAULT '00', ALGORITHM=INSTANT;
ALTER TABLE runoob_alter_test MODIFY COLUMN dataType7 BIGINT, ALGORITHM=COPY;
ALTER TABLE `runoob_alter_test` ALGORITHM=DEFAULT, ALGORITHM=INPLACE, ALGORITHM=COPY;
ALTER TABLE `runoob_alter_test` ADD COLUMN dataType11 INT, ALGORITHM=DEFAULT,
ALGORITHM=INPLACE, ALGORITHM=COPY;
ALTER TABLE runoob_alter_test CHANGE COLUMN dataType11 dataType12
SMALLINT ,ALGORITHM=INPLACE, ALGORITHM=COPY;
ALTER TABLE runoob_alter_test ALGORITHM=INPLACE, ALGORITHM=COPY, DROP COLUMN dataType12;
```

Output

```
ALTER TABLE
  "public"."runoob_alter_test"
ADD
  COLUMN "col_18" VARCHAR(256) DEFAULT '00';
ALTER TABLE
  "public"."runoob_alter_test" MODIFY "datatype7" BIGINT NULL DEFAULT NULL;
ALTER TABLE
  "public"."runoob_alter_test"
ADD
  COLUMN "datatype11" INTEGER;
ALTER TABLE
  "public"."runoob_alter_test" CHANGE COLUMN "datatype11" "datatype12" SMALLINT NULL DEFAULT
NULL;
ALTER TABLE
  "public"."runoob_alter_test" DROP COLUMN "datatype12" RESTRICT;
DROP TABLE IF EXISTS "public"."runoob_alter_test";
```

6.5.4.2 ALTER TABLE RENAME

GaussDB(DWS) prohibits including schema names in the **rename** clause, therefore, DSC supports renaming solely within the same schema. Renaming within the same schema omits the schema clause from the conversion result, while attempts to rename across schemas trigger an error report.

Input

```
ALTER TABLE `shce1`.`t1` rename to `t2`;
ALTER TABLE `shce1`.`t1` rename to t2;
ALTER TABLE `charge_data`.`group_shengfen2022` RENAME `charge_data`.`group_shengfen2022_jiu`;
ALTER TABLE `charge_data`.`group_shengfen2022` RENAME `charge_data`.`group_shengfen2022_jiu`,
RENAME `charge_data`.`group_shengfen2023_jiu`, RENAME `charge_data`.`group_shengfen2024_jiu`;
```

Output

```
ALTER TABLE "shce1"."t1" RENAME TO "t2";
ALTER TABLE "shce1"."t1" RENAME TO "t2";
ALTER TABLE "charge_data"."group_shengfen2022" RENAME TO "group_shengfen2022_jiu";
ALTER TABLE "charge_data"."group_shengfen2022" RENAME TO "group_shengfen2022_jiu", RENAME TO
"group_shengfen2023_jiu", RENAME TO "group_shengfen2024_jiu";
```

6.5.4.3 AUTO_INCREMENT

In database application, unique numbers that increase automatically are needed to identify records. In MySQL, the **AUTO_INCREMENT** attribute of a data column can be used to automatically generate the numbers. When creating a table, you can use **AUTO_INCREMENT=***n* to specify a start value. You can also use the **ALTER TABLE** *TABLE_NAME* **AUTO_INCREMENT=***n* command to reset the start value. GaussDB(DWS) does not support this parameter. During DSC migration, the

columns with this attribute set are migrated to the SERIAL type, and the keyword is deleted. The following **Table** describes the conversion:

Table 6-34 Data type conversion

MySQL Numeric Type	MySQL INPUT	GaussDB(DWS) OUTPUT
TINYINT	TINYINT	SMALLSERIAL
SMALLINT	SMALLINT UNSIGNED SMALLINT	SERIAL SMALLSERIAL
DOUBLE/FLOAT	DOUBLE/FLOAT	BIGSERIAL
INT/INTEGER	INT/INTEGER UNSIGNED INT/INTEGER	BIGSERIAL SERIAL
BIGINT/SERIAL	BIGINT/SERIAL	BIGSERIAL

Input

```
CREATE TABLE `public`.`job_instance` (
  `job_sche_id` int(11) NOT NULL AUTO_INCREMENT,
  `task_name` varchar(100) NOT NULL DEFAULT '',
  PRIMARY KEY (`job_sche_id`)
) ENGINE=InnoDB AUTO_INCREMENT=219 DEFAULT CHARSET=utf8;
```

Output

```
CREATE TABLE "public"."job_instance"
(
  "job_sche_id" SERIAL NOT NULL,
  "task_name" VARCHAR(400) NOT NULL DEFAULT '',
  PRIMARY KEY ("job_sche_id")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("job_sche_id");
```

In addition, GaussDB(DWS) does not support table definition modification using the **AUTO_INCREMENT** attribute. DSC will delete this attribute during migration.

Input

```
CREATE TABLE IF NOT EXISTS `public`.`runoob_alter_test`(
    `dataType1` int NOT NULL AUTO_INCREMENT,
    `dataType2` FLOAT(10,2),
    PRIMARY KEY(`dataType1`)
);

ALTER TABLE runoob_alter_test AUTO_INCREMENT 100;
ALTER TABLE runoob_alter_test AUTO_INCREMENT=100;
```

Output

```
CREATE TABLE IF NOT EXISTS "public"."runoob_alter_test"
(
  "datatype1" SERIAL NOT NULL,
  "datatype2" REAL,
  PRIMARY KEY ("datatype1")
)
```



```
 WITH ( ORIENTATION = ROW, COMPRESSION = NO )
 NOCOMPRESS
 DISTRIBUTE BY HASH ("datatype1");
```

6.5.4.4 AVG_ROW_LENGTH

In MySQL, **AVG_ROW_LENGTH** indicates the average length of each row. This is not supported by GaussDB(DWS) and is deleted by DSC during the migration.

Input

```
CREATE TABLE `public`.`runoob_tbl_test`(
    `runoob_id` VARCHAR(30),
    `runoob_title` VARCHAR(100) NOT NULL,
    `runoob_author` VARCHAR(40) NOT NULL,
    `submission_date` VARCHAR(30)
)AVG_ROW_LENGTH=10000;
```

Output

```
CREATE TABLE "public"."runoob_tbl_test"
(
  "runoob_id" VARCHAR(120),
  "runoob_title" VARCHAR(400) NOT NULL,
  "runoob_author" VARCHAR(160) NOT NULL,
  "submission_date" VARCHAR(120)
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("runoob_id");
```

6.5.4.5 BLOCK_SIZE

In ADB, this parameter specifies the number of values stored in each block in columnar storage, which is also the minimum I/O unit. GaussDB(DWS does not support table definition modification using this attribute. DSC will delete the keyword during migration.

Input

```
DROP TABLE IF EXISTS exists unsupport_parse_test;
CREATE TABLE `unsupport_parse_test` (
    `username` int,
    `update` timestamp not null default current_timestamp on update current_timestamp ,
     clustered key clustered_key(shopid ASC, datetype ASC)
)BLOCK_SIZE = 1024 index_ALL = 'y';
DROP TABLE IF EXISTS unsupport_parse_test;
```

Output

```
DROP TABLE IF EXISTS "public"."unsupport_parse_test";
CREATE TABLE "public"."unsupport_parse_test" (
  "username" INTEGER,
  "update" TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT CURRENT_TIMESTAMP
) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH ("username");
DROP TABLE IF EXISTS "public"."unsupport_parse_test";
```

6.5.4.6 CHARSET

CHARSET specifies the default character set for a table. GaussDB(DWS) does not support table definition modification using this attribute. DSC will delete the keyword during migration.

Input


```
CREATE TABLE `public`.`runoob_tbl_test`(
   `runoob_id` VARCHAR(30),
   `runoob_title` VARCHAR(100) NOT NULL,
   `runoob_author` VARCHAR(40) NOT NULL,
   `submission_date` VARCHAR(30)
)DEFAULT CHARSET=utf8;
```

Output

```
CREATE TABLE "public"."runoob_tbl_test"
(
  "runoob_id" VARCHAR(120),
  "runoob_title" VARCHAR(400) NOT NULL,
  "runoob_author" VARCHAR(160) NOT NULL,
  "submission_date" VARCHAR(120)
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("runoob_id");
```

6.5.4.7 CHECKSUM

In MySQL, **CHECKSUM** maintains a live checksum for all rows. GaussDB(DWS) does not support table definition modification using this attribute. DSC will delete the keyword during migration.

Input

```
CREATE TABLE `public`.`runoob_alter_test`(
   `dataType1` int NOT NULL AUTO_INCREMENT,
   `dataType2` FLOAT(10,2),
   `dataType3` DOUBLE(20,8),
   PRIMARY KEY(`dataType1`)
) CHECKSUM=1;

ALTER TABLE runoob_alter_test CHECKSUM 0;
ALTER TABLE runoob_alter_test CHECKSUM=0;

ALTER TABLE runoob_alter_test CHECKSUM 1;
ALTER TABLE runoob_alter_test CHECKSUM=1;
```

Output

```
CREATE TABLE "public"."runoob_alter_test"
(
  "datatype1" SERIAL NOT NULL,
  "datatype2" REAL,
  "datatype3" DOUBLE PRECISION,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");
```

6.5.4.8 CLUSTERED KEY

It is a clustered index in ADB that defines the sorting sequence in a table. The logical sequence of key values in a clustered index determines the physical sequence of rows in the table. GaussDB(DWS) does not support table definition modification using this attribute. DSC will delete the keyword during migration.

Input

```
DROP TABLE IF EXISTS unsupport_parse_test;
CREATE TABLE `unsupport_parse_test` (
   `username` int,
```



```
    clustered key clustered_key(shopid ASC, datetype ASC)
);
DROP TABLE IF EXISTS unsupport_parse_test;
```

Output

```
DROP TABLE IF EXISTS "public"."unsupport_parse_test";
CREATE TABLE "public"."unsupport_parse_test" (
  "username" INTEGER
) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH ("username");
DROP TABLE IF EXISTS "public"."unsupport_parse_test";
```

6.5.4.9 COLLATE

In MySQL, **COLLATE** specifies a default database sorting rule. GaussDB(DWS) does not support table definition modification using this attribute. DSC will delete the keyword during migration.

Input

```
CREATE TABLE `public`.`runoob_tbl_test`(
   `runoob_id` VARCHAR(30),
   `runoob_title` VARCHAR(100) NOT NULL,
   `runoob_author` VARCHAR(40) NOT NULL,
   `submission_date` VARCHAR(30)
) COLLATE=utf8_general_ci;

ALTER TABLE `public`.`runoob_tbl_test` COLLATE=utf8mb4_bin;
```

Output

```
CREATE TABLE "public"."runoob_tbl_test"
(
  "runoob_id" VARCHAR(120),
  "runoob_title" VARCHAR(400) NOT NULL,
  "runoob_author" VARCHAR(160) NOT NULL,
  "submission_date" VARCHAR(120)
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("runoob_id");
```

6.5.4.10 COMMENT

In MySQL, **COMMENT** is a comment for a table. In GaussDB(DWS), this attribute can be used to modify table definition. During migration using DSC, additional table attribute information is added.

Input

```
CREATE TABLE `public`.`runoob_alter_test`(
   `dataType1` int NOT NULL AUTO_INCREMENT,
`dataType2` FLOAT (10,2) COMMENT'dataType2 column',
   PRIMARY KEY(`dataType1`)
) comment='table comment';

ALTER TABLE `public`.`runoob_alter_test` COMMENT 'table comment after modification';
ALTER TABLE `public`.`runoob_alter_test` ADD INDEX age_index(dataType2) COMMENT' index';
```

Output

```
CREATE TABLE "public"."runoob_alter_test"
(
  "datatype1" SERIAL NOT NULL,
"datatype2" REAL COMMENT 'dataType2 column',
  PRIMARY KEY ("datatype1")
```



```
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1")
COMMENT 'Comment of the table';
ALTER TABLE `public`.`runoob_alter_test` COMMENT 'table comment after modification';
CREATE INDEX "age_index" ON "public"."runoob_alter_test" ("dataType2") COMMENT' index';
```

6.5.4.11 CONNECTION

GaussDB(DWS) does not support table definition modification using this attribute. DSC will delete the attribute during migration.

CAUTION

In MySQL, the keyword **CONNECTION** is used as a referenced, external data source. Currently, DSC cannot completely migrate the feature of **CONNECTION**. So as a workaround, it simply deletes the keyword.

Input

```
CREATE TABLE `public`.`runoob_alter_test`(
    `dataType1` int NOT NULL AUTO_INCREMENT,
    `dataType2` DOUBLE(20,8),
    `dataType3` TEXT NOT NULL,
    `dataType4` YEAR NOT NULL DEFAULT '2018',
    PRIMARY KEY(`dataType1`)
);

ALTER TABLE runoob_alter_test CONNECTION 'hello';
ALTER TABLE runoob_alter_test CONNECTION='hello';
```

Output

```
CREATE TABLE "public"."runoob_alter_test"
(
  "datatype1" SERIAL NOT NULL,
  "datatype2" DOUBLE PRECISION,
  "datatype3" TEXT NOT NULL,
  "datatype4" SMALLINT NOT NULL DEFAULT '2018',
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");
```

6.5.4.12 DEFAULT

In MySQL, when using ALTER TABLE to add a column with the not null constraint, if no value is specified, a default value is inserted. However, an error will be reported a value is inserted into a non-empty table in GaussDB(DWS). Therefore, for common data types, **Default values (see table 1.)** are specified for **ALTER TABLE ADD COLUMN** with **NOT NULL** constraint.

Input

```
DROP TABLE IF EXISTS default_test_1;
CREATE TABLE default_test_1
(
    c1 tinyint,
    c2 smallint,
    c3 MEDIUMINT,
```



```
   c4 int,
   c5 bigint
);
INSERT INTO default_test_1
VALUES (1, 1, 1, 1, 1);
SELECT *
FROM default_test_1;
ALTER TABLE default_test_1 add COLUMN (c1_1 tinyint not null, c2_1 SMALLINT not null, c3_1 MEDIUMINT not null, c4_1 int not null, c5_1 BIGINT not null);
SELECT *
FROM default_test_1;
```

Output

```
DROP TABLE IF EXISTS "public"."default_test_1";
CREATE TABLE "public"."default_test_1" (
  "c1" SMALLINT,
  "c2" SMALLINT,
  "c3" INTEGER,
  "c4" INTEGER,
  "c5" BIGINT
) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH ("c1");
INSERT INTO
  "public"."default_test_1"
VALUES
  (1, 1, 1, 1, 1);
SELECT
  *
FROM
  default_test_1;
ALTER TABLE
  "public"."default_test_1"
ADD
  COLUMN "c1_1" SMALLINT NOT NULL DEFAULT 0,
ADD
  COLUMN "c2_1" SMALLINT NOT NULL DEFAULT 0,
ADD
  COLUMN "c3_1" INTEGER NOT NULL DEFAULT 0,
ADD
  COLUMN "c4_1" INTEGER NOT NULL DEFAULT 0,
ADD
  COLUMN "c5_1" BIGINT NOT NULL DEFAULT 0;
SELECT
  *
FROM
  default_test_1;
```

Table 6-35

Data Type	MYSQL	GaussDB(DWS) Data Type	Conversion Result
Signed Integer	TINYINT	SMALLINT	0
	SMALLINT	SMALLINT	0
	MEDIUMINT	INT/INTEGER	0
	INT/INTEGER	INT/INTEGER	0
	BIGINT	BIGINT	0
Unsigned Integer	TINYINT UNSIGNED	TINYINT	0

Data Type	MYSQL	GaussDB(DWS) Data Type	Conversion Result
	SMALLINT UNSIGNED	INT	0
	MEDIUMINT UNSIGNED	INT	0
	INT UNSIGNED	BIGINT	0
	BIGINT UNSIGNED	NUMERIC	0
Float	FLOAT	REAL	0
	FLOAT UNSIGNED	REAL	0
	DOUBLE	DOUBLE PRECISION	0
	DOUBLE UNSIGNED	DOUBLE PRECISION	0
	DOUBLE PRECISION (DOUBLE)	DOUBLE PRECISION	0
	DOUBLE PRECISION [UNSIGNED] (DOUBLE)	DOUBLE PRECISION	0
	REAL (Processed as double by default, when set to **REAL_AS_FLOAT**, it is processed as float.)	REAL/DOUBLE PRECISION	0
	REAL [UNSIGNED] (Processed as double by default, when set to **REAL_AS_FLOAT**, it is processed as float.)	REAL/DOUBLE PRECISION	0
	NUMERIC/ DECIMAL	NUMERIC/ DECIMAL	0
Boolean	BOOLEAN/BOOL	BOOLEAN	0

Data Type	MYSQL	GaussDB(DWS) Data Type	Conversion Result
Character Type	CHAR/ CHARACTER	CHAR/ CHARACTER * 4	An empty string will be returned.
	NCHAR	CHAR/ CHARACTER * 4	An empty string will be returned.
	VARCHAR	VARCHAR * 4	An empty string will be returned.
Text	TINYTEXT	TEXT	An empty string will be returned.
	TEXT	TEXT	An empty string will be returned.
	MEDIUMTEXT	TEXT	An empty string will be returned.
	LONGTEXT	TEXT	An empty string will be returned.
Binary	BINARY	BYTEA	An empty string will be returned.
	VARBINARY	BYTEA	An empty string will be returned.
	TINYBLOB	BYTEA (column storage)/BLOB (non-column storage)	An empty string will be returned.
	BLOB	BYTEA (column storage)/BLOB (non-column storage)	An empty string will be returned.
	MEDIUMBLOB	BYTEA (column storage)/BLOB (non-column storage)	An empty string will be returned.
	LONGBLOB	BYTEA (column storage)/BLOB (non-column storage)	An empty string will be returned.
	CHAR BYTE (BINARY)	BYTEA	An empty string will be returned.
Date	DATE	DATE	1970-01-01

Data Type	MYSQL	GaussDB(DWS) Data Type	Conversion Result
	TIME	TIME(P) WITHOUT TIME ZONE	00:00:00
	DATETIME	TIMESTAMP WITHOUT TIME ZONE	1970-01-01 00:00:00
	TIMESTAMP	TIMESTAMP(P) WITH TIME ZONE	1970-01-01 00:00:00
	datetime/ datetime(0)	datetime->timestamp with out time zone datetime(0)->timestamp(0) with out time zone	1970-01-01 00:00:00.xx (The number of x depends on the precision.)
	YEAR	SMALLINT	0000
Bit String	BIT	BIT	DEFAULT 0:: BIT(x)
Set	ENUM	CHARACTER VARYING()	Return the first element by default.
	SET	CHARACTER VARYING()	An empty string will be returned.
Auto Increment	SERIAL (BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE)	CREATE SEQUENCE serial_name INCREMENT 2 START 1	No processing is required.
JSON	JSON	jsonb	No processing is required.
Spatial	point	point	No processing is required.
	polygon	polygon	No processing is required.
	geometry	Not supported. The input is returned as it was received	No processing is required.
	linestring	polygon	No processing is required.

Data Type	MYSQL	GaussDB(DWS) Data Type	Conversion Result
	geometrycollection	Not supported. The input is returned as it was received	No processing is required.
	multipoint	box	No processing is required.
	multilinestring	box	No processing is required.
	multipolygon	polygon	No processing is required.

6.5.4.13 DELAY_KEY_WRITE

DELAY_KEY_WRITE is valid only for MyISAM tables. It is used to delay updates until the table is closed. GaussDB(DWS) does not support table definition modification using this attribute. DSC will delete the attribute during migration.

Input

```
CREATE TABLE `public`.`runoob_tbl_test`(
   `runoob_id` VARCHAR(30),
   `runoob_title` VARCHAR(100) NOT NULL,
   `runoob_author` VARCHAR(40) NOT NULL,
   `submission_date` VARCHAR(30)
) ENGINE=MyISAM, DELAY_KEY_WRITE=0;

ALTER TABLE `public`.`runoob_tbl_test6` DELAY_KEY_WRITE=1;
```

Output

```
CREATE TABLE "public"."runoob_tbl_test"
(
  "runoob_id" VARCHAR(120),
  "runoob_title" VARCHAR(400) NOT NULL,
  "runoob_author" VARCHAR(160) NOT NULL,
  "submission_date" VARCHAR(120)
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("runoob_id");
```

6.5.4.14 DISTRIBUTE BY

ADB supports distribution keys. During DSC migration, the corresponding distribution keys are retained.

Input

```
CREATE TABLE COPY_DI_DISTRIBUTOR_BUYER_CONTRIBUTION_RANKING_V2 (
   SHOP_ID VARCHAR ,
   DISTRIBUTOR_ID VARCHAR ,
   BUYER_ID VARCHAR ,
   DATE_TYPE BIGINT ,
```



```
   PRIMARY KEY (SHOP_ID,DISTRIBUTOR_ID,DATE_TYPE,BUYER_ID)
) DISTRIBUTE BY HASH(SHOP_ID,DISTRIBUTOR_ID,DATE_TYPE);
```

Output

```
CREATE TABLE "public"."copy_di_distributor_buyer_contribution_ranking_v2" (
  "shop_id" VARCHAR,
  "distributor_id" VARCHAR,
  "buyer_id" VARCHAR,
  "date_type" BIGINT,
  PRIMARY KEY (
    "shop_id",
    "distributor_id",
    "date_type",
    "buyer_id"
  )
) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH (SHOP_ID,
DISTRIBUTOR_ID, DATE_TYPE);
```

6.5.4.15 DIRECTORY

DIRECTORY enables a tablespace to be created outside the data directory and index directory. It allows **DATA DIRECTORY** and **INDEX DIRECTORY**. GaussDB(DWS) does not support table definition modification using this attribute. DSC will delete the attribute during migration.

Input

```
CREATE TABLE `public`.`runoob_tbl_test1` (
`dataType1` int NOT NULL AUTO_INCREMENT,
`dataType2` DOUBLE(20,8),
PRIMARY KEY(`dataType1`)
) ENGINE=MYISAM DATA DIRECTORY = 'D:\\input' INDEX DIRECTORY= 'D:\\input';

CREATE TABLE `public`.`runoob_tbl_test2` (
`dataType1` int NOT NULL AUTO_INCREMENT,
`dataType2` DOUBLE(20,8),
PRIMARY KEY(`dataType1`)
) ENGINE=INNODB DATA DIRECTORY = 'D:\\input';
```

Output

```
CREATE TABLE "public"."runoob_tbl_test1"
(
  "datatype1" SERIAL NOT NULL,
  "datatype2" DOUBLE PRECISION,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");

CREATE TABLE "public"."runoob_tbl_test2"
(
  "datatype1" SERIAL NOT NULL,
  "datatype2" DOUBLE PRECISION,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");
```

6.5.4.16 ENGINE

In MySQL, **ENGINE** specifies the storage engine for a table. When the storage engine is **ARCHIVE**, **BLACKHOLE**, **CSV**, **FEDERATED**, **INNODB**, **MYISAM**,

MEMORY, **MRG_MYISAM**, **NDB**, **NDBCLUSTER**, or **PERFORMANCE_SCHEMA**, this attribute can be migrated and will be deleted during the migration.

Input

```
CREATE TABLE `public`.`runoob_alter_test`(
`dataType1` int NOT NULL,
`dataType2` DOUBLE(20,8),
PRIMARY KEY(`dataType1`)
)ENGINE=MYISAM;

## A.
ALTER TABLE runoob_alter_test ENGINE INNODB;
ALTER TABLE runoob_alter_test ENGINE=INNODB;

## B.
ALTER TABLE runoob_alter_test ENGINE MYISAM;
ALTER TABLE runoob_alter_test ENGINE=MYISAM;

## C.
ALTER TABLE runoob_alter_test ENGINE MEMORY;
ALTER TABLE runoob_alter_test ENGINE=MEMORY;
```

Output

```
CREATE TABLE "public"."runoob_alter_test"
(
  "datatype1" INTEGER NOT NULL,
  "datatype2" DOUBLE PRECISION,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");

-- A.

-- B.

-- C.
```

6.5.4.17 FOREIGN_KEY_CHECKS

For foreign key constraints in MySQL, GaussDB(DWS) does not support them modifying table definition. They will be deleted during DSC migration.

Input

```
SET foreign_key_checks = 0;
CREATE TABLE mall_order_dc (
    id bigint NOT NULL AUTO_INCREMENT,
    order_id varchar(50) NOT NULL,
    key order_id(order_id)
);
```

Output

```
CREATE TABLE "public"."mall_order_dc" (
  "id" BIGSERIAL NOT NULL,
  "order_id" VARCHAR(200) NOT NULL
) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH ("id");
CREATE INDEX "order_id" ON "public"."mall_order_dc" USING BTREE ("order_id");
```

6.5.4.18 IF NOT EXISTS

DSC supports the conversion of **IF NOT EXISTS**, which is reserved during the migration.

Input

```
DROP TABLE IF EXISTS `categories`;
CREATE TABLE IF NOT EXISTS `categories`
(
    `CategoryID` tinyint(5) unsigned NOT NULL PRIMARY KEY AUTO_INCREMENT,
    `CategoryName` varchar(15) CHARACTER SET utf8 COLLATE utf8_unicode_ci NOT NULL DEFAULT '' ,
    `Description` mediumtext CHARACTER SET utf8 COLLATE utf8_unicode_ci NOT NULL ,
    `Picture` varchar(50) CHARACTER SET utf8 COLLATE utf8_unicode_ci NOT NULL DEFAULT '',
    UNIQUE    (`CategoryID`)
)ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=latin1;
DROP TABLE IF EXISTS `categories`;
```

Output

```
DROP TABLE IF EXISTS "public"."categories";
CREATE TABLE IF NOT EXISTS "public"."categories" (
  "categoryid" SMALLSERIAL NOT NULL PRIMARY KEY,
  "categoryname" VARCHAR(60) NOT NULL DEFAULT '',
  "description" TEXT NOT NULL,
  "picture" VARCHAR(200) NOT NULL DEFAULT ''
) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH ("categoryid");
DROP TABLE IF EXISTS "public"."categories";
```

6.5.4.19 INDEX_ALL

In the ADB, **index_all='Y'** is used to create the full-column index. GaussDB(DWS) does not support table definition modification using this attribute. DSC will delete the attribute during migration.

Input

```
DROP TABLE IF EXISTS unsupport_parse_test;
CREATE TABLE `unsupport_parse_test` (
    `username` int,
    `update` timestamp not null default current_timestamp on update current_timestamp ,
     clustered key clustered_key(shopid ASC, datetype ASC)
)index_ALL = 'Y';
DROP TABLE IF EXISTS unsupport_parse_test;
```

Output

```
DROP TABLE IF EXISTS "public"."unsupport_parse_test";
CREATE TABLE "public"."unsupport_parse_test" (
  "username" INTEGER,
  "update" TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT CURRENT_TIMESTAMP
) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH ("username");
DROP TABLE IF EXISTS "public"."unsupport_parse_test";
```

6.5.4.20 INSERT_METHOD

INSERT_METHOD specifies the table into which the row should be inserted. Use a value of **FIRST** or **LAST** to have inserts go to the first or last table, or a value of **NO** to prevent inserts. DSC will delete this attribute during migration.

Input

```
CREATE TABLE `public`.`runoob_alter_test`(
    `dataType1` int NOT NULL AUTO_INCREMENT,
    `dataType2` DOUBLE(20,8),
    `dataType3` TEXT NOT NULL,
    PRIMARY KEY(`dataType1`)
) INSERT_METHOD=LAST;

ALTER TABLE runoob_alter_test INSERT_METHOD NO;
ALTER TABLE runoob_alter_test INSERT_METHOD=NO;
```



```
ALTER TABLE runoob_alter_test INSERT_METHOD FIRST;
ALTER TABLE runoob_alter_test INSERT_METHOD=FIRST;
ALTER TABLE runoob_alter_test INSERT_METHOD LAST;
ALTER TABLE runoob_alter_test INSERT_METHOD=LAST;
```

Output

```
CREATE TABLE "public"."runoob_alter_test"
(
  "datatype1" SERIAL NOT NULL,
  "datatype2" DOUBLE PRECISION,
  "datatype3" TEXT NOT NULL,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");
```

6.5.4.21 KEY_BLOCK_SIZE

KEY_BLOCK_SIZE choices vary depending on the storage engine used for a table. For MyISAM tables, **KEY_BLOCK_SIZE** optionally specifies the size in bytes to be used for index key blocks. For InnoDB tables, **KEY_BLOCK_SIZE** specifies the page size in kilobytes to be used for compressed InnoDB tables. GaussDB(DWS) does not support this attribute, which will be deleted by DSC during migration.

Input

```
CREATE TABLE `public`.`runoob_tbl_test`(
    `runoob_id` VARCHAR(30),
    `runoob_title` VARCHAR(100) NOT NULL,
    `runoob_author` VARCHAR(40) NOT NULL,
    `submission_date` VARCHAR(30)
) ENGINE=MyISAM KEY_BLOCK_SIZE=8;

ALTER TABLE runoob_tbl_test ENGINE=InnoDB;
ALTER TABLE runoob_tbl_test KEY_BLOCK_SIZE=0;
```

Output

```
CREATE TABLE "public"."runoob_tbl_test"
(
  "runoob_id" VARCHAR(120),
  "runoob_title" VARCHAR(400) NOT NULL,
  "runoob_author" VARCHAR(160) NOT NULL,
  "submission_date" VARCHAR(120)
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("runoob_id");
```

6.5.4.22 LOCK

GaussDB(DWS) does not support the **ALTER TABLE** *tbName* **LOCK** statement of MySQL, which will be deleted by DSC during migration.

Input

```
CREATE TABLE IF NOT EXISTS `runoob_alter_test`(
    `dataType1` int NOT NULL AUTO_INCREMENT,
    `dataType2` FLOAT(10),
    `dataType4` TEXT NOT NULL,
    `dataType5` YEAR NOT NULL DEFAULT '2018',
    `dataType6` DATETIME NOT NULL,
    `dataType7` CHAR NOT NULL DEFAULT '',
    `dataType8` VARCHAR(50),
```



```
  `dataType9` VARCHAR(50) NOT NULL DEFAULT '',
  `dataType10` TIME NOT NULL DEFAULT '10:20:59',
  PRIMARY KEY(`dataType1`)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;

## A.
ALTER TABLE runoob_alter_test LOCK DEFAULT;

## B.
ALTER TABLE runoob_alter_test LOCK=DEFAULT;

## C.
ALTER TABLE runoob_alter_test LOCK EXCLUSIVE;

## D.
ALTER TABLE runoob_alter_test LOCK=EXCLUSIVE;
```

Output

```
CREATE TABLE IF NOT EXISTS "public"."runoob_alter_test"
(
  "datatype1" SERIAL NOT NULL,
  "datatype2" REAL,
  "datatype4" TEXT NOT NULL,
  "datatype5" SMALLINT NOT NULL DEFAULT '2018',
  "datatype6" TIMESTAMP WITHOUT TIME ZONE NOT NULL,
  "datatype7" CHAR(4) NOT NULL DEFAULT '',
  "datatype8" VARCHAR(200),
  "datatype9" VARCHAR(200) NOT NULL DEFAULT '',
  "datatype10" TIME WITHOUT TIME ZONE NOT NULL DEFAULT '10:20:59',
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");

-- A.

-- B.

-- C.

-- D.
```

6.5.4.23 MAX_ROWS

In MySQL, **MAX_ROWS** indicates the maximum number of rows that can be stored in a table. This attribute will be deleted during migration using DSC.

Input

```
CREATE TABLE `public`.`runoob_alter_test`(
  `dataType1` int NOT NULL AUTO_INCREMENT,
  `dataType2` DOUBLE(20,8),
  `dataType3` TEXT NOT NULL,
  PRIMARY KEY(`dataType1`)
);

ALTER TABLE runoob_alter_test MAX_ROWS 100000;
ALTER TABLE runoob_alter_test MAX_ROWS=100000;
```

Output

```
CREATE TABLE "public"."runoob_alter_test"
(
  "datatype1" SERIAL NOT NULL,
  "datatype2" DOUBLE PRECISION,
  "datatype3" TEXT NOT NULL,
```



```
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");
```

6.5.4.24 MIN_ROWS

MIN_ROWS indicates the minimum number of rows that can be stored in a table. This attribute will be deleted during migration using DSC.

Input

```
CREATE TABLE `public`.`runoob_alter_test`(
    `dataType1` int NOT NULL AUTO_INCREMENT,
    `dataType2` DOUBLE(20,8),
    `dataType3` TEXT NOT NULL,
    PRIMARY KEY(`dataType1`)
);
ALTER TABLE runoob_alter_test MIN_ROWS 10000;
ALTER TABLE runoob_alter_test MIN_ROWS=10000;
```

Output

```
CREATE TABLE "public"."runoob_alter_test"
(
  "datatype1" SERIAL NOT NULL,
  "datatype2" DOUBLE PRECISION,
  "datatype3" TEXT NOT NULL,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");
```

6.5.4.25 PACK_KEYS

In MySQL, **PACK_KEYS** specifies the index compression mode in the MyISAM storage engine. GaussDB(DWS) does not support this attribute, which will be deleted by DSC during migration.

Input

```
CREATE TABLE `public`.`runoob_alter_test`(
    `dataType1` int NOT NULL AUTO_INCREMENT,
    `dataType2` DOUBLE(20,8),
    `dataType3` TEXT NOT NULL,
    PRIMARY KEY(`dataType1`)
) ENGINE=MyISAM PACK_KEYS=1;

##A
ALTER TABLE runoob_alter_test PACK_KEYS 0;
ALTER TABLE runoob_alter_test PACK_KEYS=0;

##B
ALTER TABLE runoob_alter_test PACK_KEYS 1;
ALTER TABLE runoob_alter_test PACK_KEYS=1;

##C
ALTER TABLE runoob_alter_test PACK_KEYS DEFAULT;
ALTER TABLE runoob_alter_test PACK_KEYS=DEFAULT;
```

Output

```
CREATE TABLE "public"."runoob_alter_test"
(
```



```
  "datatype1" SERIAL NOT NULL,

  "datatype2" DOUBLE PRECISION,
  "datatype3" TEXT NOT NULL,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");

--A

--B

--C
```

6.5.4.26 PARTITION BY

In MySQL, PARTITION BY is used to create partitioned tables. Currently, GaussDB(DWS) supports only range and list partitions in MySQL.

CAUTION

Hash partitions of PARTITION BY are not supported and deleted during migration. The deletion does not affect the current table functions, but may affect the performance slightly.

PARTITION BY RANGE

Input

```
CREATE TABLE IF NOT EXISTS `runoob_tbl_part_test`(
  `runoob_id` INT NOT NULL,
  `runoob_title` VARCHAR(100) NOT NULL,
  `runoob_author` VARCHAR(40) NOT NULL,
  `submission_date` VARCHAR(30)
)ENGINE=InnoDB DEFAULT CHARSET=utf8
PARTITION BY RANGE (`runoob_id`)(
    PARTITION p0 VALUES LESS THAN(100),
    PARTITION p1 VALUES LESS THAN(200),
    PARTITION p2 VALUES LESS THAN(300),
    PARTITION p3 VALUES LESS THAN(400),
    PARTITION p4 VALUES LESS THAN(500),
    PARTITION p5 VALUES LESS THAN (MAXVALUE)
);

ALTER TABLE `runoob_tbl_part_test` ADD PARTITION (PARTITION p6 VALUES LESS THAN (600));

ALTER TABLE `runoob_tbl_part_test` ADD PARTITION (PARTITION p7 VALUES LESS THAN (700),PARTITION
p8 VALUES LESS THAN (800));
```

Output

```
CREATE TABLE IF NOT EXISTS "public"."runoob_tbl_part_test" (
  "runoob_id" INTEGER NOT NULL,
  "runoob_title" VARCHAR(400) NOT NULL,
  "runoob_author" VARCHAR(160) NOT NULL,
  "submission_date" VARCHAR(120)
) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH ("runoob_id")
PARTITION BY RANGE ("runoob_id") (
  PARTITION p0
  VALUES
    LESS THAN (100),
```



```
  PARTITION p1
 VALUES
  LESS THAN (200),
  PARTITION p2
 VALUES
  LESS THAN (300),
  PARTITION p3
 VALUES
  LESS THAN (400),
  PARTITION p4
 VALUES
  LESS THAN (500),
  PARTITION p5
 VALUES
  LESS THAN (MAXVALUE)
);
ALTER TABLE "public"."runoob_tbl_part_test" ADD PARTITION p6 VALUES LESS THAN (600);
ALTER TABLE "public"."runoob_tbl_part_test" ADD PARTITION p7 VALUES LESS THAN (700), ADD
PARTITION p8 VALUES LESS THAN (800);
```

PARTITION BY LIST

Input

```
CREATE TABLE IF NOT EXISTS `runoob_tbl_part_test`(
   `runoob_id` INT NOT NULL,
   `runoob_title` VARCHAR(100) NOT NULL,
   `runoob_author` VARCHAR(40) NOT NULL,
   `submission_date` VARCHAR(30),
   PRIMARY KEY (`runoob_id`)
   )ENGINE=InnoDB DEFAULT CHARSET=utf8
   PARTITION BY LIST (runoob_id)(
   PARTITION r0 VALUES IN (1, 5, 9, 13, 17, 21),
   PARTITION r1 VALUES IN (2, 6, 10, 14, 18, 22),
   PARTITION r2 VALUES IN (3, 7, 11, 15, 19, 23),
   PARTITION r3 VALUES IN (4, 8, 12, 16, 20, 24)
   );
ALTER TABLE `runoob_tbl_part_test` ADD PARTITION (PARTITION p5 VALUES IN(30, 40, 50, 60, 70, 80));
```

Output

```
CREATE TABLE IF NOT EXISTS "public"."runoob_tbl_part_test" (
 "runoob_id" INTEGER NOT NULL,
 "runoob_title" VARCHAR(400) NOT NULL,
 "runoob_author" VARCHAR(160) NOT NULL,
 "submission_date" VARCHAR(120),
 PRIMARY KEY ("runoob_id")
) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH ("runoob_id")
PARTITION BY LIST (runoob_id) (
 PARTITION r0
 VALUES
  (1, 5, 9, 13, 17, 21),
  PARTITION r1
 VALUES
  (2, 6, 10, 14, 18, 22),
  PARTITION r2
 VALUES
  (3, 7, 11, 15, 19, 23),
  PARTITION r3
 VALUES
  (4, 8, 12, 16, 20, 24)
);
ALTER TABLE "public"."runoob_tbl_part_test" ADD PARTITION p5 VALUES (30, 40, 50, 60, 70, 80);
```


6.5.4.27 PASSWORD

In MySQL, **PASSWORD** indicates the user password. GaussDB(DWS) does not support this attribute, which will be deleted by DSC during migration.

Input

```
CREATE TABLE `public`.`runoob_alter_test`(
    `dataType1` int NOT NULL AUTO_INCREMENT,
    `dataType2` DOUBLE(20,8),
    `dataType3` TEXT NOT NULL,
    PRIMARY KEY(`dataType1`)
);
ALTER TABLE runoob_alter_test PASSWORD 'HELLO';
```

Output

```
CREATE TABLE "public"."runoob_alter_test"
(
  "datatype1" SERIAL NOT NULL,
  "datatype2" DOUBLE PRECISION,
  "datatype3" TEXT NOT NULL,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");
```

6.5.4.28 ROW_FORMAT

ROW_FORMAT defines the physical format in which the rows are stored. Row format choices vary depending on the storage engine used for the table. If you specify a row format that is not supported by the storage engine that is used for the table, the table will be created using that storage engine's default row format. If **ROW_FORMAT** is set to **DEFAULT**, the value will be migrated to **SET NOCOMPRESS**. If **ROW_FORMAT** is set to **COMPRESSED**, the value will be migrated to **SET COMPRESS**. GaussDB(DWS) supports only the preceding two **ROW_FORMAT** values. If other values are used, they will be deleted by DSC.

Input

```
CREATE TABLE `public`.`runoob_alter_test`(
    `dataType1` int NOT NULL AUTO_INCREMENT,
    `dataType2` FLOAT(10,2),
    `dataType3` DOUBLE(20,8),
    `dataType4` TEXT NOT NULL,
    PRIMARY KEY(`dataType1`)
) ENGINE=InnoDB;

## A.
ALTER TABLE runoob_alter_test ROW_FORMAT DEFAULT;
ALTER TABLE runoob_alter_test ROW_FORMAT=DEFAULT;

## B.
ALTER TABLE runoob_alter_test ROW_FORMAT DYNAMIC;
ALTER TABLE runoob_alter_test ROW_FORMAT=DYNAMIC;

## C.
ALTER TABLE runoob_alter_test ROW_FORMAT COMPRESSED;
ALTER TABLE runoob_alter_test ROW_FORMAT=COMPRESSED;

## D.
ALTER TABLE runoob_alter_test ROW_FORMAT REDUNDANT;
ALTER TABLE runoob_alter_test ROW_FORMAT=REDUNDANT;
```



```
## E.
ALTER TABLE runoob_alter_test ROW_FORMAT COMPACT;
ALTER TABLE runoob_alter_test ROW_FORMAT=COMPACT;
```

Output

```
CREATE TABLE "public"."runoob_alter_test"
(
  "datatype1" SERIAL NOT NULL,
  "datatype2" REAL,
  "datatype3" DOUBLE PRECISION,
  "datatype4" TEXT NOT NULL,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");

-- A.
ALTER TABLE "public"."runoob_alter_test" SET NOCOMPRESS;
ALTER TABLE "public"."runoob_alter_test" SET NOCOMPRESS;

-- B.

-- C.
ALTER TABLE "public"."runoob_alter_test" SET COMPRESS;
ALTER TABLE "public"."runoob_alter_test" SET COMPRESS;

-- D.

-- E.
```

6.5.4.29 STATS_AUTO_RECALC

STATS_AUTO_RECALC specifies whether to automatically recalculate persistent statistics for an InnoDB table. GaussDB(DWS) does not support this attribute, which will be deleted by DSC during migration.

Input

```
CREATE TABLE `public`.`runoob_alter_test`(
    `runoob_id` VARCHAR(30),
    `runoob_title` VARCHAR(100) NOT NULL,
    `runoob_author` VARCHAR(40) NOT NULL,
    `submission_date` VARCHAR(30)
) ENGINE=InnoDB, STATS_AUTO_RECALC=DEFAULT;

## A.
ALTER TABLE runoob_alter_test STATS_AUTO_RECALC DEFAULT;
ALTER TABLE runoob_alter_test STATS_AUTO_RECALC=DEFAULT;

## B.
ALTER TABLE runoob_alter_test STATS_AUTO_RECALC 0;
ALTER TABLE runoob_alter_test STATS_AUTO_RECALC=0;

## C.
ALTER TABLE runoob_alter_test STATS_AUTO_RECALC 1;
ALTER TABLE runoob_alter_test STATS_AUTO_RECALC=1;
```

Output

```
CREATE TABLE "public"."runoob_alter_test"
(
  "runoob_id" VARCHAR(120),
  "runoob_title" VARCHAR(400) NOT NULL,
  "runoob_author" VARCHAR(160) NOT NULL,
  "submission_date" VARCHAR(120)
)
```



```
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("runoob_id");

-- A.

-- B.

-- C.
```

6.5.4.30 STATS_PERSISTENT

In MySQL, **STATS_PERSISTENT** specifies whether to enable persistence statistics for an InnoDB table. The **CREATE TABLE** and **ALTER TABLE** statements can be used to enable persistence statistics. DSC will delete this attribute during migration.

Input

```
CREATE TABLE `public`.`runoob_alter_test`(
    `dataType1` int NOT NULL AUTO_INCREMENT,
    `dataType2` DOUBLE(20,8),
    `dataType3` TEXT NOT NULL,
    PRIMARY KEY(`dataType1`)
) ENGINE=InnoDB, STATS_PERSISTENT=0;

## A.
ALTER TABLE runoob_alter_test STATS_PERSISTENT DEFAULT;
ALTER TABLE runoob_alter_test STATS_PERSISTENT=DEFAULT;

## B.
ALTER TABLE runoob_alter_test STATS_PERSISTENT 0;
ALTER TABLE runoob_alter_test STATS_PERSISTENT=0;

## C.
ALTER TABLE runoob_alter_test STATS_PERSISTENT 1;
ALTER TABLE runoob_alter_test STATS_PERSISTENT=1;
```

Output

```
CREATE TABLE "public"."runoob_alter_test"
(
  "datatype1" SERIAL NOT NULL,
  "datatype2" DOUBLE PRECISION,
  "datatype3" TEXT NOT NULL,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");

-- A.

-- B.

-- C.
```

6.5.4.31 STATS_SAMPLE_PAGES

STATS_SAMPLE_PAGES specifies the number of index pages to sample when cardinality and other statistics for an indexed column are estimated. DSC will delete this attribute during migration.

Input

```
CREATE TABLE `public`.`runoob_alter_test`(
    `dataType1` int NOT NULL AUTO_INCREMENT,
```



```
    `dataType2` DOUBLE(20,8),
    `dataType3` TEXT NOT NULL,
    PRIMARY KEY(`dataType1`)
) ENGINE=InnoDB,STATS_SAMPLE_PAGES=25;

ALTER TABLE runoob_alter_test STATS_SAMPLE_PAGES 100;
ALTER TABLE runoob_alter_test STATS_SAMPLE_PAGES=100;
```

Output

```
CREATE TABLE "public"."runoob_alter_test"
(
  "datatype1" SERIAL NOT NULL,
  "datatype2" DOUBLE PRECISION,
  "datatype3" TEXT NOT NULL,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");
```

6.5.4.32 UNION

UNION is a table creation parameter of the MERGE storage engine. Creating a table using this keyword is similar to creating a common view. The created table logically combines the data of multiple tables that are specified by UNION. DSC migrates this feature to the view creation statement in GaussDB.

Input

```
CREATE TABLE t1 (
    a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    message CHAR(20)
) ENGINE=MyISAM;

CREATE TABLE t2 (
    a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    message CHAR(20)
) ENGINE=MyISAM;

CREATE TABLE total (
    a INT NOT NULL AUTO_INCREMENT,
    message CHAR(20))
ENGINE=MyISAM UNION=(t1,t2) INSERT_METHOD=LAST;
```

Output

```
CREATE TABLE "public"."t1" (
    "a" SERIAL NOT NULL PRIMARY KEY,
    "message" CHAR(80)
)
WITH ( ORIENTATION = ROW, COMPRESSION = NO )
NOCOMPRESS
DISTRIBUTE BY HASH ("a");

CREATE TABLE "public"."t2" (
    a SERIAL NOT NULL PRIMARY KEY,
    message CHAR(80)
)
WITH ( ORIENTATION = ROW, COMPRESSION = NO )
NOCOMPRESS
DISTRIBUTE BY HASH ("a");

CREATE VIEW "public"."total"(a, message) AS
SELECT * FROM "public"."t1"
UNION ALL
SELECT * FROM "public"."t2";
```


6.5.4.33 WITH AS

WITH AS is used in GaussDB(DWS) to declare one or more subqueries that can be referenced by name in the main query. It is equivalent to a temporary table. DSC supports this keyword, which is retained in the migration.

Input

```
WITH e AS (
  SELECT city, sum(population) FROM t1 group by city
  ),
  d AS (
  SELECT max(music) as max_music, min(music) as min_music from student
  ),
  s AS (
  SELECT * FROM subject
  )
SELECT * FROM e;
```

Output

```
WITH e AS (
  SELECT
    city, sum(population)
  FROM
    t1
  GROUP BY
    city
),
d AS (
  SELECT
    max(music) AS "max_music", min(music) AS "min_music"
  FROM
    student
),
s AS (
  SELECT
    *
  FROM
    subject
)
SELECT
  *
FROM
  e;
```

6.5.4.34 CHANGE (Column Modification)

MySQL uses the **CHANGE** keyword to change column names and data types and set **NOT NULL** constraints. DSC will perform adaptation based on GaussDB(DWS) features during migration.

Input

```
CREATE TABLE IF NOT EXISTS `runoob_alter_test`(
    `dataType0` varchar(128),
    `dataType1` bigint,
    `dataType2` bigint,
    `dataType3` bigint,
    `dataType4` bigint
)ENGINE=InnoDB DEFAULT CHARSET=utf8;

## A.
ALTER TABLE runoob_alter_test CHANGE dataType1 dataType1New VARCHAR(50);

## B.
ALTER TABLE runoob_alter_test CHANGE dataType2 dataType2New VARCHAR(50) NOT NULL;
```



```
## C.
ALTER TABLE runoob_alter_test CHANGE dataType3 dataType3New VARCHAR(100) FIRST;

## D.
ALTER TABLE runoob_alter_test CHANGE dataType4 dataType4New VARCHAR(50) AFTER dataType1;
```

Output

```
CREATE TABLE "public"."runoob_alter_test"
(
  "datatype0" VARCHAR(512),
  "datatype1" BIGINT,
  "datatype2" BIGINT,
  "datatype3" BIGINT,
  "datatype4" BIGINT
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype0");

-- A.
ALTER TABLE "public"."runoob_alter_test" CHANGE COLUMN "datatype1" "datatype1new" VARCHAR(200)
NULL DEFAULT NULL;

-- B.
ALTER TABLE "public"."runoob_alter_test" CHANGE COLUMN "datatype2" "datatype2new" VARCHAR(200)
NOT NULL;

-- C.
ALTER TABLE "public"."runoob_alter_test" CHANGE COLUMN "datatype3" "datatype3new" VARCHAR(400)
NULL DEFAULT NULL;

-- D.
ALTER TABLE "public"."runoob_alter_test" CHANGE COLUMN "datatype4" "datatype4new" VARCHAR(200)
NULL DEFAULT NULL;
```

6.5.4.35 CHECK Constraint

Specifies an expression producing a Boolean result which new or updated rows must satisfy for an insert or update operation to succeed. Expressions evaluating to **TRUE** or **UNKNOWN** succeed. If any row of an insert or update operation produces a FALSE result, an error exception is raised and the insert or update does not alter the database.

A check constraint specified as a column constraint should reference only the column's values, while an expression appearing in a table constraint can reference multiple columns.

CREATE TABLE CHECK

When creating a table in GaussDB(DWS), you can place the **CHECK** constraint of a column after the column field or below the column field. The syntax is as follows: CHECK (column name > 0), To name the **CHECK** constraint or define the **CHECK** constraint of multiple columns, use the **CONSTRAINT chk_name CHECK (column_namw1 >0 AND column_name2='x>x")** syntax.

CAUTION

The CHECK constraint is also removed if the table is removed.

Input

```
DROP TABLE IF EXISTS `t1`;
CREATE TABLE IF NOT EXISTS t1  (
  id int(25) not null primary key check (id > 1 and id < 100),
city varchar(255) not null unique check (city='city 1' or city='city 2' or city='city 3'),
  population int(25) not null ,
  constraint t1_1 check (population>0 and population<10000)
) ;
```

Output

```
DROP TABLE IF EXISTS "public"."t1";
CREATE TABLE IF NOT EXISTS "public"."t1" (
  "id" INTEGER NOT NULL PRIMARY KEY CHECK (
    id > 1
    and id < 100
  ),
  "city" VARCHAR(1020) NOT NULL CHECK (
city ='City 1'
or city ='City 2'
or city ='City 3'
  ),
  "population" INTEGER NOT NULL,
  CONSTRAINT t1_1 CHECK (
    population > 0
    and population < 10000
  )
) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH ("id");
```

ALTER TABLE CHECK

When you perform operations on table columns, you can have or have no naming constraint. To create a CHECK constraint in a column, use the **ALTER TABLE** *Table name* **ADD CHECK (<***Check constraint***>);** syntax.

To name a **CHECK** constraint and define it for multiple columns, use the **ALTER TABLE** *Table name* **ADD CONSTRAINT <***Name of the check constraint***> CHECK(<***Check constraint***>);** syntax.

Input

```
ALTER TABLE t1 add check (id>1 and id > 2  and id < 100 or id =50);
ALTER TABLE student add constraint check02 check (id > 1 and class = '3');
ALTER TABLE t1 add check (class=1 or class =2  or class = 3 or class=4 or class = 5 or class = 6);
ALTER TABLE t1 add check (city='city 1' or city='city 2' or city='city 3');
```

Output

```
ALTER TABLE "public"."t1" ADD CHECK ( id > 1  and id > 2 and id < 100  or id = 50 );
ALTER TABLE "public"."student" ADD CONSTRAINT check02 CHECK (  id > 1  and class = '3');
ALTER TABLE "public"."t1" ADD CHECK ( class = 1 or class = 2 or class = 3  or class = 4 or class = 5 or class
= 6);
ALTER TABLE "public"."t1" ADD CHECK (city ='City 1' or city =' City 2' or city =' City 3');
```

6.5.4.36 DROP (Table Deletion)

Both GaussDB(DWS) and MySQL support using the **DROP** statement to delete tables. However, GaussDB(DWS) does not support the **RESTRICT | CASCADE** keyword in the **DROP** statement. DSC will delete the keywords during migration.

Input

```
CREATE TABLE IF NOT EXISTS `public`.`express_elb_server`(
   `runoob_id` VARCHAR(10),
```



```
  `runoob_title` VARCHAR(100) NOT NULL,
  `runoob_author` VARCHAR(40) NOT NULL,
  `submission_date` VARCHAR(10)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
DROP TABLE  `public`.`express_elb_server` RESTRICT;
```

Output

```
CREATE TABLE IF NOT EXISTS "public"."express_elb_server"
(
  "runoob_id" VARCHAR(40),
  "runoob_title" VARCHAR(400) NOT NULL,
  "runoob_author" VARCHAR(160) NOT NULL,
  "submission_date" VARCHAR(40)
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("runoob_id");
DROP TABLE "public"."express_elb_server";
```

6.5.4.37 LIKE (Table Cloning)

In the MySQL database, you can use the **CREATE TABLE .. LIKE ..** method to clone the old table structure to create a table. It is also supported by GaussDB(DWS). DSC will add additional table attribute information during migration.

Input

```
CREATE TABLE IF NOT EXISTS `public`.`runoob_tbl_old`(
   `dataType_1` YEAR,
   `dataType_2` YEAR(4),
   `dataType_3` YEAR DEFAULT '2018',
   `dataType_4` TIME DEFAULT NULL
);

CREATE TABLE `runoob_tbl` (like `runoob_tbl_old`);
```

Output

```
CREATE TABLE IF NOT EXISTS "public"."runoob_tbl_old"
(
  "datatype_1" SMALLINT,
  "datatype_2" SMALLINT,
  "datatype_3" SMALLINT DEFAULT '2018',
  "datatype_4" TIME WITHOUT TIME ZONE DEFAULT NULL
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_1");

CREATE TABLE "public"."runoob_tbl"( LIKE "public"."runoob_tbl_old"
   INCLUDING COMMENTS INCLUDING CONSTRAINTS INCLUDING DEFAULTS INCLUDING INDEXES
INCLUDING STORAGE);
```

6.5.4.38 MODIFY (Modifying a Column)

MySQL uses the **MODIFY** keyword to change column data types and set **NOT NULL** constraints. DSC will perform adaptation based on GaussDB(DWS) features during migration.

Input

```
CREATE TABLE IF NOT EXISTS `runoob_alter_test`(
   `dataType0` varchar(100),
   `dataType1` bigint,
   `dataType2` bigint,
```



```
   `dataType3` bigint
)ENGINE=InnoDB DEFAULT CHARSET=utf8;

## A.
ALTER TABLE runoob_alter_test MODIFY dataType1 smallint;

## B.
ALTER TABLE runoob_alter_test MODIFY dataType1 smallint NOT NULL;

## C.
ALTER TABLE runoob_alter_test MODIFY dataType1 smallint NOT NULL FIRST;

## D.
ALTER TABLE runoob_alter_test MODIFY dataType1 smallint NOT NULL AFTER dataType3;
```

Output

```
CREATE TABLE "public"."runoob_alter_test"
(
  "datatype0" VARCHAR(400),
  "datatype1" BIGINT,
  "datatype2" BIGINT,
  "datatype3" BIGINT
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype0");

-- A.
ALTER TABLE "public"."runoob_alter_test" MODIFY "datatype1" SMALLINT NULL DEFAULT NULL;

-- B.
ALTER TABLE "public"."runoob_alter_test" MODIFY "datatype1" SMALLINT NOT NULL;

-- C.
ALTER TABLE "public"."runoob_alter_test" MODIFY "datatype1" SMALLINT NOT NULL;

-- D.
ALTER TABLE "public"."runoob_alter_test" MODIFY "datatype1" SMALLINT NOT NULL;
```

6.5.4.39 TRUNCATE (Table Deletion)

In MySQL, the **TABLE** keyword can be omitted when the **TRUNCATE** statement is used to delete table data. GaussDB(DWS) does not support this usage. In addition, DSC will add **CONTINUE IDENTITY RESTRICT** to the **TRUNCATE** statement for migration.

Input

```
TRUNCATE TABLE `public`.`test_create_table01`;
TRUNCATE TEST_CREATE_TABLE01;
```

Output

```
TRUNCATE TABLE "public"."test_create_table01" CONTINUE IDENTITY RESTRICT;
TRUNCATE TABLE "public"."test_create_table01" CONTINUE IDENTITY RESTRICT;
```

6.5.4.40 ROUNDROBIN Table

GaussDB(DWS) supports the creation of **roundrobin** tables. You can set **table.type** in **Table 6-13**. Set table.type=ROUND-ROBIN.

Input

```
CREATE TABLE charge_snapshot (
   id bigint NOT NULL,
```



```
    profit_model integer,
    ladder_rebate_rule text
);
```

Output

```
CREATE TABLE "public"."charge_snapshot" (
  "id" BIGINT NOT NULL,
  "profit_model" INTEGER,
  "ladder_rebate_rule" TEXT
) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY ROUNDROBIN;
```

6.5.4.41 RENAME (Table Renaming)

The statement for renaming a table in MySQL is slightly different from that in GaussDB(DWS). DSC will perform adaptation based on GaussDB(DWS) features during migration.

CAUTION

Currently, DSC does not support original table names prefixed with **DATABASE/SCHEMA.**

1. MySQL allows you to use the **RENAME TABLE** statement to change a table name.

   Input

   ```
   # Rename a single table.
   RENAME TABLE DEPARTMENT TO NEWDEPT;

    # Rename multiple tables.
   RENAME TABLE NEWDEPT TO NEWDEPT_02,PEOPLE TO PEOPLE_02;
   ```

   Output

   ```
   -- Rename a single table.
   ALTER TABLE "public"."department" RENAME TO "newdept";

   -- Rename multiple tables.
   ALTER TABLE "public"."newdept" RENAME TO "newdept_02";
   ALTER TABLE "public"."people" RENAME TO "people_02";
   ```

2. In MySQL, the **ALTER TABLE RENAME** statement is used to change a table name. When this statement is migrated by DSC, the keyword **AS** is converted to **TO**.

   Input

   ```
   ## A.
   ALTER TABLE runoob_alter_test RENAME TO runoob_alter_testnew;

   ## B.
   ALTER TABLE runoob_alter_testnew RENAME AS runoob_alter_testnewnew;
   ```

   Output

   ```
   -- A.
   ALTER TABLE "public"."runoob_alter_test" RENAME TO "runoob_alter_testnew";

   -- B.
   ALTER TABLE "public"."runoob_alter_testnew" RENAME TO "runoob_alter_testnewnew";
   ```


6.5.4.42 SET|DROP COLUMN DEFAULT VALUE

In MySQL, the **COLUMN** keyword can be omitted when the **ALTER** statement is used to set the default value of a column. DSC will perform adaptation based on GaussDB(DWS) features during migration.

Input

```
CREATE TABLE IF NOT EXISTS `runoob_alter_test`(
   `dataType1` int NOT NULL AUTO_INCREMENT,
   `dataType2` FLOAT(10,2),
   `dataType3` DOUBLE(20,8),
   `dataType4` TEXT NOT NULL,
   `dataType5` YEAR NOT NULL DEFAULT '2018',
   `dataType6` DATETIME NOT NULL DEFAULT '2018-10-12 15:27:33.999999',
   `dataType7` CHAR NOT NULL DEFAULT '',
   `dataType8` VARCHAR(50),
   `dataType9` VARCHAR(50) NOT NULL DEFAULT '',
   `dataType10` TIME NOT NULL DEFAULT '10:20:59',
   PRIMARY KEY(`dataType1`)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;

ALTER TABLE runoob_alter_test ALTER dataType2 SET DEFAULT 1;
ALTER TABLE runoob_alter_test ALTER COLUMN dataType2 SET DEFAULT 3;
ALTER TABLE runoob_alter_test ALTER dataType2 DROP DEFAULT;
ALTER TABLE runoob_alter_test ALTER COLUMN dataType2 DROP DEFAULT;
```

Output

```
CREATE TABLE IF NOT EXISTS "public"."runoob_alter_test"
(
  "datatype1" SERIAL NOT NULL,
  "datatype2" REAL,
  "datatype3" DOUBLE PRECISION,
  "datatype4" TEXT NOT NULL,
  "datatype5" SMALLINT NOT NULL DEFAULT '2018',
  "datatype6" TIMESTAMP WITHOUT TIME ZONE NOT NULL DEFAULT '2018-10-12 15:27:33.999999',
  "datatype7" CHAR(4) NOT NULL DEFAULT '',
  "datatype8" VARCHAR(200),
  "datatype9" VARCHAR(200) NOT NULL DEFAULT '',
  "datatype10" TIME WITHOUT TIME ZONE NOT NULL DEFAULT '10:20:59',
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");

ALTER TABLE "public"."runoob_alter_test" ALTER COLUMN "datatype2" SET DEFAULT '1';
ALTER TABLE "public"."runoob_alter_test" ALTER COLUMN "datatype2" SET DEFAULT '3';
ALTER TABLE "public"."runoob_alter_test" ALTER COLUMN "datatype2" DROP DEFAULT;
ALTER TABLE "public"."runoob_alter_test" ALTER COLUMN "datatype2" DROP DEFAULT;
```

6.5.4.43 Renaming a Column

Using a Reserved Keyword As a Column Name

In GaussDB(DWS), reserved keywords must be enclosed in double quotation marks when they are used as column names. The following reserved keywords are supported: desc, checksum, operator, and size.

Input

```
CREATE TABLE `desc` (
   user char,
   checksum int,
   operator smallint,
```



```
    desc char,
    size bigint
);
```

Output

```
CREATE TABLE "public"."desc" (
  "user" CHAR(4),
  "checksum" INTEGER,
  "operator" SMALLINT,
  "desc" CHAR(4),
  "size" BIGINT
) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH ("user");
```

Using the Name of a Hidden Column of a System Catalog As a Column Name

If the defined column name is the same as that of a hidden column in a GaussDB(DWS) system catalog (including **xc_node_id** and **tableoid**, **cmax**, **xmax**, **cmin**, **xmin**, **ctid**, **tid.**), you need to rename the column. To rename the column, add **_new** at the end. For example, change **xc_node_id** to **xc_node_id_new**.

Input

```
DROP TABLE IF EXISTS `renameFieldName`;
CREATE TABLE IF NOT EXISTS `renameFieldName`(
    xc_node_id int,
    tableoid char(3),
    cmax smallint,
    xmax bigint auto_increment,
    cmin varchar(10),
    xmin int,
    ctid int auto_increment,
    tid int
);
DROP TABLE IF EXISTS `renameFieldName`;
```

Output

```
DROP TABLE IF EXISTS "public"."renamefieldname";
CREATE TABLE IF NOT EXISTS "public"."renamefieldname" (
  "xc_node_id_new" INTEGER,
  "tableoid_new" CHAR(12),
  "cmax_new" SMALLINT,
  "xmax_new" BIGSERIAL,
  "cmin_new" VARCHAR(40),
  "xmin_new" INTEGER,
  "ctid_new" SERIAL,
  "tid_new" INTEGER
) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH
("xc_node_id_new");
DROP TABLE IF EXISTS "public"."renamefieldname";
```

6.5.4.44 Row-Store/Column-Store Table Compression

In GaussDB(DWS), only column-store tables can be compressed. Row-store tables cannot be compressed. The row-column storage compression mechanism is optimized. DSC performs adaptation based on GaussDB(DWS) features during migration.

Compression parameters:

table.compress.mode. If keyword **COMPRESS** is specified in **CREATE TABLE**, the compression feature will be triggered in the case of bulk **INSERT** operations. If this

feature is enabled, a scan will be performed on all tuple data within the page to generate a dictionary and then the tuple data will be compressed and stored. If **NOCOMPRESS** is used, tables will not be compressed.

table.compress.row and **table.compress.column** determine the compression level, which determines the compression ratio and duration. Generally, the higher the level of compression, the higher the ratio, the longer the duration, vice versa. The actual compression ratio depends on the distribution mode of table data loaded.

table.compress.level determines the compression sublevel, which determines the table data compression ratio and duration. A compression level is divided into sublevels, providing more choices for the compression ratio and duration. As the value becomes larger, the compression ratio becomes higher and duration longer at the same compression level.

Input example of a row-store table

```
DROP TABLE IF EXISTS `public`.`runoob_tbl`;
CREATE TABLE IF NOT EXISTS `public`.`runoob_tbl`(
   `runoob_id` VARCHAR,
   `runoob_title` VARCHAR(100) NOT NULL,
   `runoob_author` VARCHAR(40) NOT NULL,
   `submission_date` VARCHAR
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Output example of a row-store table

```
DROP TABLE IF EXISTS "public"."runoob_tbl";
CREATE TABLE IF NOT EXISTS "public"."runoob_tbl" (
  "runoob_id" VARCHAR,
  "runoob_title" VARCHAR(400) NOT NULL,
  "runoob_author" VARCHAR(160) NOT NULL,
  "submission_date" VARCHAR
) WITH (ORIENTATION = ROW, COMPRESSION = YES) COMPRESS DISTRIBUTE BY HASH ("runoob_id");
```

Input example of a column-store table

```
DROP TABLE IF EXISTS `public`.`runoob_tbl`;
CREATE TABLE IF NOT EXISTS `public`.`runoob_tbl`(
   `runoob_id` VARCHAR,
   `runoob_title` VARCHAR(100) NOT NULL,
   `runoob_author` VARCHAR(40) NOT NULL,
   `submission_date` VARCHAR
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Output example of a column-store table

```
DROP TABLE IF EXISTS "public"."runoob_tbl";
CREATE TABLE IF NOT EXISTS "public"."runoob_tbl" (
  "runoob_id" VARCHAR,
  "runoob_title" VARCHAR(400) NOT NULL,
  "runoob_author" VARCHAR(160) NOT NULL,
  "submission_date" VARCHAR
) WITH (
  COMPRESSLEVEL = 1,
  ORIENTATION = COLUMN,
  COMPRESSION = LOW
) DISTRIBUTE BY HASH ("runoob_id");
```


6.5.4.45 Adding/Deleting a Column

The statements for adding and deleting columns in MySQL are different from those in GaussDB(DWS). DSC will perform adaptation based on GaussDB(DWS) features during migration.

CAUTION

GaussDB does not support the update of sequence numbers in table definitions. Temporarily, Migration Tool does not support the complete migration of the <cf id="Bold">FRIST</cf> and <cf id="Bold">AFTER</cf> features. So as a workaround, it simply deletes the keyword.

Input

```
CREATE TABLE IF NOT EXISTS `runoob_alter_test`(
   `dataType1` int NOT NULL AUTO_INCREMENT,
   `dataType2` FLOAT(10,2),
   `dataType3` DOUBLE(20,8),
   `dataType4` TEXT NOT NULL,
   `dataType5` YEAR NOT NULL DEFAULT '2018',
   `dataType6` DATETIME NOT NULL DEFAULT '2018-10-12 15:27:33.999999',
   `dataType7` CHAR NOT NULL DEFAULT '',
   `dataType8` VARCHAR(50),
   `dataType9` VARCHAR(50) NOT NULL DEFAULT '',
   `dataType10` TIME NOT NULL DEFAULT '10:20:59',
   PRIMARY KEY(`dataType1`)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;

## A.
ALTER TABLE runoob_alter_test ADD dataType1_1 INT NOT NULL AFTER dataType1;
ALTER TABLE runoob_alter_test DROP dataType1_1;

## B.
ALTER TABLE runoob_alter_test ADD dataType1_1 INT NOT NULL FIRST;
ALTER TABLE runoob_alter_test DROP dataType1_1;

## C.
ALTER TABLE runoob_alter_test ADD COLUMN dataType1_1 INT NOT NULL AFTER dataType2;
ALTER TABLE runoob_alter_test DROP COLUMN dataType1_1;

## D.
ALTER TABLE runoob_alter_test ADD COLUMN dataType1_1 INT NOT NULL FIRST;
ALTER TABLE runoob_alter_test DROP COLUMN dataType1_1;

## E.
ALTER TABLE runoob_alter_test ADD COLUMN(dataType1_1 INT NOT NULL, dataType1_2 VARCHAR(200)
NOT NULL);
ALTER TABLE runoob_alter_test DROP COLUMN dataType1_1, DROP COLUMN dataType1_2;
```

Output

```
CREATE TABLE "public"."runoob_alter_test"
(
  "datatype1" SERIAL NOT NULL,
  "datatype2" REAL,
  "datatype3" DOUBLE PRECISION,
  "datatype4" TEXT NOT NULL,
  "datatype5" SMALLINT NOT NULL DEFAULT '2018',
  "datatype6" TIMESTAMP WITHOUT TIME ZONE NOT NULL DEFAULT '2018-10-12 15:27:33.999999',
  "datatype7" CHAR(4) NOT NULL DEFAULT '',
  "datatype8" VARCHAR(200),
  "datatype9" VARCHAR(200) NOT NULL DEFAULT '',
  "datatype10" TIME WITHOUT TIME ZONE NOT NULL DEFAULT '10:20:59',
```



```
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");

-- A.
ALTER TABLE "public"."runoob_alter_test" ADD COLUMN "datatype1_1" INTEGER NOT NULL DEFAULT 0;
ALTER TABLE "public"."runoob_alter_test" DROP COLUMN "datatype1_1" RESTRICT;

-- B.
ALTER TABLE "public"."runoob_alter_test" ADD COLUMN "datatype1_1" INTEGER NOT NULL DEFAULT 0;
ALTER TABLE "public"."runoob_alter_test" DROP COLUMN "datatype1_1" RESTRICT;

-- C.
ALTER TABLE "public"."runoob_alter_test" ADD COLUMN "datatype1_1" INTEGER NOT NULL DEFAULT 0;
ALTER TABLE "public"."runoob_alter_test" DROP COLUMN "datatype1_1" RESTRICT;

-- D.
ALTER TABLE "public"."runoob_alter_test" ADD COLUMN "datatype1_1" INTEGER NOT NULL DEFAULT 0;
ALTER TABLE "public"."runoob_alter_test" DROP COLUMN "datatype1_1" RESTRICT;

-- E.
ALTER TABLE "public"."runoob_alter_test" ADD COLUMN "datatype1_1" INTEGER NOT NULL, ADD
COLUMN "datatype1_2" VARCHAR(800) NOT NULL DEFAULT '';
ALTER TABLE "public"."runoob_alter_test" DROP COLUMN "datatype1_1" RESTRICT, DROP COLUMN
"datatype1_2" RESTRICT;
```

6.5.5 Indexes

6.5.5.1 Unique Indexes

GaussDB(DWS) does not support the combination of unique indexes (constraints) and primary key constraints. DSC will perform adaptation based on GaussDB(DWS) features during migration.

CAUTION

If MySQL unique indexes (constraints) and primary key constraints are used together during migration, OLAP distribution keys may become unavailable. Therefore, this scenario is not supported by DSC.

1. For a unique inline index, if the primary key index and the unique index are the same column, DSC will remove the unique index during migration.

   Input

   ```
   CREATE TABLE IF NOT EXISTS `public`.`runoob_dataType_test`
   (
     `id` INT PRIMARY KEY AUTO_INCREMENT,
     `name` VARCHAR(128) NOT NULL,
     UNIQUE (id ASC)
   );
   ```

   Output

   ```
   CREATE TABLE IF NOT EXISTS "public"."runoob_datatype_test"
   (
     "id" SERIAL PRIMARY KEY,
     "name" VARCHAR(128) NOT NULL
   )
     WITH ( ORIENTATION = ROW, COMPRESSION = NO )
     NOCOMPRESS
     DISTRIBUTE BY HASH ("id");
   ```


2. If there is a unique index created by **ALTER TABLE**, DSC will create a normal index based on GaussDB(DWS) features.

   Input

```
CREATE TABLE IF NOT EXISTS `public`.`runoob_alter_test`(
    `dataType1` int,
    `dataType2` FLOAT(10,2),
    `dataType3` DOUBLE(20,8)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;

ALTER TABLE runoob_alter_test ADD UNIQUE idx_runoob_alter_test_datatype1(dataType1);
ALTER TABLE runoob_alter_test ADD UNIQUE INDEX idx_runoob_alter_test_datatype1(dataType2);
ALTER TABLE runoob_alter_test ADD UNIQUE KEY idx_runoob_alter_test_datatype1(dataType3);

CREATE TABLE IF NOT EXISTS `public`.`runoob_alter_test`(
    `dataType1` int,
    `dataType2` FLOAT(10,2),
    `dataType3` DOUBLE(20,8),
    `dataType4` TEXT NOT NULL,
    `dataType5` YEAR NOT NULL DEFAULT '2018',
    `dataType6` DATETIME NOT NULL DEFAULT '2018-10-12 15:27:33.999999'
)ENGINE=InnoDB DEFAULT CHARSET=utf8;

ALTER TABLE runoob_alter_test ADD CONSTRAINT UNIQUE
idx_runoob_alter_test_datatype1(dataType1);
ALTER TABLE runoob_alter_test ADD CONSTRAINT UNIQUE INDEX
idx_runoob_alter_test_datatype2(dataType2);
ALTER TABLE runoob_alter_test ADD CONSTRAINT UNIQUE KEY
idx_runoob_alter_test_datatype3(dataType3);
ALTER TABLE runoob_alter_test ADD CONSTRAINT constraint_dataType UNIQUE
idx_runoob_alter_test_datatype4(dataType4);
ALTER TABLE runoob_alter_test ADD CONSTRAINT constraint_dataType UNIQUE INDEX
idx_runoob_alter_test_datatype5(dataType5);
ALTER TABLE runoob_alter_test ADD CONSTRAINT constraint_dataType UNIQUE KEY
idx_runoob_alter_test_datatype6(dataType6);
```

   Output

```
CREATE TABLE IF NOT EXISTS "public"."runoob_alter_test"
(
  "datatype1" INTEGER,
  "datatype2" REAL,
  "datatype3" DOUBLE PRECISION
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");

CREATE INDEX "idx_runoob_alter_test_datatype1" ON "public"."runoob_alter_test" ("datatype1");
CREATE INDEX "idx_runoob_alter_test_datatype1" ON "public"."runoob_alter_test" ("datatype2");
CREATE INDEX "idx_runoob_alter_test_datatype1" ON "public"."runoob_alter_test" ("datatype3");

CREATE TABLE IF NOT EXISTS "public"."runoob_alter_test"
(
  "datatype1" INTEGER,
  "datatype2" REAL,
  "datatype3" DOUBLE PRECISION,
  "datatype4" TEXT NOT NULL,
  "datatype5" SMALLINT NOT NULL DEFAULT '2018',
  "datatype6" TIMESTAMP WITHOUT TIME ZONE NOT NULL DEFAULT '2018-10-12 15:27:33.999999'
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");

CREATE INDEX "idx_runoob_alter_test_datatype1" ON "public"."runoob_alter_test" ("datatype1");
CREATE INDEX "idx_runoob_alter_test_datatype2" ON "public"."runoob_alter_test" ("datatype2");
CREATE INDEX "idx_runoob_alter_test_datatype3" ON "public"."runoob_alter_test" ("datatype3");
CREATE INDEX "idx_runoob_alter_test_datatype4" ON "public"."runoob_alter_test" ("datatype4");
```



```
CREATE INDEX "idx_runoob_alter_test_datatype5" ON "public"."runoob_alter_test" ("datatype5");
CREATE INDEX "idx_runoob_alter_test_datatype6" ON "public"."runoob_alter_test" ("datatype6");
```

3. If there is a unique index created by **CREATE INDEX**, DSC will create a normal index based on GaussDB(DWS) features.

   Input

   ```
   CREATE TABLE `public`.`test_index_table01` (
      `TABLE01_ID` INT(11) NOT NULL,
      `TABLE01_THEME` VARCHAR(100) NULL DEFAULT NULL,
      `AUTHOR_NAME` CHAR(10) NULL DEFAULT NULL,
      `AUTHOR_ID` INT(11) NULL DEFAULT NULL,
      `CREATE_TIME` INT NULL DEFAULT NULL,
      PRIMARY KEY(`TABLE01_ID`)
   );
   CREATE UNIQUE INDEX AUTHOR_INDEX ON `test_index_table01`(AUTHOR_ID);
   ```

   Output

   ```
   CREATE TABLE "public"."test_index_table01"
   (
    "table01_id" INTEGER NOT NULL,
    "table01_theme" VARCHAR(400) DEFAULT NULL,
    "author_name" CHAR(40) DEFAULT NULL,
    "author_id" INTEGER DEFAULT NULL,
    "create_time" INTEGER DEFAULT NULL,
    PRIMARY KEY ("table01_id")
   )
    WITH ( ORIENTATION = ROW, COMPRESSION = NO )
    NOCOMPRESS
    DISTRIBUTE BY HASH ("table01_id");
   CREATE INDEX "author_index" ON "public"."test_index_table01" ("author_id");
   ```

4. If CREATE TABLE has multiple unique indexes, during migration, DSC will create all unique indexes as common indexes based on GaussDB(DWS) features.

   Input

   ```
   CREATE TABLE `public`.`test_index_table01` (
      `TABLE01_ID` INT(11) NOT NULL,
      `TABLE01_THEME` VARCHAR(100) NULL DEFAULT NULL,
      `AUTHOR_NAME` CHAR(10) NULL DEFAULT NULL,
      `AUTHOR_ID` INT(11) NULL DEFAULT NULL,
      `CREATE_TIME` INT NULL DEFAULT NULL,
      UNIQUE(`TABLE01_ID`),
      UNIQUE(`AUTHOR_ID`)
   );
   ```

   Output

   ```
   CREATE TABLE "public"."test_index_table01" (
    "table01_id" INTEGER NOT NULL,
    "table01_theme" VARCHAR(400) DEFAULT NULL,
    "author_name" CHAR(40) DEFAULT NULL,
    "author_id" INTEGER DEFAULT NULL,
    "create_time" INTEGER DEFAULT NULL
   ) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH
   ("table01_id");
   CREATE INDEX "idx_test_index_table01_table01_id" ON "public"."test_index_table01"("TABLE01_ID");
   CREATE INDEX "idx_test_index_table01_author_id" ON "public"."test_index_table01"("AUTHOR_ID");
   ```

5. If CREATE TABLE has a unique index but does not have a primary key index, DSC retains the unique index based on GaussDB (DWS) features during migration.

   Input

   ```
   CREATE TABLE `public`.`test_index_table01` (
      `TABLE01_ID` INT(11) NOT NULL,
      `TABLE01_THEME` VARCHAR(100) NULL DEFAULT NULL,
      `AUTHOR_NAME` CHAR(10) NULL DEFAULT NULL,
      `AUTHOR_ID` INT(11) NULL DEFAULT NULL,
   ```



```
  `CREATE_TIME` INT NULL DEFAULT NULL,
  UNIQUE(`AUTHOR_ID`)
);
```

Output

```
CREATE TABLE "public"."test_index_table01" (
 "table01_id" INTEGER NOT NULL,
 "table01_theme" VARCHAR(400) DEFAULT NULL,
 "author_name" CHAR(40) DEFAULT NULL,
 "author_id" INTEGER DEFAULT NULL,
 "create_time" INTEGER DEFAULT NULL,
 UNIQUE ("author_id")
) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH
("author_id");
```

6. If CREATE TABLE has a primary key index, DSC creates all unique indexes as common indexes based on GaussDB(DWS) features during migration.

   Input

```
CREATE TABLE `public`.`test_index_table01` (
  `TABLE01_ID` INT(11) NOT NULL,
  `TABLE01_THEME` VARCHAR(100) NULL DEFAULT NULL,
  `AUTHOR_NAME` CHAR(10) NULL DEFAULT NULL,
  `AUTHOR_ID` INT(11) NULL DEFAULT NULL,
  `CREATE_TIME` INT NULL DEFAULT NULL,
  PRIMARY KEY(`TABLE01_ID`),
  UNIQUE(`AUTHOR_ID`)
);
```

   Output

```
CREATE TABLE "public"."test_index_table01" (
 "table01_id" INTEGER NOT NULL,
 "table01_theme" VARCHAR(400) DEFAULT NULL,
 "author_name" CHAR(40) DEFAULT NULL,
 "author_id" INTEGER DEFAULT NULL,
 "create_time" INTEGER DEFAULT NULL,
 PRIMARY KEY ("table01_id")
) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH
("table01_id");
CREATE INDEX "idx_test_index_table01_author_id" ON "public"."test_index_table01"("AUTHOR_ID");
```

6.5.5.2 Normal and Prefix Indexes

GaussDB(DWS) does not support prefix indexes or inline normal indexes. DSC will replace these indexes with normal indexes based on GaussDB(DWS) features.

1. Inline normal/prefix indexes

   Input

```
CREATE TABLE IF NOT EXISTS `public`.`runoob_dataType_test`
(
 `id` INT PRIMARY KEY AUTO_INCREMENT,
 `name` VARCHAR(128) NOT NULL,
 INDEX index_single(name(10))
);
```

   Output

```
CREATE TABLE IF NOT EXISTS "public"."runoob_datatype_test"
(
 "id" SERIAL PRIMARY KEY,
 "name" VARCHAR(512) NOT NULL
)
 WITH ( ORIENTATION = ROW, COMPRESSION = NO )
 NOCOMPRESS
 DISTRIBUTE BY HASH ("id");
CREATE INDEX "index_single" ON "public"."runoob_datatype_test" USING BTREE ("name");
```


2. Normal/Prefix index created by **ALTER TABLE**

   Input

```
CREATE TABLE `public`.`test_create_table05` (
 `ID` INT(11) NOT NULL AUTO_INCREMENT,
 `USER_ID` INT(20) NOT NULL,
 `USER_NAME` CHAR(20) NULL DEFAULT NULL,
 `DETAIL` VARCHAR(100) NULL DEFAULT NULL,
 PRIMARY KEY (`ID`)
);

ALTER TABLE TEST_CREATE_TABLE05 ADD INDEX USER_NAME_INDEX_02(USER_NAME(10));
```

   Output

```
CREATE TABLE "public"."test_create_table05"
(
  "id" SERIAL NOT NULL,
  "user_id" INTEGER NOT NULL,
  "user_name" CHAR(80) DEFAULT NULL,
  "detail" VARCHAR(400) DEFAULT NULL,
  PRIMARY KEY ("id")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("id");

CREATE INDEX "user_name_index_02" ON "public"."test_create_table05" ("user_name");
```

3. Normal/Prefix index created by **CREATE INDEX**

   Input

```
CREATE TABLE IF NOT EXISTS `public`.`customer`(
    `name` varchar(64) primary key,
    id integer,
    id2 integer
);

CREATE INDEX part_of_name ON customer (name(10));
```

   Output

```
CREATE TABLE IF NOT EXISTS "public"."customer"
(
  "name" VARCHAR(256) PRIMARY KEY,
  "id" INTEGER,
  "id2" INTEGER
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("name");

CREATE INDEX "part_of_name" ON "public"."customer" USING BTREE ("name");
```

6.5.5.3 Hash Indexes

GaussDB(DWS) does not support hash indexes. DSC will replace these indexes with normal indexes based on GaussDB(DWS) features.

1. Inline hash indexes

   Input

```
CREATE TABLE `public`.`test_create_table03` (
    `DEMAND_ID` INT(11) NOT NULL AUTO_INCREMENT,
    `DEMAND_NAME` CHAR(100) NOT NULL,
    `THEME` VARCHAR(200) NULL DEFAULT NULL,
    `SEND_ID` INT(11) NULL DEFAULT NULL,
    `SEND_NAME` CHAR(20) NULL DEFAULT NULL,
    `SEND_TIME` DATETIME NULL DEFAULT NULL,
```



```
    `DEMAND_CONTENT` TEXT NOT NULL,
    PRIMARY KEY(`DEMAND_ID`),
    INDEX CON_INDEX(DEMAND_CONTENT(100)) USING HASH ,
    INDEX SEND_INFO_INDEX USING HASH (SEND_ID,SEND_NAME(10),SEND_TIME)
);
```

Output

```
CREATE TABLE "public"."test_create_table03"
(
  "demand_id" SERIAL NOT NULL,
  "demand_name" CHAR(400) NOT NULL,
  "theme" VARCHAR(800) DEFAULT NULL,
  "send_id" INTEGER DEFAULT NULL,
  "send_name" CHAR(80) DEFAULT NULL,
  "send_time" TIMESTAMP WITHOUT TIME ZONE DEFAULT NULL,
  "demand_content" TEXT NOT NULL,
  PRIMARY KEY ("demand_id")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("demand_id");
CREATE INDEX "con_index" ON "public"."test_create_table03" ("demand_content");
CREATE INDEX "send_info_index" ON "public"."test_create_table03"
("send_id","send_name","send_time");
```

2. Hash index created by **ALTER TABLE**

Input

```
CREATE TABLE IF NOT EXISTS `public`.`runoob_alter_test`(
    `dataType1` int NOT NULL AUTO_INCREMENT,
    `dataType2` FLOAT(10,2),
    PRIMARY KEY(`dataType1`)
)ENGINE=InnoDB DEFAULT CHARSET=utf8;

ALTER TABLE runoob_alter_test ADD KEY alterTable_addKey_indexType(dataType1) USING HASH;
```

Output

```
CREATE TABLE IF NOT EXISTS "public"."runoob_alter_test"
(
  "datatype1" SERIAL NOT NULL,
  "datatype2" REAL,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");

CREATE INDEX "altertable_addkey_indextype" ON "public"."runoob_alter_test" ("datatype1");
```

3. Hash index created by **CREATE INDEX**

Input

```
CREATE TABLE `public`.`test_index_table06` (
    `ID` INT(11) NOT NULL AUTO_INCREMENT,
    `FNAME` VARCHAR(30) NOT NULL,
    `INAME` VARCHAR(30) NOT NULL,
    PRIMARY KEY (`ID`)
);
CREATE INDEX FNAME_INDEX ON TEST_INDEX_TABLE06(FNAME(10)) USING HASH;
CREATE INDEX NAME_01 ON TEST_INDEX_TABLE06(FNAME(10),INAME(10)) USING HASH;
```

Output

```
CREATE TABLE "public"."test_index_table06"
(
  "id" SERIAL NOT NULL,
  "fname" VARCHAR(120) NOT NULL,
  "iname" VARCHAR(120) NOT NULL,
  PRIMARY KEY ("id")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
```



```
  NOCOMPRESS
  DISTRIBUTE BY HASH ("id");
CREATE INDEX "fname_index" ON "public"."test_index_table06" ("fname");
CREATE INDEX "name_01" ON "public"."test_index_table06" ("fname","iname");
```

6.5.5.4 B-tree Indexes

GaussDB(DWS) supports B-tree indexes, but the position of the **USING BTREE** keyword in a statement is different from that in MySQL. DSC will perform adaptation based on GaussDB(DWS) features during migration.

1. Inline B-tree index

   Input

   ```
   CREATE TABLE `public`.`test_create_table03` (
      `DEMAND_ID` INT(11) NOT NULL AUTO_INCREMENT,
      `DEMAND_NAME` CHAR(100) NOT NULL,
      `THEME` VARCHAR(200) NULL DEFAULT NULL,
      `SEND_ID` INT(11) NULL DEFAULT NULL,
      `SEND_NAME` CHAR(20) NULL DEFAULT NULL,
      `SEND_TIME` DATETIME NULL DEFAULT NULL,
      `DEMAND_CONTENT` TEXT NOT NULL,
      PRIMARY KEY(`DEMAND_ID`),
      INDEX THEME_INDEX(THEME) USING BTREE,
      INDEX NAME_INDEX USING BTREE (SEND_NAME(10))
   );
   ```

   Output

   ```
   CREATE TABLE "public"."test_create_table03"
   (
    "demand_id" SERIAL NOT NULL,
    "demand_name" CHAR(400) NOT NULL,
    "theme" VARCHAR(800) DEFAULT NULL,
    "send_id" INTEGER DEFAULT NULL,
    "send_name" CHAR(80) DEFAULT NULL,
    "send_time" TIMESTAMP WITHOUT TIME ZONE DEFAULT NULL,
    "demand_content" TEXT NOT NULL,
    PRIMARY KEY ("demand_id")
   )
    WITH ( ORIENTATION = ROW, COMPRESSION = NO )
    NOCOMPRESS
    DISTRIBUTE BY HASH ("demand_id");
   CREATE INDEX "theme_index" ON "public"."test_create_table03" USING BTREE ("theme");
   CREATE INDEX "name_index" ON "public"."test_create_table03" USING BTREE ("send_name");
   ```

2. B-tree index created by **ALTER TABLE**

   Input

   ```
   CREATE TABLE IF NOT EXISTS `public`.`runoob_alter_test`(
      `dataType1` int NOT NULL AUTO_INCREMENT,
      `dataType2` FLOAT(10,2),
      PRIMARY KEY(`dataType1`)
   );

   ALTER TABLE runoob_alter_test ADD KEY alterTable_addKey_indexType (dataType1) USING BTREE;
   ```

   Output

   ```
   CREATE TABLE IF NOT EXISTS "public"."runoob_alter_test"
   (
    "datatype1" SERIAL NOT NULL,
    "datatype2" REAL,
    PRIMARY KEY ("datatype1")
   )
    WITH ( ORIENTATION = ROW, COMPRESSION = NO )
    NOCOMPRESS
    DISTRIBUTE BY HASH ("datatype1");

   CREATE INDEX "altertable_addkey_indextype" ON "public"."runoob_alter_test" ("datatype1");
   ```


3. B-tree index created by **CREATE INDEX**

   Input

```
CREATE TABLE `public`.`test_index_table05` (
    `ID` INT(11) NOT NULL AUTO_INCREMENT,
    `USER_ID` INT(20) NOT NULL,
    `USER_NAME` CHAR(20) NULL DEFAULT NULL,
    `DETAIL` VARCHAR(100) NULL DEFAULT NULL,
    PRIMARY KEY (`ID`)
);
CREATE UNIQUE INDEX USER_ID_INDEX USING BTREE ON TEST_INDEX_TABLE05(USER_ID);
CREATE INDEX USER_NAME_INDEX USING BTREE ON TEST_INDEX_TABLE05(USER_NAME(10));
CREATE INDEX DETAIL_INDEX ON TEST_INDEX_TABLE05(DETAIL(50)) USING BTREE;
CREATE INDEX USER_INFO_INDEX USING BTREE ON
TEST_INDEX_TABLE05(USER_ID,USER_NAME(10));
```

   Output

```
CREATE TABLE "public"."test_index_table05"
(
  "id" SERIAL NOT NULL,
  "user_id" INTEGER NOT NULL,
  "user_name" CHAR(80) DEFAULT NULL,
  "detail" VARCHAR(400) DEFAULT NULL,
  PRIMARY KEY ("id")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("id");
CREATE INDEX "user_id_index" ON "public"."test_index_table05" ("user_id");
CREATE INDEX "user_name_index" ON "public"."test_index_table05" USING BTREE ("user_name");
CREATE INDEX "detail_index" ON "public"."test_index_table05" USING BTREE ("detail");
CREATE INDEX "user_info_index" ON "public"."test_index_table05" USING BTREE
("user_id","user_name");
```

6.5.5.5 Spatial Indexes

GaussDB(DWS) does not support spatial indexes. DSC will perform adaptation based on GaussDB(DWS) features during migration.

1. Inline spatial index

   Input

```
CREATE TABLE `public`.`test_create_table04` (
    `ID` INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    `A` POINT NOT NULL,
    `B` POLYGON NOT NULL,
    `C` GEOMETRYCOLLECTION NOT NULL,
    `D` LINESTRING NOT NULL,
    `E` MULTILINESTRING NOT NULL,
    `F` MULTIPOINT NOT NULL,
    `G` MULTIPOLYGON NOT NULL,
    SPATIAL INDEX A_INDEX(A),
    SPATIAL INDEX B_INDEX(B),
    SPATIAL INDEX C_INDEX(C),
    SPATIAL KEY D_INDEX(D),
    SPATIAL KEY E_INDEX(E),
    SPATIAL KEY F_INDEX(F),
    SPATIAL INDEX G_INDEX(G)
);
```

   Output

```
CREATE TABLE "public"."test_create_table04"
(
  "id" SERIAL NOT NULL PRIMARY KEY,
  "a" POINT NOT NULL,
  "b" POLYGON NOT NULL,
  "c" GEOMETRYCOLLECTION NOT NULL,
```



```
  "d" POLYGON NOT NULL,
  "e" BOX NOT NULL,
  "f" BOX NOT NULL,
  "g" POLYGON NOT NULL
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("id");
CREATE INDEX "a_index" ON "public"."test_create_table04" USING GIST ("a");
CREATE INDEX "b_index" ON "public"."test_create_table04" USING GIST ("b");
CREATE INDEX "c_index" ON "public"."test_create_table04" USING GIST ("c");
CREATE INDEX "d_index" ON "public"."test_create_table04" USING GIST ("d");
CREATE INDEX "e_index" ON "public"."test_create_table04" USING GIST ("e");
CREATE INDEX "f_index" ON "public"."test_create_table04" USING GIST ("f");
CREATE INDEX "g_index" ON "public"."test_create_table04" USING GIST ("g");
```

2. Spatial index created by **ALTER TABLE**

   Input

```
CREATE TABLE `public`.`test_create_table04` (
 `ID` INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
 `A` POINT NOT NULL,
 `B` POLYGON NOT NULL,
 `C` GEOMETRYCOLLECTION NOT NULL,
 `D` LINESTRING NOT NULL,
 `E` MULTILINESTRING NOT NULL,
 `F` MULTIPOINT NOT NULL,
 `G` MULTIPOLYGON NOT NULL
);

ALTER TABLE `test_create_table04` ADD SPATIAL INDEX A_INDEX(A);
ALTER TABLE `test_create_table04` ADD SPATIAL INDEX E_INDEX(E) USING BTREE;
```

   Output

```
CREATE TABLE "public"."test_create_table04"
(
  "id" SERIAL NOT NULL PRIMARY KEY,
  "a" POINT NOT NULL,
  "b" POLYGON NOT NULL,
  "c" GEOMETRYCOLLECTION NOT NULL,
  "d" POLYGON NOT NULL,
  "e" BOX NOT NULL,
  "f" BOX NOT NULL,
  "g" POLYGON NOT NULL
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("id");

CREATE INDEX "a_index" ON "public"."test_create_table04" USING GIST ("a");
CREATE INDEX "e_index" ON "public"."test_create_table04" USING GIST ("e");
```

3. Spatial index created by **CREATE INDEX**

   Input

```
CREATE TABLE `public`.`test_create_table04` (
    `ID` INT(11) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    `A` POINT NOT NULL,
    `B` POLYGON NOT NULL,
    `C` GEOMETRYCOLLECTION NOT NULL,
    `D` LINESTRING NOT NULL,
    `E` MULTILINESTRING NOT NULL,
    `F` MULTIPOINT NOT NULL,
    `G` MULTIPOLYGON NOT NULL
);

CREATE SPATIAL INDEX A_INDEX ON `test_create_table04`(A);
```

   Output


```
CREATE TABLE "public"."test_create_table04"
(
  "id" SERIAL NOT NULL PRIMARY KEY,
  "a" POINT NOT NULL,
  "b" POLYGON NOT NULL,
  "c" GEOMETRYCOLLECTION NOT NULL,
  "d" POLYGON NOT NULL,
  "e" BOX NOT NULL,
  "f" BOX NOT NULL,
  "g" POLYGON NOT NULL
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("id");

CREATE INDEX "a_index" ON "public"."test_create_table04" USING GIST ("a");
```

6.5.5.6 Full-Text Indexes

GaussDB(DWS) does not support full-text indexes. DSC will perform adaptation based on GaussDB(DWS) features during migration.

1. Inline full-text index

   Input

```
## A.
CREATE TABLE `public`.`test_create_table02` (
    `ID` INT(11) NOT NULL PRIMARY KEY,
    `TITLE` CHAR(255) NOT NULL,
    `CONTENT` TEXT NULL,
    `CREATE_TIME` DATETIME NULL DEFAULT NULL,
     FULLTEXT (`CONTENT`)
);

## B.
CREATE TABLE IF NOT EXISTS `public`.`runoob_dataType_test`
(
  `id` INT PRIMARY KEY AUTO_INCREMENT,
  `name` VARCHAR(128) NOT NULL,
  FULLTEXT INDEX (name)
);

## C.
CREATE TABLE IF NOT EXISTS `public`.`runoob_dataType_test`
(
  `id` INT PRIMARY KEY AUTO_INCREMENT,
  `name` VARCHAR(128) NOT NULL,
  FULLTEXT INDEX (name ASC)
);
```

   Output

```
-- A.
CREATE TABLE "public"."test_create_table02"
(
  "id" INTEGER NOT NULL PRIMARY KEY,
  "title" CHAR(1020) NOT NULL,
  "content" TEXT,
  "create_time" TIMESTAMP WITHOUT TIME ZONE DEFAULT NULL
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("id");
CREATE INDEX "idx_test_create_table02_content" ON "public"."test_create_table02" USING
GIN(to_tsvector(coalesce("content","")));

-- B.
CREATE TABLE IF NOT EXISTS "public"."runoob_datatype_test"
(
```



```
 "id" SERIAL PRIMARY KEY,
 "name" VARCHAR(512) NOT NULL
)
 WITH ( ORIENTATION = ROW, COMPRESSION = NO )
 NOCOMPRESS
 DISTRIBUTE BY HASH ("id");
CREATE INDEX "idx_runoob_datatype_test_name" ON "public"."runoob_datatype_test" USING
GIN(to_tsvector(coalesce("name",'')));

-- C.
CREATE TABLE IF NOT EXISTS "public"."runoob_datatype_test"
(
 "id" SERIAL PRIMARY KEY,
 "name" VARCHAR(512) NOT NULL
)
 WITH ( ORIENTATION = ROW, COMPRESSION = NO )
 NOCOMPRESS
 DISTRIBUTE BY HASH ("id");
CREATE INDEX "idx_runoob_datatype_test_name" ON "public"."runoob_datatype_test" USING
GIN(to_tsvector(coalesce("name",'')));
```

2. Full-text index created by **ALTER TABLE**

   Input

```
CREATE TABLE `public`.`test_create_table05` (
 `ID` INT(11) NOT NULL AUTO_INCREMENT,
 `USER_ID` INT(20) NOT NULL,
 `USER_NAME` CHAR(20) NULL DEFAULT NULL,
 `DETAIL` VARCHAR(100) NULL DEFAULT NULL,
 PRIMARY KEY (`ID`)
);
ALTER TABLE TEST_CREATE_TABLE05 ADD FULLTEXT INDEX USER_ID_INDEX_02(USER_ID);
ALTER TABLE TEST_CREATE_TABLE05 ADD FULLTEXT USER_NAME_INDEX_02(USER_NAME);
```

   Output

```
CREATE TABLE "public"."test_create_table05"
(
 "id" SERIAL NOT NULL,
 "user_id" INTEGER NOT NULL,
 "user_name" CHAR(80) DEFAULT NULL,
 "detail" VARCHAR(400) DEFAULT NULL,
 PRIMARY KEY ("id")
)
 WITH ( ORIENTATION = ROW, COMPRESSION = NO )
 NOCOMPRESS
 DISTRIBUTE BY HASH ("id");
CREATE INDEX "user_id_index_02" ON "public"."test_create_table05" USING
GIN(to_tsvector(coalesce("user_id",'')));
CREATE INDEX "user_name_index_02" ON "public"."test_create_table05" USING
GIN(to_tsvector(coalesce("user_name",'')));
```

3. Full-text index created by **CREATE INDEX**

   Input

```
CREATE TABLE `public`.`test_index_table02` (
   `ID` INT(11) NOT NULL PRIMARY KEY,
   `TITLE` CHAR(255) NOT NULL,
   `CONTENT` TEXT NULL,
   `CREATE_TIME` INT(10) NULL DEFAULT NULL
);
CREATE FULLTEXT INDEX CON_INDEX ON TEST_INDEX_TABLE02(CONTENT);
```

   Output

```
CREATE TABLE "public"."test_index_table02"
(
 "id" INTEGER NOT NULL PRIMARY KEY,
 "title" CHAR(1020) NOT NULL,
 "content" TEXT,
 "create_time" INTEGER DEFAULT NULL
)
```



```
 WITH ( ORIENTATION = ROW, COMPRESSION = NO )
 NOCOMPRESS
 DISTRIBUTE BY HASH ("id");
CREATE INDEX "con_index" ON "public"."test_index_table02" USING
GIN(to_tsvector(coalesce("content",'')));
```

6.5.5.7 Deleting an Index

MySQL supports both **DROP INDEX** and **ALTER TABLE DROP INDEX** for deleting indexes. DSC will perform adaptation based on GaussDB(DWS) features during migration.

1. DROP INDEX

   Input

```
CREATE TABLE `test_create_table03` (
   `DEMAND_ID` INT(11) NOT NULL,
   `DEMAND_NAME` CHAR(100) NOT NULL,
   `THEME` VARCHAR(200) NULL DEFAULT NULL,
   `SEND_ID` INT(11) NULL DEFAULT NULL,
   `SEND_NAME` CHAR(20) NULL DEFAULT NULL,
   `SEND_TIME` DATETIME NULL DEFAULT NULL,
   `DEMAND_CONTENT` TEXT NOT NULL
)
COLLATE='utf8_general_ci'
ENGINE=InnoDB;

CREATE UNIQUE INDEX DEMAND_NAME_INDEX ON TEST_CREATE_TABLE03(DEMAND_NAME);
DROP INDEX DEMAND_NAME_INDEX ON TEST_CREATE_TABLE03;

CREATE INDEX SEND_ID_INDEX ON TEST_CREATE_TABLE03(SEND_ID);
DROP INDEX SEND_ID_INDEX ON TEST_CREATE_TABLE03;
```

   Output

```
CREATE TABLE "public"."test_create_table03"
(
 "demand_id" INTEGER NOT NULL,
 "demand_name" CHAR(400) NOT NULL,
 "theme" VARCHAR(800) DEFAULT NULL,
 "send_id" INTEGER DEFAULT NULL,
 "send_name" CHAR(80) DEFAULT NULL,
 "send_time" TIMESTAMP WITHOUT TIME ZONE DEFAULT NULL,
 "demand_content" TEXT NOT NULL
)
 WITH ( ORIENTATION = ROW, COMPRESSION = NO )
 NOCOMPRESS
 DISTRIBUTE BY HASH ("demand_id");

CREATE INDEX "demand_name_index" ON "public"."test_create_table03" ("demand_name");
DROP INDEX "public"."demand_name_index" RESTRICT;

CREATE INDEX "send_id_index" ON "public"."test_create_table03" USING BTREE ("send_id");
DROP INDEX "public"."send_id_index" RESTRICT;
```

2. ALTER TABLE DROP INDEX

   Input

```
CREATE TABLE `test_create_table03` (
   `DEMAND_ID` INT(11) NOT NULL,
   `DEMAND_NAME` CHAR(100) NOT NULL,
   `THEME` VARCHAR(200) NULL DEFAULT NULL,
   `SEND_ID` INT(11) NULL DEFAULT NULL,
   `SEND_NAME` CHAR(20) NULL DEFAULT NULL,
   `SEND_TIME` DATETIME NULL DEFAULT NULL,
   `DEMAND_CONTENT` TEXT NOT NULL
)
COLLATE='utf8_general_ci'
ENGINE=InnoDB;
```



```
ALTER TABLE TEST_CREATE_TABLE03 ADD UNIQUE INDEX
TEST_CREATE_TABLE03_NAME_INDEX(DEMAND_NAME(50));
ALTER TABLE TEST_CREATE_TABLE03 DROP INDEX TEST_CREATE_TABLE03_NAME_INDEX;
```

Output

```
CREATE TABLE "public"."test_create_table03"
(
  "demand_id" INTEGER NOT NULL,
  "demand_name" CHAR(400) NOT NULL,
  "theme" VARCHAR(800) DEFAULT NULL,
  "send_id" INTEGER DEFAULT NULL,
  "send_name" CHAR(80) DEFAULT NULL,
  "send_time" TIMESTAMP WITHOUT TIME ZONE DEFAULT NULL,
  "demand_content" TEXT NOT NULL
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("demand_id");

CREATE INDEX "test_create_table03_name_index" ON "public"."test_create_table03"
("demand_name");
DROP INDEX "public"."test_create_table03_name_index" RESTRICT;
```

6.5.5.8 Renaming an Index

DSC supports renaming indexes. Prefix a table name to an index name to prevent name conflicts. (Only DDL statements with specific index names can be created. Currently, the name of a renamed index cannot be deleted. Exercise caution when modifying this parameter.)

Modifying the configuration

Open **Table 1 Configuration parameters in the features-mysql.properties file** and change the value of the following parameter to **true**: The default value is **false**, indicating that the file will not be renamed.

```
# Whether to rename an index when creating the index.
table.index.rename=true
```

Input

```
CREATE TABLE IF NOT EXISTS `CUSTOMER`(
   `NAME` VARCHAR(64) PRIMARY KEY,
   ID INTEGER,
   ID2 INTEGER);
CREATE INDEX ID_INDEX USING BTREE ON CUSTOMER (ID);
ALTER TABLE CUSTOMER ADD INDEX ID3_INDEX(ID2);
```

Output

```
CREATE TABLE IF NOT EXISTS "public"."customer" (
   "name" VARCHAR(256) PRIMARY KEY,
   "id" INTEGER,
   "id2" INTEGER) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY
HASH ("name");
CREATE INDEX customer_id_index ON "public"."customer" USING BTREE ("id");
CREATE INDEX customer_id3_index ON "public"."customer" ("id2");
```

6.5.6 Comment

To comment out a single line, MySQL uses # or --, and GaussDB(DWS) uses --. DSC will replace # with -- for commenting out a single line during migration.

Input


```
## comment sample create a table
CREATE TABLE IF NOT EXISTS `public`.`runoob_tbl`(
   `runoob_id` VARCHAR,
   `runoob_title` VARCHAR(100) NOT NULL,
   `runoob_author` VARCHAR(40) NOT NULL,
   `submission_date` VARCHAR
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Output

```
-- comment sample create a table
CREATE TABLE IF NOT EXISTS "public"."runoob_tbl"
(
  "runoob_id" VARCHAR,
  "runoob_title" VARCHAR(400) NOT NULL,
  "runoob_author" VARCHAR(160) NOT NULL,
  "submission_date" VARCHAR
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("runoob_id");
```

6.5.7 Databases

In MySQL, **DATABASE** is a schema object, which is equivalent to the **SCHEMA** of Oracle and GaussDB(DWS). DSC supports the following two scenarios:

1. Database creation

   Input

   ```
   create database IF NOT EXISTS dbname1 CHARACTER SET=utf8 COLLATE=utf8_unicode_ci;
   create database IF NOT EXISTS dbname2;

   drop database if exists dbname1;
   drop database if exists dbname2;
   ```

   Output

   ```
   CREATE SCHEMA "dbname1";
   CREATE SCHEMA "dbname2";

   DROP SCHEMA IF EXISTS "dbname1";
   DROP SCHEMA IF EXISTS "dbname2";
   ```

2. Database use

   Input

   ```
   drop database if exists test;
   create database if not exists test;
   use test;
   ```

   Output

   ```
   DROP SCHEMA IF EXISTS "test";
   CREATE SCHEMA "test";
   SET CURRENT_SCHEMA = "test";
   ```

6.5.8 Data Manipulation Language (DML)

6.5.8.1 INSERT

In MySQL, **INSERT** allows the following keywords: **HIGH_PRIORITY**, **LOW_PRIORITY**, **PARTITION**, **DELAYED**, **IGNORE**, **VALUES**, and **ON DUPLICATE KEY UPDATE**.

HIGH_PRIORITY

MySQL uses **HIGH_PRIORITY** will override the effect of the **LOW_PRIORITY** option.

Input

```
# HIGH_PRIORITY
INSERT HIGH_PRIORITY INTO  exmp_tb2 VALUES(100, 12.3, 'cheap', '2018-11-11');
INSERT HIGH_PRIORITY INTO  exmp_tb2 VALUES(DEFAULT, 128.23, 'nice', '2018-10-11');
INSERT HIGH_PRIORITY INTO  exmp_tb2 VALUES(DEFAULT, DEFAULT, 'nice', '2018-12-14');
INSERT HIGH_PRIORITY INTO  exmp_tb2 VALUES(DEFAULT, DEFAULT, 'nice', DEFAULT);
INSERT HIGH_PRIORITY INTO  exmp_tb2 (tb2_id, tb2_price) VALUES(DEFAULT, DEFAULT);
INSERT HIGH_PRIORITY INTO  exmp_tb2 (tb2_id, tb2_price, tb2_note) VALUES(DEFAULT, DEFAULT,
DEFAULT);
INSERT HIGH_PRIORITY INTO  exmp_tb2 (tb2_id, tb2_price , tb2_note) VALUES(DEFAULT, DEFAULT,
DEFAULT);
INSERT HIGH_PRIORITY INTO  exmp_tb2 (tb2_id, tb2_price, tb2_note, tb2_date) VALUES(DEFAULT,
DEFAULT, DEFAULT, DEFAULT);
```

Output

```
-- HIGH_PRIORITY
INSERT INTO "public"."exmp_tb2"  VALUES (100,12.3,'cheap','2018-11-11');
INSERT INTO "public"."exmp_tb2"  VALUES (DEFAULT,128.23,'nice','2018-10-11');
INSERT INTO "public"."exmp_tb2"  VALUES (DEFAULT,DEFAULT,'nice','2018-12-14');
INSERT INTO "public"."exmp_tb2"  VALUES (DEFAULT,DEFAULT,'nice',DEFAULT);
INSERT INTO "public"."exmp_tb2" ("tb2_id","tb2_price") VALUES (DEFAULT,DEFAULT);
INSERT INTO "public"."exmp_tb2" ("tb2_id","tb2_price","tb2_note") VALUES (DEFAULT,DEFAULT,DEFAULT);
INSERT INTO "public"."exmp_tb2" ("tb2_id","tb2_price","tb2_note") VALUES (DEFAULT,DEFAULT,DEFAULT);
INSERT INTO "public"."exmp_tb2" ("tb2_id","tb2_price","tb2_note","tb2_date") VALUES
(DEFAULT,DEFAULT,DEFAULT,DEFAULT);
```

LOW_PRIORITY

When the **LOW_PRIORITY** modifier is used, execution of **INSERT** is delayed.

Input

```
# LOW_PRIORITY
INSERT LOW_PRIORITY INTO  exmp_tb2 VALUES( DEFAULT, '128.23', 'nice', '2018-10-11');
INSERT LOW_PRIORITY INTO  exmp_tb2 VALUES(DEFAULT, DEFAULT, 'nice', '2018-12-14' );
INSERT LOW_PRIORITY INTO  exmp_tb2 VALUES(DEFAULT, DEFAULT, 'nice', DEFAULT);
INSERT LOW_PRIORITY INTO  exmp_tb2 (tb2_id, tb2_price) VALUES(DEFAULT, DEFAULT);
INSERT LOW_PRIORITY INTO  exmp_tb2 (tb2_id, tb2_price, tb2_note) VALUES(DEFAULT, DEFAULT,
DEFAULT);
```

Output

```
-- LOW_PRIORITY
INSERT INTO "public"."exmp_tb2"  VALUES (DEFAULT,'128.23','nice','2018-10-11');
INSERT INTO "public"."exmp_tb2"  VALUES (DEFAULT,DEFAULT,'nice','2018-12-14');
INSERT INTO "public"."exmp_tb2"  VALUES (DEFAULT,DEFAULT,'nice',DEFAULT);
INSERT INTO "public"."exmp_tb2" ("tb2_id","tb2_price") VALUES (DEFAULT,DEFAULT);
INSERT INTO "public"."exmp_tb2" ("tb2_id","tb2_price","tb2_note") VALUES (DEFAULT,DEFAULT,DEFAULT);
```

PARTITION

When inserting into a partitioned table, you can control which partitions and subpartitions accept new rows.

Input

```
INSERT INTO employees PARTITION(p3) VALUES (19, 'Frank1', 'Williams', 1, 2);
INSERT INTO employees PARTITION(p0) VALUES (4, 'Frank1', 'Williams', 1, 2);
INSERT INTO employees PARTITION(p1) VALUES (9, 'Frank1', 'Williams', 1, 2);
```



```
INSERT INTO employees PARTITION(p2) VALUES (10, 'Frank1', 'Williams', 1, 2);
INSERT INTO employees PARTITION(p2) VALUES (11, 'Frank1', 'Williams', 1, 2);
```

Output

```
INSERT INTO "public"."employees"  VALUES (19,'Frank1','Williams',1,2);
INSERT INTO "public"."employees"  VALUES (4,'Frank1','Williams',1,2);
INSERT INTO "public"."employees"  VALUES (9,'Frank1','Williams',1,2);
INSERT INTO "public"."employees"  VALUES (10,'Frank1','Williams',1,2);
INSERT INTO "public"."employees"  VALUES (11,'Frank1','Williams',1,2);
```

DELAYED

NOTICE

In MySQL 5.7, the **DELAYED** keyword is recognized but ignored by the server.

Input

```
# DELAYED
INSERT DELAYED INTO  exmp_tb2 VALUES(99, 15.68, 'good', '2018-11-12');
INSERT DELAYED INTO  exmp_tb2 VALUES(80, 12.3, 'cheap', '2018-11-11');
INSERT DELAYED INTO  exmp_tb2 VALUES(DEFAULT, 128.23, 'nice', '2018-10-11');
INSERT DELAYED INTO  exmp_tb2 VALUES(DEFAULT, DEFAULT, 'nice', '2018-12-14');
INSERT DELAYED INTO  exmp_tb2 VALUES(DEFAULT, DEFAULT, 'nice', DEFAULT);
INSERT DELAYED INTO  exmp_tb2 (tb2_id, tb2_price) VALUES(DEFAULT, DEFAULT);
INSERT DELAYED INTO  exmp_tb2 (tb2_id, tb2_price, tb2_note) VALUES(DEFAULT, DEFAULT, DEFAULT);
INSERT DELAYED INTO  exmp_tb2 (tb2_id, tb2_price, tb2_note, tb2_date) VALUES(DEFAULT, DEFAULT,
DEFAULT, DEFAULT);
```

Output

```
-- DELAYED
INSERT INTO "public"."exmp_tb2"  VALUES (99,15.68,'good','2018-11-12');
INSERT INTO "public"."exmp_tb2"  VALUES (80,12.3,'cheap','2018-11-11');
INSERT INTO "public"."exmp_tb2"  VALUES (DEFAULT,128.23,'nice','2018-10-11');
INSERT INTO "public"."exmp_tb2"  VALUES (DEFAULT,DEFAULT,'nice','2018-12-14');
INSERT INTO "public"."exmp_tb2"  VALUES (DEFAULT,DEFAULT,'nice',DEFAULT);
INSERT INTO "public"."exmp_tb2" ("tb2_id","tb2_price") VALUES (DEFAULT,DEFAULT);
INSERT INTO "public"."exmp_tb2" ("tb2_id","tb2_price","tb2_note") VALUES (DEFAULT,DEFAULT,DEFAULT);
INSERT INTO "public"."exmp_tb2" ("tb2_id","tb2_price","tb2_note","tb2_date") VALUES
(DEFAULT,DEFAULT,DEFAULT,DEFAULT);
```

IGNORE

When the **IGNORE** modifier is used, errors that occur during **INSERT** execution are ignored.

Input

```
# New data will be ignored if there is duplicate in the table.
INSERT IGNORE INTO  exmp_tb2 VALUES(189, '189.23','nice','2017-11-12');
INSERT IGNORE INTO  exmp_tb2 VALUES(130,'189.23','nice','2017-11-12');
INSERT IGNORE INTO  exmp_tb2 VALUES(120,15.68,'good','2018-11-12');
INSERT IGNORE INTO  exmp_tb2 VALUES(DEFAULT,128.23,'nice','2018-10-11');
INSERT IGNORE INTO  exmp_tb2 VALUES(DEFAULT,DEFAULT,'nice','2018-12-14');
INSERT IGNORE INTO  exmp_tb2 VALUES(DEFAULT,DEFAULT,'nice',DEFAULT);
INSERT IGNORE INTO  exmp_tb2 (tb2_id,tb2_price) VALUES(DEFAULT,DEFAULT);
INSERT IGNORE INTO  exmp_tb2 (tb2_id,tb2_price,tb2_note) VALUES(DEFAULT,DEFAULT,DEFAULT);
INSERT IGNORE INTO  exmp_tb2 (tb2_id,tb2_price,tb2_note,tb2_date)
VALUES(DEFAULT,DEFAULT,DEFAULT,DEFAULT);
```

Output


```
-- New data will be ignored if there is duplicate in the table.
INSERT INTO "public"."exmp_tb2"  VALUES (101,'189.23','nice','2017-11-12');
INSERT INTO "public"."exmp_tb2"  VALUES (130,'189.23','nice','2017-11-12');
INSERT INTO "public"."exmp_tb2"  VALUES (120,15.68,'good','2018-11-12');
INSERT INTO "public"."exmp_tb2"  VALUES (DEFAULT,128.23,'nice','2018-10-11');
INSERT INTO "public"."exmp_tb2"  VALUES (DEFAULT,DEFAULT,'nice','2018-12-14');
INSERT INTO "public"."exmp_tb2"  VALUES (DEFAULT,DEFAULT,'nice',DEFAULT);
INSERT INTO "public"."exmp_tb2" ("tb2_id","tb2_price") VALUES (DEFAULT,DEFAULT);
INSERT INTO "public"."exmp_tb2" ("tb2_id","tb2_price","tb2_note") VALUES (DEFAULT,DEFAULT,DEFAULT);
INSERT INTO "public"."exmp_tb2" ("tb2_id","tb2_price","tb2_note","tb2_date") VALUES
(DEFAULT,DEFAULT,DEFAULT,DEFAULT);
```

VALUES

INSERT statements that use the **VALUES** syntax can insert multiple lines, separated by commas.

Input

```
INSERT INTO  exmp_tb1 (tb1_name,tb1_gender,tb1_address,tb1_number)
VALUES('David','male','NewYork','01015827875'),('Rachel','female','NewYork','01015827749'),
('Monica','female','NewYork','010158996743');
```

Output

```
INSERT INTO "public"."exmp_tb1" ("tb1_name","tb1_gender","tb1_address","tb1_number") VALUES
('David','male','NewYork','01015827875');
INSERT INTO "public"."exmp_tb1" ("tb1_name","tb1_gender","tb1_address","tb1_number") VALUES
('Rachel','female','NewYork','01015827749');
INSERT INTO "public"."exmp_tb1" ("tb1_name","tb1_gender","tb1_address","tb1_number") VALUES
('Monica','female','NewYork','010158996743');
```

ON DUPLICATE KEY UPDATE

INSERT uses the **ON DUPLICATE KEY UPDATE** clause to update existing rows.

Input

```
# ON DUPLICATE KEY UPDATE (If new data will cause a duplicate value in the primary/unique key,
UPDATE will work. Otherwise, INSERT will work.)
INSERT INTO  exmp_tb2(tb2_id,tb2_price) VALUES(3,12.3) ON DUPLICATE KEY UPDATE tb2_price=12.3;
INSERT INTO  exmp_tb2(tb2_id,tb2_price) VALUES(4,12.3) ON DUPLICATE KEY UPDATE tb2_price=12.3;
INSERT INTO  exmp_tb2(tb2_id,tb2_price,tb2_note) VALUES(10,DEFAULT,DEFAULT) ON DUPLICATE KEY
UPDATE tb2_price=66.6;
INSERT INTO  exmp_tb2(tb2_id,tb2_price,tb2_note,tb2_date) VALUES(11,DEFAULT,DEFAULT,DEFAULT) ON
DUPLICATE KEY UPDATE tb2_price=66.6;
```

Output

```
-- ON DUPLICATE KEY UPDATE (If new data will cause a duplicate value in the primary/unique key,
UPDATE will work. Otherwise, INSERT will work.)
INSERT INTO "public"."exmp_tb2" ("tb2_id","tb2_price") VALUES (3,12.3);
INSERT INTO "public"."exmp_tb2" ("tb2_id","tb2_price") VALUES (4,12.3);
INSERT INTO "public"."exmp_tb2" ("tb2_id","tb2_price","tb2_note") VALUES (10,DEFAULT,DEFAULT);
INSERT INTO "public"."exmp_tb2" ("tb2_id","tb2_price","tb2_note","tb2_date") VALUES
(11,DEFAULT,DEFAULT,DEFAULT);
```

SET

MySQL INSERT...SET statement inserts rows based on explicitly specified values.

Input

```
# INSERT INTO  SET (Data records can be inserted specially. One record can be inserted at a time and batch
insertion is not supported.)
INSERT INTO  exmp_tb2 SET tb2_price=56.1,tb2_note='unbelievable',tb2_date='2018-11-13';
```



```
INSERT INTO  exmp_tb2 SET tb2_price=99.9,tb2_note='perfect',tb2_date='2018-10-13';
INSERT INTO  exmp_tb2 SET tb2_id=9,tb2_price=99.9,tb2_note='perfect',tb2_date='2018-10-13';
```

Output

```
-- INSERT INTO  SET (Data records can be inserted specially. One record can be inserted at a time, and batch insertion is not supported.)
INSERT INTO "public"."exmp_tb2" ("tb2_price","tb2_note","tb2_date") VALUES (56.1,'unbelievable','2018-11-13');
INSERT INTO "public"."exmp_tb2" ("tb2_price","tb2_note","tb2_date") VALUES (99.9,'perfect','2018-10-13');
INSERT INTO "public"."exmp_tb2" ("tb2_id","tb2_price","tb2_note","tb2_date") VALUES (9,99.9,'perfect','2018-10-13');
```

6.5.8.2 UPDATE

In MySQL, **UPDATE** allows the following keywords: **LOW_PRIORITY**, **ORDER BY**, **LIMIT**, and **IGNORE**.

LOW_PRIORITY

With the **LOW_PRIORITY** modifier, execution of **UPDATE** is delayed.

Input

```
# LOW_PRIORITY
UPDATE LOW_PRIORITY employees SET department_id=2;
```

Output

```
-- LOW_PRIORITY
UPDATE "public"."employees" SET "department_id" = 2;
```

ORDER BY

In MySQL, if an **UPDATE** statement includes an **ORDER BY** clause, the rows will be updated in the order specified by the clause.

Input

```
# ORDER BY
UPDATE employees SET department_id=department_id+1 ORDER BY id;
```

Output

```
-- ORDER BY
UPDATE "public"."employees" SET "department_id" = department_id+1;
```

LIMIT

UPDATE LIMIT syntax can be used to limit the scope. A clause is a limit on row matching. As long as the rows that satisfy the clause are found, the statements will stop, regardless of whether they have actually changed.

Input

```
# LIMIT
UPDATE  employees SET department_id=department_id+1   LIMIT 3 ;
UPDATE  employees SET department_id=department_id+1   LIMIT 3 , 10 ;

# LIMIT + OFFSET
UPDATE  employees SET department_id=department_id+1   LIMIT 3   OFFSET 2;

# LIMIT + ORDER BY
UPDATE  employees SET department_id=department_id+1 ORDER BY fname  LIMIT 3 ;
```



```
# LIMIT + WHERE + ORDER BY
UPDATE  employees SET department_id=department_id+1 WHERE id<5 ORDER BY  fname  LIMIT 3 ;

# LIMIT + WHERE + ORDER BY + OFFSET
UPDATE  employees SET department_id=department_id+1 WHERE id<5 ORDER BY  fname  LIMIT 3 OFFSET
2 ;
```

Output

```
-- LIMIT
UPDATE "public"."employees" SET "department_id" = department_id+1;
UPDATE "public"."employees" SET "department_id" = department_id+1;

-- LIMIT + OFFSET
UPDATE "public"."employees" SET "department_id" = department_id+1;

-- LIMIT + ORDER BY
UPDATE "public"."employees" SET "department_id" = department_id+1;

-- LIMIT + WHERE + ORDER BY
UPDATE "public"."employees" SET "department_id" = department_id+1 WHERE id<5;

-- LIMIT + WHERE + ORDER BY + OFFSET
UPDATE "public"."employees" SET "department_id" = department_id+1 WHERE id<5;
```

IGNORE

With the **IGNORE** modifier, the **UPDATE** statement does not abort even if errors occur during execution.

Input

```
# IGNORE
UPDATE IGNORE employees SET department_id=3;
```

Output

```
-- IGNORE
UPDATE "public"."employees" SET "department_id" = 3;
```

6.5.8.3 REPLACE

In MySQL, **REPLACE** allows the following keywords: **LOW_PRIORITY**, **PARTITION**, **DELAYED**, **VALUES**, and **SET**. The following examples are temporary migration solutions only.

NOTE

REPLACE works exactly like **INSERT**, except that if an old row in the table has the same value as a new row for a primary key or unique index, the old row is deleted before the new row is inserted.

LOW_PRIORITY

MySQL REPLACE supports the use of **LOW_PRIORITY**, which is converted by DSC.

Input

```
# LOW_PRIORITY
Replace LOW_PRIORITY INTO  exmp_tb2 VALUES(1, '128.23', 'nice', '2018-10-11 19:00:00');
Replace LOW_PRIORITY INTO  exmp_tb2 VALUES(2, DEFAULT, 'nice', '2018-12-14 19:00:00' );
Replace LOW_PRIORITY INTO  exmp_tb2 VALUES(3, DEFAULT, 'nice', DEFAULT);
Replace LOW_PRIORITY INTO  exmp_tb2 (tb2_id, tb2_price) VALUES(5, DEFAULT);
Replace LOW_PRIORITY INTO  exmp_tb2 (tb2_id, tb2_price, tb2_note) VALUES(4, DEFAULT, DEFAULT);
```


Output

```
-- LOW_PRIORITY
INSERT INTO "public"."exmp_tb2"  VALUES (1,'128.23','nice','2018-10-11 19:00:00');
INSERT INTO "public"."exmp_tb2"  VALUES (2,DEFAULT,'nice','2018-12-14 19:00:00');
INSERT INTO "public"."exmp_tb2"  VALUES (3,DEFAULT,'nice',DEFAULT);
INSERT INTO "public"."exmp_tb2" ("tb2_id","tb2_price") VALUES (5,DEFAULT);
INSERT INTO "public"."exmp_tb2" ("tb2_id","tb2_price","tb2_note") VALUES (4,DEFAULT,DEFAULT);
```

PARTITION

MySQL REPLACE supports explicit partitioning selection using the **PARTITION** keyword and a comma-separated name list for partitions, subpartitions, or both.

Input

```
replace INTO employees PARTITION(p3) VALUES (19, 'Frank1', 'Williams', 1, 2);
replace INTO employees PARTITION(p0) VALUES (4, 'Frank1', 'Williams', 1, 2);
replace INTO employees PARTITION(p1) VALUES (9, 'Frank1', 'Williams', 1, 2);
replace INTO employees PARTITION(p2) VALUES (10, 'Frank1', 'Williams', 1, 2);
replace INTO employees PARTITION(p2) VALUES (11, 'Frank1', 'Williams', 1, 2);
```

Output

```
INSERT INTO "public"."employees"  VALUES (19,'Frank1','Williams',1,2);
INSERT INTO "public"."employees"  VALUES (4,'Frank1','Williams',1,2);
INSERT INTO "public"."employees"  VALUES (9,'Frank1','Williams',1,2);
INSERT INTO "public"."employees"  VALUES (10,'Frank1','Williams',1,2);
INSERT INTO "public"."employees"  VALUES (11,'Frank1','Williams',1,2);
```

DELAYED

WARNING

- **DELAYED INSERT** and **REPLACE** operations were deprecated in MySQL 5.6. In MySQL 5.7, **DELAYED** was not supported. The server recognizes but ignores the **DELAYED** keyword, handles **REPLACE** as a non-delayed one, and generates an **ER_WARN_LEGACY_SYNTAX_CONVERTED** warning.
- (**REPLACE DELAYED** is no longer supported and the statement is converted to **REPLACE**.)
- The keyword **DELAYED** will be deleted in later versions.

Input

```
#DELAYED INSERT DELAYED works only with MyISAM, MEMORY, ARCHIVE, and BLACKHOLE tables.
#If you execute INSERT DELAYED with another storage engine,
#you will get an error like this: ERROR 1616 (HY000): DELAYED option not supported
Replace DELAYED INTO  exmp_tb2 VALUES(10, 128.23, 'nice', '2018-10-11 19:00:00');
Replace DELAYED INTO  exmp_tb2 VALUES(6, DEFAULT, 'nice', '2018-12-14 19:00:00');
Replace DELAYED INTO  exmp_tb2 VALUES(7, 20, 'nice', DEFAULT);
Replace DELAYED INTO  exmp_tb2 (tb2_id, tb2_price) VALUES(11, DEFAULT);
Replace DELAYED INTO  exmp_tb2 (tb2_id, tb2_price, tb2_note) VALUES(12, DEFAULT, DEFAULT);
Replace DELAYED INTO  exmp_tb2 (tb2_id, tb2_price, tb2_note, tb2_date) VALUES(13, DEFAULT, DEFAULT,
DEFAULT);
```

Output

```
--DELAYED INSERT DELAYED works only with MyISAM, MEMORY, ARCHIVE, and BLACKHOLE tables.
--If you execute INSERT DELAYED with another storage engine,
--you will get an error like this: ERROR 1616 (HY000): DELAYED option not supported.
INSERT INTO "public"."exmp_tb2"  VALUES (10,128.23,'nice','2018-10-11 19:00:00');
```



```
INSERT INTO "public"."exmp_tb2"  VALUES (6,DEFAULT,'nice','2018-12-14 19:00:00');
INSERT INTO "public"."exmp_tb2"  VALUES (7,20,'nice',DEFAULT);
INSERT INTO "public"."exmp_tb2" ("tb2_id","tb2_price") VALUES (11,DEFAULT);
INSERT INTO "public"."exmp_tb2" ("tb2_id","tb2_price","tb2_note") VALUES (12,DEFAULT,DEFAULT);
INSERT INTO "public"."exmp_tb2" ("tb2_id","tb2_price","tb2_note","tb2_date") VALUES
(13,DEFAULT,DEFAULT,DEFAULT);
```

VALUES

MySQL REPLACE supports a statement to insert or delete multiple values, separated by commas.

Input

```
# If data is available, replacement will be performed. Otherwise, INSERT will be performed.
Replace INTO  exmp_tb1 (tb1_id,tb1_name,tb1_gender,tb1_address,tb1_number)
VALUES(17,'David','male','NewYork11','01015827875'),(18,'Rachel','female','NewYork22','01015827749'),
(20,'Monica','female','NewYork','010158996743');
Replace INTO  exmp_tb1 (tb1_id,tb1_name,tb1_gender,tb1_address,tb1_number)
VALUES(17,'David1','male','NewYork11','01015827875'),(21,'Rachel','female','NewYork22','01015827749'),
(22,'Monica','female','NewYork','010158996743');
Replace INTO  exmp_tb1 (tb1_id,tb1_name,tb1_gender,tb1_address,tb1_number,tb1_date)
VALUES(17,'David2',DEFAULT,'NewYork11','01015827875',DEFAULT),
(18,'Rachel','female',DEFAULT,'01015827749','2018-12-14 10:44:20'),
(DEFAULT,'Monica','female',DEFAULT,DEFAULT,'2018-12-14 10:44:20');
Replace INTO  exmp_tb1 VALUES(DEFAULT,'David',DEFAULT,'NewYork11','01015827875',DEFAULT),
(18,'Rachel','female',DEFAULT,'01015827749','2018-12-14 10:44:20'),
(DEFAULT,'Monica','female',DEFAULT,DEFAULT,'2018-12-14 10:44:20');
```

Output

```
-- If data is available, replacement will be performed. Otherwise, INSERT will be performed.
INSERT INTO "public"."exmp_tb1" ("tb1_id","tb1_name","tb1_gender","tb1_address","tb1_number") VALUES
(17,'David','male','NewYork11','01015827875');
INSERT INTO "public"."exmp_tb1" ("tb1_id","tb1_name","tb1_gender","tb1_address","tb1_number") VALUES
(18,'Rachel','female','NewYork22','01015827749');
INSERT INTO "public"."exmp_tb1" ("tb1_id","tb1_name","tb1_gender","tb1_address","tb1_number") VALUES
(20,'Monica','female','NewYork','010158996743');
INSERT INTO "public"."exmp_tb1" ("tb1_id","tb1_name","tb1_gender","tb1_address","tb1_number") VALUES
(17,'David1','male','NewYork11','01015827875');
INSERT INTO "public"."exmp_tb1" ("tb1_id","tb1_name","tb1_gender","tb1_address","tb1_number") VALUES
(21,'Rachel','female','NewYork22','01015827749');
INSERT INTO "public"."exmp_tb1" ("tb1_id","tb1_name","tb1_gender","tb1_address","tb1_number") VALUES
(22,'Monica','female','NewYork','010158996743');
INSERT INTO "public"."exmp_tb1"
("tb1_id","tb1_name","tb1_gender","tb1_address","tb1_number","tb1_date") VALUES
(17,'David2',DEFAULT,'NewYork11','01015827875',DEFAULT);
INSERT INTO "public"."exmp_tb1"
("tb1_id","tb1_name","tb1_gender","tb1_address","tb1_number","tb1_date") VALUES
(18,'Rachel','female',DEFAULT,'01015827749','2018-12-14 10:44:20');
INSERT INTO "public"."exmp_tb1"
("tb1_id","tb1_name","tb1_gender","tb1_address","tb1_number","tb1_date") VALUES
(DEFAULT,'Monica','female',DEFAULT,DEFAULT,'2018-12-14 10:44:20');
INSERT INTO "public"."exmp_tb1"  VALUES (DEFAULT,'David',DEFAULT,'NewYork11','01015827875',DEFAULT);
INSERT INTO "public"."exmp_tb1"  VALUES (18,'Rachel','female',DEFAULT,'01015827749','2018-12-14
10:44:20');
INSERT INTO "public"."exmp_tb1"  VALUES (DEFAULT,'Monica','female',DEFAULT,DEFAULT,'2018-12-14
10:44:20');
```

SET

MySQL REPLACE supports the use of **SET** settings, which DSC will convert.

Input

```
replace INTO `runoob_datatype_test` VALUES (100, 100, 100, 0, 1);
replace INTO `runoob_datatype_test` VALUES (100.23, 100.25, 100.26, 0.12,1.5);
```



```
replace INTO `runoob_datatype_test` (dataType_numeric,dataType_numeric1) VALUES (100.23, 100.25);
replace INTO `runoob_datatype_test` (dataType_numeric,dataType_numeric1,dataType_numeric2) VALUES
(100.23, 100.25, 2.34);
replace into runoob_datatype_test set dataType_numeric=23.1, dataType_numeric4 = 25.12 ;
```

Output

```
INSERT INTO "public"."runoob_datatype_test"  VALUES (100,100,100,0,1);
INSERT INTO "public"."runoob_datatype_test"  VALUES (100.23,100.25,100.26,0.12,1.5);
INSERT INTO "public"."runoob_datatype_test" ("datatype_numeric","datatype_numeric1") VALUES
(100.23,100.25);
INSERT INTO "public"."runoob_datatype_test"
("datatype_numeric","datatype_numeric1","datatype_numeric2") VALUES (100.23,100.25,2.34);
INSERT INTO "public"."runoob_datatype_test" ("datatype_numeric","datatype_numeric4") VALUES
(23.1,25.12);
```

6.5.8.4 Quotation Marks

Single Quotation Mark (')

Aliases in MySQL contain single quotation marks, which are not supported by GaussDB(DWS). DSC changes them to double quotation marks during migration.

Input

```
select name as 'mingzi' from t1;
```

Output

```
SELECT
  name AS "mingzi"
FROM
  t1;
```

Backquote (`)

Aliases and column names in MySQL contain backquotes, which are not supported by GaussDB(DWS). DSC changes them to double quotation marks during migration.

Input

```
select `name` as `mingzi` from t1;
```

Output

```
SELECT
  "name" AS "mingzi"
FROM
  t1;
```

Double Quotation Mark (")

In MySQL, constant strings are enclosed in double quotation marks, which are not supported by GaussDB(DWS). DSC changes them to single quotation marks during migration.

Input

```
select name from t1 where name = "test2";
```

Output


```
SELECT
  name
FROM
  t1
WHERE
  name = 'test2';
```

6.5.8.5 INTERVAL

In MySQL, the interval expression format is **INTERVAL N**, which is not supported by GaussDB(DWS) and needs to be converted to **INTERVAL'N'**.

Input

```
SELECT CURRENT_TIME() - INTERVAL 4 DAY;
SELECT NOW() - INTERVAL 5 HOUR;
SELECT CURRENT_TIME() - INTERVAL '4' DAY;
SELECT NOW() - INTERVAL '5' HOUR;
SELECT CURRENT_TIME() - INTERVAL "4" DAY;
SELECT NOW() - INTERVAL "5" HOUR;
```

Output

```
SELECT  (CURRENT_TIME () - INTERVAL '4' DAY);
SELECT  (NOW () - INTERVAL '5' HOUR);
SELECT  (CURRENT_TIME () - INTERVAL '4' DAY);
SELECT  (NOW () - INTERVAL '5' HOUR);
SELECT  (CURRENT_TIME () - INTERVAL '4' DAY);
SELECT  (NOW () - INTERVAL '5' HOUR);
```

6.5.8.6 Division Expressions

In MySQL, in a division expression, if the divisor is 0, null is returned. GaussDB(DWS) reports an error. Therefore, the division expression is converted by adding an if condition expression.

Input

```
select sum(c1) / c2 as result from table_t1;
select sum(c1) / count (c3/c4) as result from table_t1;
```

Output

```
SELECT  (if (c2 = 0, null, sum(c1) / c2)) AS "result" FROM  table_t1;
SELECT  (if (count(if (c4 = 0, null, c3 / c4)) = 0, null, sum(c1) / count(if (c4 = 0, null, c3 / c4)))) AS "result"
FROM  table_t1;
```

6.5.8.7 GROUP BY Conversion

During MySQL/ADB group query, non-group columns can be queried. During GaussDB(DWS) group query, only group columns and aggregate functions can be queried, and if non-group columns are queried, an error will be reported. Therefore, GROUP BY in GaussDB(DWS) is changed to allow querying on non-group columns.

Input

```
SELECT e.department_id, department_name, ROUND(AVG(salary), 0) avg_salary FROM employees e JOIN
departments d on e.department_id = d.department_id GROUP BY department_name ORDER BY
department_name;
```

Output

```
SELECT
  e.department_id,
```



```
  department_name,
  ROUND (AVG(salary), 0) AS "avg_salary"
FROM
  employees "e"
  JOIN departments "d" ON e.department_id = d.department_id
GROUP BY
  department_name,
  1
ORDER BY
  department_name;
```

6.5.8.8 ROLLUP

group by column with rollup in MySQL needs to be converted to **group by rollup (column)** in GaussDB(DWS).

Input

```
select id,product_id,count(1) from czb_account.equity_account_log
where id in (6957343,6957397,6957519,6957541,6957719)
group by 1, 2 with rollup;
```

Output

```
SELECT
  id,
  product_id,
  count(1)
FROM
  czb_account.equity_account_log
WHERE
  id IN (6957343, 6957397, 6957519, 6957541, 6957719)
GROUP BY
  ROLLUP(1, 2);
```

6.5.9 Transaction Management and Database Management

6.5.9.1 Transaction Management

TRANSACTION

DSC will perform adaptation based on GaussDB(DWS) features during MySQL transaction statement migration.

Input

```
## Each statement applies only to the next single transaction performed within the session.
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET TRANSACTION READ ONLY;
SET TRANSACTION READ WRITE;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED,READ ONLY;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE,READ WRITE;
## Each statement (with the SESSION keyword) applies to all subsequent transactions performed within the current session.
START TRANSACTION;
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;
commit ;
```


Output

```
-- Each statement applies only to the next single transaction performed within the session.
SET LOCAL TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET LOCAL TRANSACTION ISOLATION LEVEL REPEATABLE READ;
SET LOCAL TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
SET LOCAL TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET LOCAL TRANSACTION READ ONLY;
SET LOCAL TRANSACTION READ WRITE;
SET LOCAL TRANSACTION ISOLATION LEVEL READ COMMITTED READ ONLY;
SET LOCAL TRANSACTION ISOLATION LEVEL SERIALIZABLE READ WRITE;
-- Each statement (with the SESSIONkeyword) applies to all subsequent transactions performed within the current session.
START TRANSACTION;
SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION LEVEL SERIALIZABLE;
COMMIT WORK;
```

LOCK

DSC will perform adaptation based on GaussDB(DWS) features during the migration of MySQL table locking statements which are used in transaction processing.

Input

```
## A.
START TRANSACTION;
LOCK TABLES `mt`.`runoob_tbl` WRITE,`mt`.`runoob_tb2` READ;
commit;

## B.
START TRANSACTION;
LOCK TABLES `mt`.`runoob_tbl` WRITE;
commit;

## C.
START TRANSACTION;
LOCK TABLES `mt`.`runoob_tbl` READ,`mt`.`runoob_tbl` AS t1 READ;
commit;
```

Output

```
-- A.
START TRANSACTION;
LOCK TABLE "mt"."runoob_tbl" IN ACCESS EXCLUSIVE MODE;
LOCK TABLE "mt"."runoob_tb2" IN ACCESS SHARE MODE;
COMMIT WORK;

-- B.
START TRANSACTION;
LOCK TABLE "mt"."runoob_tbl" IN ACCESS EXCLUSIVE MODE;
COMMIT WORK;

-- C.
START TRANSACTION;
LOCK TABLE "mt"."runoob_tbl" IN ACCESS SHARE MODE;
COMMIT WORK;
```

6.5.9.2 Database Management

DSC will replace MySQL **SET CHARACTER SET** with **SET SESSION NAMES** during migration. The following table lists the character sets.

Table 6-36 Character set comparison table

MySQL CHARACTER SET	GaussDB(DWS) SESSION NAMES
ASCII	SQL_ASCII
BIG5	BIG5
CP1250	WIN1250
CP1251	WIN1251
CP1256	WIN1256
CP1257	WIN1257
CP932	SJIS
EUCJPMS	EUC_JP
EUCKR	EUC_KR
GB2312	GB18030
GBK	GBK
GREEK	ISO_8859_7
HEBREW	ISO_8859_8
KOI8R	KOI8R
KOI8U	KOI8U
LATIN1	LATIN1
LATIN2	LATIN2
LATIN5	LATIN5
LATIN7	LATIN7
SJIS	SJIS
SWE7	UTF8
TIS620	WIN874
UTF8	UTF8
UTF8MB4	UTF8

Input

```
SET CHARACTER SET 'ASCII';
SET CHARACTER SET 'BIG5';
SET CHARACTER SET 'CP1250';
SET CHARACTER SET 'CP1251';
SET CHARACTER SET 'CP1256';
SET CHARACTER SET 'CP1257';
SET CHARACTER SET 'CP932';
```



```
SET CHARACTER SET 'EUCJPMS';
SET CHARACTER SET 'EUCKR';
SET CHARACTER SET 'GB2312';
SET CHARACTER SET 'GBK';
SET CHARACTER SET 'GREEK';
SET CHARACTER SET 'HEBREW';
SET CHARACTER SET 'KOI8R';
SET CHARACTER SET 'KOI8U';
SET CHARACTER SET 'LATIN1';
SET CHARACTER SET 'LATIN2';
SET CHARACTER SET 'LATIN5';
SET CHARACTER SET 'LATIN7';
SET CHARACTER SET 'SJIS';
SET CHARACTER SET 'SWE7';
SET CHARACTER SET 'TIS620';
SET CHARACTER SET 'UTF8';
SET CHARACTER SET 'UTF8MB4';
## MySQL does not support SET CHARACTER SET 'UCS2';.
## MySQL does not support SET CHARACTER SET 'UTF16';.
## MySQL does not support SET CHARACTER SET 'UTF16LE';.
## MySQL does not support SET CHARACTER SET 'UTF32';.
```

Output

```
SET SESSION NAMES 'SQL_ASCII';
SET SESSION NAMES 'BIG5';
SET SESSION NAMES 'WIN1250';
SET SESSION NAMES 'WIN1251';
SET SESSION NAMES 'WIN1256';
SET SESSION NAMES 'WIN1257';
SET SESSION NAMES 'SJIS';
SET SESSION NAMES 'EUC_JP';
SET SESSION NAMES 'EUC_KR';
SET SESSION NAMES 'GB18030';
SET SESSION NAMES 'GBK';
SET SESSION NAMES 'ISO_8859_7';
SET SESSION NAMES 'ISO_8859_8';
SET SESSION NAMES 'KOI8R';
SET SESSION NAMES 'KOI8U';
SET SESSION NAMES 'LATIN1';
SET SESSION NAMES 'LATIN2';
SET SESSION NAMES 'LATIN5';
SET SESSION NAMES 'LATIN7';
SET SESSION NAMES 'SJIS';
SET SESSION NAMES 'UTF8';
SET SESSION NAMES 'WIN874';
SET SESSION NAMES 'UTF8';
SET SESSION NAMES 'UTF8';
-- MySQL does not support SET CHARACTER SET 'UCS2';.
-- MySQL does not support SET CHARACTER SET 'UTF16';.
-- MySQL does not support SET CHARACTER SET 'UTF16LE';.
-- MySQL does not support SET CHARACTER SET 'UTF32';.
```

6.6 Oracle Syntax Migration

6.6.1 Overview

This section lists the Oracle features supported by the syntax migration tool, and provides the Oracle syntax and the equivalent GaussDB(DWS) syntax for each feature. The syntax listed in this section illustrates the internal logic for Oracle script migration.

This section is also a reference for the database migration team and for the on-site verification of Oracle script migration.

6.6.2 Schema Objects

This section contains the migration syntax for migrating Oracle schema objects. The migration syntax decides how the supported keywords/features are migrated.

This section describes the following:

Tables, temporary tables, global temporaray tables, indexes, views, sequences, PURGE, and database keywords. For details, see **Tables** to **Database Keywords**.

6.6.2.1 Tables

CREATE TABLE

The **CREATE TABLE** statement in Oracle databases is used to create new tables. GaussDB(DWS) also supports this statement. So it does not need to be migrated.

ALTER TABLE

The **ALTER TABLE** statement in Oracle databases is used to add, rename, modify, or drop/delete columns in a table. GaussDB(DWS) also supports this statement. It does not need to be migrated.

PRIMARY KEY

The **ALTER TABLE** statement in Oracle databases is used to add table names when the primary key appears in a different file other than the CREATE table statement.

Input - PRIMARY KEY

```
CREATE TABLE CTP_ARM_CONFIG
   ( HOSTNAME VARCHAR2(50),
 OPNAME VARCHAR2(50),
 PARAMTYPE VARCHAR2(2),
 PARAMVALUE NUMBER(*,0),
 MODIFYDATE DATE
  ) SEGMENT CREATION DEFERRED
 PCTFREE 10 PCTUSED 0 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
 STORAGE( PCTINCREASE 0
 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
 TABLESPACE SPMS_DATA ;

 ALTER TABLE CTP_ARM_CONFIG ADD CONSTRAINT PKCTP_ARM_CONFIG PRIMARY KEY (HOSTNAME,
OPNAME)
  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE( PCTINCREASE 0
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE SPMS_DATA  ENABLE;
```

Output

```
CREATE
    TABLE
        CTP_ARM_CONFIG (
            HOSTNAME VARCHAR2 (50)
            ,OPNAME VARCHAR2 (50)
            ,PARAMTYPE VARCHAR2 (2)
            ,PARAMVALUE NUMBER (
                38
                ,0
            )
```



```
            ,MODIFYDATE DATE
            ,CONSTRAINT PKCTP_ARM_CONFIG PRIMARY KEY (
                HOSTNAME
                ,OPNAME
            )
        ) /*SEGMENT CREATION DEFERRED*/
        /*PCTFREE 10*/
        /*PCTUSED 0*/
        /*INITRANS 1*/
        /*MAXTRANS 255*/
        /*NOCOMPRESS*/
        /*LOGGING*/
        /*STORAGE(  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)*/
        /*TABLESPACE SPMS_DATA */
;
```

UNIQUE constraint

The **ALTER TABLE** query contains the **UNIQUE** constraint. If it is directly executed in GaussDB(DWS), the following error shows: "Cannot create index whose evaluation cannot be enforced to remote nodes".

Similar to **PRIMARY KEY**, if the **PRIMARY KEY**/**UNIQUE** constraint already exists, you do not need to migrate it.

Input

```
CREATE
    TABLE
        GCC_PLAN.T1033 (
            ROLLOUT_PLAN_LINE_ID NUMBER NOT NULL
            ,UDF_FIELD_VALUE_ID NUMBER NOT NULL
        ) ;
ALTER TABLE
    GCC_PLAN.T1033 ADD CONSTRAINT UDF_FIELD_VALUE_ID_PK UNIQUE (UDF_FIELD_VALUE_ID) ;
```

Output

```
CREATE TABLE
       GCC_PLAN.T1033
                        (
                         ROLLOUT_PLAN_LINE_ID NUMBER NOT NULL
                         ,UDF_FIELD_VALUE_ID NUMBER NOT NULL
                         ,CONSTRAINT UDF_FIELD_VALUE_ID_PK UNIQUE (UDF_FIELD_VALUE_ID)
                        ) ;
```

NULL Constraint

NULL constraint during local variable declaration is not supported in packages - that is L_CONTRACT_DISTRIBUTE_STATUS SAD_DISTRIBUTION_HEADERS_T.STATUS%TYPE NULL ;

Input

```
CREATE OR REPLACE FUNCTION CONTRACT_DISTRIBUTE_STATUS_S2(PI_CONTRACT_NUMBER IN
VARCHAR2)
  RETURN VARCHAR2 IS
  L_CONTRACT_DISTRIBUTE_STATUS BAS_SUBTYPE_PKG.STATUS NULL;

BEGIN

  FOR CUR_CONTRACT IN (SELECT HT.CONTRACT_STATUS
                       FROM SAD_CONTRACTS_V HT
                      WHERE HT.HTH = PI_CONTRACT_NUMBER)
  LOOP
    IF CUR_CONTRACT.CONTRACT_STATUS = 0 THEN
      L_CONTRACT_DISTRIBUTE_STATUS := 'Cancel';
```



```
   ELSE
     L_CONTRACT_DISTRIBUTE_STATUS := BAS_SUBTYPE_PKG.G_HEADER_WAITING_SPLIT_STATUS;
   END IF;
 END LOOP;

 RETURN L_CONTRACT_DISTRIBUTE_STATUS;

END CONTRACT_DISTRIBUTE_STATUS_S2;
/
```

Output

```
CREATE OR REPLACE FUNCTION CONTRACT_DISTRIBUTE_STATUS_S2
  ( PI_CONTRACT_NUMBER IN VARCHAR2 )
RETURN VARCHAR2
PACKAGE
IS
 L_CONTRACT_DISTRIBUTE_STATUS BAS_SUBTYPE_PKG.STATUS /*NULL*/;
BEGIN
     FOR CUR_CONTRACT IN ( SELECT HT.CONTRACT_STATUS
        FROM SAD_CONTRACTS_V HT
       WHERE HT.HTH = PI_CONTRACT_NUMBER )
  LOOP
           IF CUR_CONTRACT.CONTRACT_STATUS = 0 THEN
              L_CONTRACT_DISTRIBUTE_STATUS := 'Cancel' ;

   ELSE
    L_CONTRACT_DISTRIBUTE_STATUS := BAS_SUBTYPE_PKG.G_HEADER_WAITING_SPLIT_STATUS ;

   END IF ;

     END LOOP ;

     RETURN L_CONTRACT_DISTRIBUTE_STATUS ;
END ;
/
```

NO INDEX CREATED

If the **INDEX** or **STORAGE** parameter is used in **ALTER TABLE**, delete the parameter. Add constraints to **CREATE TABLE**.

Input - PRIMARY KEY

```
CREATE TABLE CTP_ARM_CONFIG
( HOSTNAME VARCHAR2(50),
OPNAME VARCHAR2(50),
PARAMTYPE VARCHAR2(2),
PARAMVALUE NUMBER(*,0),
MODIFYDATE DATE
) SEGMENT CREATION DEFERRED
PCTFREE 10 PCTUSED 0 INITRANS 1 MAXTRANS 255
NOCOMPRESS LOGGING
STORAGE( PCTINCREASE 0
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE SPMS_DATA ;
ALTER TABLE CTP_ARM_CONFIG ADD CONSTRAINT PKCTP_ARM_CONFIG PRIMARY KEY
(HOSTNAME, OPNAME)
USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
STORAGE( PCTINCREASE 0
BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
TABLESPACE SPMS_DATA ENABLE;
```

Output

```
CREATE TABLE
CTP_ARM_CONFIG (
HOSTNAME VARCHAR2 (50)
```

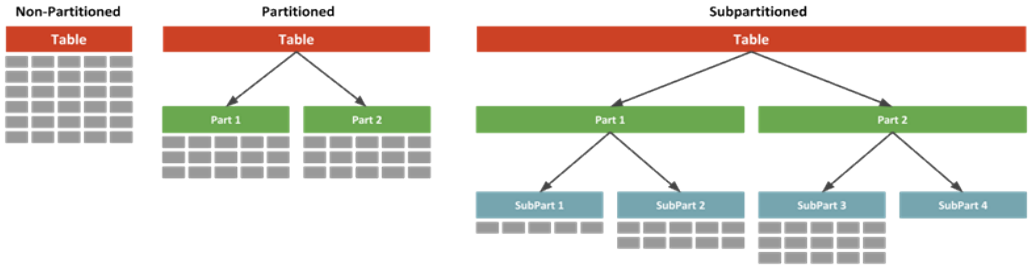


```
,OPNAME VARCHAR2 (50)
,PARAMTYPE VARCHAR2 (2)
,PARAMVALUE NUMBER (
38
,0
)
,MODIFYDATE DATE
,CONSTRAINT PKCTP_ARM_CONFIG PRIMARY KEY (
HOSTNAME
,OPNAME
)
) /*SEGMENT CREATION DEFERRED*/
/*PCTFREE 10*/
/*PCTUSED 0*/
/*INITRANS 1*/
/*MAXTRANS 255*/
/*NOCOMPRESS*/
/*LOGGING*/
/*STORAGE( BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
DEFAULT)*/
/*TABLESPACE SPMS_DATA */
;
```

PARTITIONS

Maintenance of large tables and indexes can become very time and resource consuming. At the same time, data access performance can reduce drastically for these objects. Partitioning of tables and indexes can benefit the performance and maintenance in several ways.

Figure 6-5 Partitioning and sub-partitioning of tables



DSC supports migration of range partition.

The tool does not support the following partitions/subpartitions and these are commented in the migrated scripts:

- List partition
- Hash partition
- Range subpartition
- List subpartition
- Hash subpartition

The unsupported partitions/subpartitions may be supported in the future. Configuration parameters have been provided to enable/disable commenting of the unsupported statements. For details, see **Configuration Parameters for Oracle Features**.

- **PARTITION BY HASH**

  Hash partitioning is a partitioning technique where a hash algorithm is used to distribute rows evenly across the different partitions (sub-tables). This is typically used where ranges are not appropriate, for example employee ID, product ID, and so on. DSC does not support PARTITION and SUBPARTITION by HASH and will comment these statements.

  Input - HASH PARTITION

  ```
  CREATE TABLE dept (deptno NUMBER, deptname VARCHAR(32)) PARTITION BY HASH(deptno)
  PARTITIONS 16;
  ```

  Output

  ```
  CREATE TABLE dept ( deptno NUMBER ,deptname VARCHAR( 32 ) ) /* PARTITION BY HASH(deptno)
  PARTITIONS 16 */ ;
  ```

  Input - HASH PARTITION without partition names

  ```
  CREATE TABLE dept (deptno NUMBER, deptname VARCHAR(32))
      PARTITION BY HASH(deptno) PARTITIONS 16;
  ```

  Output

  ```
  CREATE TABLE dept (deptno NUMBER, deptname VARCHAR(32))
   /*    PARTITION BY HASH(deptno) PARTITIONS 16 */;
  ```

  Input - HASH SUBPARTITION

  ```
  CREATE TABLE sales
    ( prod_id       NUMBER(6)
    , cust_id       NUMBER
    , time_id       DATE
    , channel_id    CHAR(1)
    , promo_id      NUMBER(6)
    , quantity_sold NUMBER(3)
    , amount_sold   NUMBER(10,2)
    )
   PARTITION BY RANGE (time_id) SUBPARTITION BY HASH (cust_id)
    SUBPARTITIONS 8STORE IN (ts1, ts2, ts3, ts4)
   ( PARTITION sales_q1_2006 VALUES LESS THAN (TO_DATE('01-APR-2006','dd-MON-yyyy'))
   , PARTITION sales_q2_2006 VALUES LESS THAN (TO_DATE('01-JUL-2006','dd-MON-yyyy'))
   , PARTITION sales_q3_2006 VALUES LESS THAN (TO_DATE('01-OCT-2006','dd-MON-yyyy'))
   , PARTITION sales_q4_2006 VALUES LESS THAN (TO_DATE('01-JAN-2007','dd-MON-yyyy'))
   );
  ```

  Output

  ```
  CREATE TABLE sales
    ( prod_id       NUMBER(6)
    , cust_id       NUMBER
    , time_id       DATE
    , channel_id    CHAR(1)
    , promo_id      NUMBER(6)
    , quantity_sold NUMBER(3)
    , amount_sold   NUMBER(10,2)
    )
   PARTITION BY RANGE (time_id) /*SUBPARTITION BY HASH (cust_id)
    SUBPARTITIONS 8  STORE IN (ts1, ts2, ts3, ts4) */
   ( PARTITION sales_q1_2006 VALUES LESS THAN (TO_DATE('01-APR-2006','dd-MON-yyyy'))
   , PARTITION sales_q2_2006 VALUES LESS THAN (TO_DATE('01-JUL-2006','dd-MON-yyyy'))
   , PARTITION sales_q3_2006 VALUES LESS THAN (TO_DATE('01-OCT-2006','dd-MON-yyyy'))
   , PARTITION sales_q4_2006 VALUES LESS THAN (TO_DATE('01-JAN-2007','dd-MON-yyyy'))
   );
  ```

- **PARTITION BY LIST**

  List partitioning is a partitioning technique where you specify a list of discrete values for the partitioning key in the description for each partition. DSC does not support PARTITION and SUBPARTITION by LIST and will comment these statements.

  Input - LIST PARTITION


```
CREATE TABLE sales_by_region (item# INTEGER, qty INTEGER, store_name VARCHAR(30), state_code
VARCHAR(2), sale_date DATE) STORAGE(INITIAL 10K NEXT 20K) TABLESPACE tbs5 PARTITION BY
LIST (state_code) ( PARTITION region_east VALUES ('MA','NY','CT','NH','ME','MD','VA','PA','NJ')
STORAGE (INITIAL 8M) TABLESPACE tbs8, PARTITION region_west VALUES
('CA','AZ','NM','OR','WA','UT','NV','CO') NOLOGGING, PARTITION region_south VALUES
('TX','KY','TN','LA','MS','AR','AL','GA'), PARTITION region_central VALUES
('OH','ND','SD','MO','IL','MI','IA'), PARTITION region_null VALUES (NULL), PARTITION region_unknown
VALUES (DEFAULT) );
```

Output

```
CREATE UNLOGGED TABLE sales_by_region ( item# INTEGER ,qty INTEGER ,store_name
VARCHAR( 30 ) ,state_code VARCHAR( 2 ) ,sale_date DATE ) TABLESPACE tbs5 /* PARTITION BY
LIST(state_code)(PARTITION region_east VALUES('MA','NY','CT','NH','ME','MD','VA','PA','NJ')
TABLESPACE tbs8, PARTITION region_west VALUES('CA','AZ','NM','OR','WA','UT','NV','CO') , PARTITION
region_south VALUES('TX','KY','TN','LA','MS','AR','AL','GA'), PARTITION region_central
VALUES('OH','ND','SD','MO','IL','MI','IA'), PARTITION region_null VALUES(NULL), PARTITION
region_unknown VALUES(DEFAULT) ) */ ;
```

Input - LIST PARTITION (With Storage Parameters)

```
CREATE TABLE store_master
        ( Store_id NUMBER
        , Store_address VARCHAR2 (40)
        , City VARCHAR2 (30)
        , State VARCHAR2 (2)
        , zip VARCHAR2 (10)
        , manager_id NUMBER
        )
   /*TABLESPACE users*/
   STORAGE ( INITIAL 100 k NEXT 100 k
    PCTINCREASE 0 )
  PARTITION BY LIST (city)
      ( PARTITION south_florida
        VALUES ( 'MIA', 'ORL' )
        /*TABLESPACE users*/
        STORAGE ( INITIAL 100 k NEXT 100
        k PCTINCREASE 0 )
      , PARTITION north_florida
        VALUES ( 'JAC', 'TAM', 'PEN' )
        /*TABLESPACE users*/
        STORAGE ( INITIAL 100 k NEXT 100
        k PCTINCREASE 0 )
      , PARTITION south_georga VALUES
        ( 'BRU', 'WAY', 'VAL' )
        /*TABLESPACE users*/
        STORAGE ( INITIAL 100 k NEXT 100
        k PCTINCREASE 0 )
      , PARTITION north_georgia
        VALUES ( 'ATL', 'SAV', NULL )
      );
```

Output

```
CREATE TABLE store_master
        ( Store_id NUMBER
        , Store_address VARCHAR2 (40)
        , City VARCHAR2 (30)
        , State VARCHAR2 (2)
        , zip VARCHAR2 (10)
        , manager_id NUMBER
        )
   /*TABLESPACE users*/
   STORAGE ( INITIAL 100 k NEXT 100 k );
```

Input: LIST PARTITION TABLE from another TABLE

```
CREATE TABLE tab1_list
     PARTITION BY LIST (col1)
       ( partition part1 VALUES ( 1 )
        , partition part2 VALUES ( 2,
         3, 4 )
        , partition part3 VALUES
```



```
        (DEFAULT)
        )
AS
SELECT *
  FROM  tab1;
```

Output

```
CREATE TABLE tab1_list
 AS
 ( SELECT *
   FROM  tab1 );
```

Input - LIST PARTITION with SUBPARTITIONS

```
CREATE TABLE big_t_list PARTITION BY LIST(n10) (partition part1 VALUES (1) ,partition part2 VALUES (2,3,4) ,partition part3 VALUES (DEFAULT)) AS SELECT * FROM big_t;
```

Output

```
CREATE TABLE big_t_list /* PARTITION BY LIST(n10)(partition part1 VALUES(1) ,partition part2 VALUES(2,3,4) ,partition part3 VALUES(DEFAULT))  */ AS ( SELECT * FROM big_t ) ;
```

Input - LIST PARTITION with SUBPARTITION TEMPLATE

```
CREATE TABLE q1_sales_by_region
        ( deptno NUMBER
        , deptname varchar2 (20)
        , quarterly_sales NUMBER
        (10,2)
        , state varchar2 (2)
        )
  PARTITION BY LIST (state)
       SUBPARTITION BY RANGE
       (quarterly_sales)
       SUBPARTITION TEMPLATE
       ( SUBPARTITION original VALUES
       LESS THAN (1001)
       , SUBPARTITION acquired VALUES
       LESS THAN (8001)
       , SUBPARTITION recent VALUES
       LESS THAN (MAXVALUE)
       )
   ( PARTITION q1_northwest VALUES
    ( 'OR', 'WA' )
   , PARTITION q1_southwest VALUES
    ( 'AZ', 'UT', 'NM' )
   , PARTITION q1_northeast VALUES
    ( 'NY', 'VM', 'NJ' )
   , PARTITION q1_southcentral VALUES
    ( 'OK', 'TX' )
   );
```

Output

```
CREATE TABLE q1_sales_by_region
        ( deptno NUMBER
        , deptname varchar2 (20)
        , quarterly_sales NUMBER (10,2)
        , state varchar2 (2)
        );
```

- **PARTITION BY RANGE**

  Range partitioning is a partitioning technique where data of different ranges is stored separately in different sub-tables. Range partitioning is useful when you have distinct ranges of data you want to store together, for example the date field. DSC supports PARTITION by RANGE. It does not support SUBPARTITION by RANGE and will comment these statements.

  Input - RANGE PARTITION (With STORAGE Parameters)

```
CREATE
    TABLE
```



```
        CCM_TA550002_H (
            STRU_ID VARCHAR2 (10)
            ,ORGAN1_NO VARCHAR2 (10)
            ,ORGAN2_NO VARCHAR2 (10)
        ) partition BY range (ORGAN2_NO) (
            partition CCM_TA550002_01
            VALUES LESS than ('00100') /* TABLESPACE users */
            /*pctfree 10*/
            /*initrans 1*/
            /*storage(initial 256 K NEXT 256 K minextents 1 maxextents unlimited  )*/
            ,partition CCM_TA550002_02
            VALUES LESS than ('00200') /* TABLESPACE users */
            /*pctfree 10*/
            /*initrans 1*/
            /* storage ( initial 256 K NEXT
256K  minextents 1
maxextents unlimited
pctincrease 0 )*/
```

Output

```
CREATE TABLE CCM_TA550002_H
        ( STRU_ID VARCHAR2 (10)
        , ORGAN1_NO VARCHAR2 (10)
        , ORGAN2_NO VARCHAR2 (10)
        )
   partition BY range (ORGAN2_NO)
          ( partition CCM_TA550002_01 VALUES LESS
            than ('00100')
            /*TABLESPACE users*/
          , partition CCM_TA550002_02 VALUES LESS
            than ('00200')
            /*TABLESPACE users*/
          );
```

Input - RANGE PARTITION with SUBPARTITIONS

```
CREATE TABLE composite_rng_list (
cust_id     NUMBER(10),
cust_name   VARCHAR2(25),
cust_state  VARCHAR2(2),
time_id     DATE)
PARTITION BY RANGE(time_id)
SUBPARTITION BY LIST (cust_state)
SUBPARTITION TEMPLATE(
SUBPARTITION west VALUES ('OR', 'WA') TABLESPACE part1,
SUBPARTITION east VALUES ('NY', 'CT') TABLESPACE part2,
SUBPARTITION cent VALUES ('OK', 'TX') TABLESPACE part3) (
PARTITION per1 VALUES LESS THAN (TO_DATE('01/01/2000','DD/MM/YYYY')),
PARTITION per2 VALUES LESS THAN (TO_DATE('01/01/2005','DD/MM/YYYY')),
PARTITION per3 VALUES LESS THAN (TO_DATE('01/01/2010','DD/MM/YYYY')),
PARTITION future VALUES LESS THAN(MAXVALUE));
```

Output

```
CREATE TABLE composite_rng_list (
cust_id     NUMBER(10),
cust_name   VARCHAR2(25),
cust_state  VARCHAR2(2),
time_id     DATE)
PARTITION BY RANGE(time_id)
/*SUBPARTITION BY LIST (cust_state)
SUBPARTITION TEMPLATE(
SUBPARTITION west VALUES ('OR', 'WA') TABLESPACE part1,
SUBPARTITION east VALUES ('NY', 'CT') TABLESPACE part2,
SUBPARTITION cent VALUES ('OK', 'TX') TABLESPACE part3)*/ (
PARTITION per1 VALUES LESS THAN (TO_DATE('01/01/2000','DD/MM/YYYY')),
PARTITION per2 VALUES LESS THAN (TO_DATE('01/01/2005','DD/MM/YYYY')),
PARTITION per3 VALUES LESS THAN (TO_DATE('01/01/2010','DD/MM/YYYY')),
PARTITION future VALUES LESS THAN(MAXVALUE));
```

Input - RANGE PARTITION with SUBPARTITION TEMPLATE


```
CREATE TABLE composite_rng_rng (
cust_id     NUMBER(10),
cust_name   VARCHAR2(25),
cust_state  VARCHAR2(2),
time_id     DATE)
PARTITION BY RANGE(time_id)
SUBPARTITION BY RANGE (cust_id)
SUBPARTITION TEMPLATE(
SUBPARTITION original VALUES LESS THAN (1001) TABLESPACE part1,
SUBPARTITION acquired VALUES LESS THAN (8001) TABLESPACE part2,
SUBPARTITION recent VALUES LESS THAN (MAXVALUE) TABLESPACE part3) (
PARTITION per1 VALUES LESS THAN (TO_DATE('01/01/2000','DD/MM/YYYY')),
PARTITION per2 VALUES LESS THAN (TO_DATE('01/01/2005','DD/MM/YYYY')),
PARTITION per3 VALUES LESS THAN (TO_DATE('01/01/2010','DD/MM/YYYY')),
PARTITION future VALUES LESS THAN (MAXVALUE));
```

Output

```
CREATE TABLE composite_rng_rng (
cust_id     NUMBER(10),
cust_name   VARCHAR2(25),
cust_state  VARCHAR2(2),
time_id     DATE)
PARTITION BY RANGE(time_id)
/*SUBPARTITION BY RANGE (cust_id)
SUBPARTITION TEMPLATE(
SUBPARTITION original VALUES LESS THAN (1001) TABLESPACE part1,
SUBPARTITION acquired VALUES LESS THAN (8001) TABLESPACE part2,
SUBPARTITION recent VALUES LESS THAN (MAXVALUE) TABLESPACE part3)*/ (
PARTITION per1 VALUES LESS THAN (TO_DATE('01/01/2000','DD/MM/YYYY')),
PARTITION per2 VALUES LESS THAN (TO_DATE('01/01/2005','DD/MM/YYYY')),
PARTITION per3 VALUES LESS THAN (TO_DATE('01/01/2010','DD/MM/YYYY')),
PARTITION future VALUES LESS THAN (MAXVALUE));
```

PRIMARY KEY/UNIQUE Constraint for Partitioned Table

If the CREATE TABLE statement contains range/hash/list partitioning, the following error is reported:

Invalid PRIMARY KEY/UNIQUE constraint for partitioned table

Note: Columns of the PRIMARY KEY/UNIQUE constraint must contain PARTITION KEY.

Scripts: wo_integrate_log_t.sql, wo_change_log_t.sql

Input:

```
CREATE TABLE SD_WO.WO_INTEGRATE_LOG_T
(
LOG_ID            NUMBER not null,
PROJECT_NUMBER    VARCHAR2(40),
MESSAGE_ID        VARCHAR2(100),
BUSINESS_ID       VARCHAR2(100),
BUSINESS_TYPE     VARCHAR2(100),
INTEGRATE_CONTENT CLOB,
OPERATION_RESULT  VARCHAR2(100),
FAILED_MSG        VARCHAR2(4000),
HOST_NAME         VARCHAR2(100) not null,
CREATED_BY        NUMBER not null,
CREATION_DATE     DATE not null,
LAST_UPDATED_BY   NUMBER not null,
LAST_UPDATE_DATE  DATE not null,
SOURCE_CODE       VARCHAR2(100),
TENANT_ID         NUMBER
)
partition by range (CREATION_DATE)
(
partition P2018 values less than (TO_DATE(' 2018-10-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS',
'NLS_CALENDAR=GREGORIAN'))
tablespace SDWO_DATA,
partition SYS_P53873 values less than (TO_DATE(' 2018-11-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS',
```



```
'NLS_CALENDAR=GREGORIAN'))
tablespace SDWO_DATA,
partition SYS_P104273 values less than (TO_DATE(' 2018-12-01 00:00:00', 'SYYYY-MM-DD
HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN'))
tablespace SDWO_DATA,
partition SYS_P105533 values less than (TO_DATE(' 2019-01-01 00:00:00', 'SYYYY-MM-DD
HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN'))
tablespace SDWO_DATA,
partition SYS_P108153 values less than (TO_DATE(' 2019-02-01 00:00:00', 'SYYYY-MM-DD
HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN'))
tablespace SDWO_DATA,
partition SYS_P127173 values less than (TO_DATE(' 2019-03-01 00:00:00', 'SYYYY-MM-DD
HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN'))
tablespace SDWO_DATA,
partition SYS_P130313 values less than (TO_DATE(' 2019-04-01 00:00:00', 'SYYYY-MM-DD
HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN'))
tablespace SDWO_DATA
);
alter table SD_WO.WO_INTEGRATE_LOG_T
add constraint WO_INTEGRATE_LOG_PK primary key (LOG_ID);
create index SD_WO.WO_INTEGRATE_LOG_N1 on SD_WO.WO_INTEGRATE_LOG_T (BUSINESS_ID);
create index SD_WO.WO_INTEGRATE_LOG_N2 on SD_WO.WO_INTEGRATE_LOG_T (CREATION_DATE,
BUSINESS_TYPE);
create index SD_WO.WO_INTEGRATE_LOG_N3 on SD_WO.WO_INTEGRATE_LOG_T
(PROJECT_NUMBER, BUSINESS_TYPE);
```

Output:

```
CREATE TABLE
SD_WO.WO_INTEGRATE_LOG_T (
LOG_ID NUMBER NOT NULL
,PROJECT_NUMBER VARCHAR2 (40)
,MESSAGE_ID VARCHAR2 (100)
,BUSINESS_ID VARCHAR2 (100)
,BUSINESS_TYPE VARCHAR2 (100)
,INTEGRATE_CONTENT CLOB
,OPERATION_RESULT VARCHAR2 (100)
,FAILED_MSG VARCHAR2 (4000)
,HOST_NAME VARCHAR2 (100) NOT NULL
,CREATED_BY NUMBER NOT NULL
,CREATION_DATE DATE NOT NULL
,LAST_UPDATED_BY NUMBER NOT NULL
,LAST_UPDATE_DATE DATE NOT NULL
,SOURCE_CODE VARCHAR2 (100)
,TENANT_ID NUMBER
,CONSTRAINT WO_INTEGRATE_LOG_PK PRIMARY KEY (LOG_ID)
) partition BY range (CREATION_DATE) (
partition P2018
VALUES LESS than (
TO_DATE( ' 2018-10-01 00:00:00' ,'SYYYY-MM-DD HH24:MI:SS'/*, 'NLS_CALENDAR=GREGORIAN'*/ )
) /* tablespace SDWO_DATA */
,partition SYS_P53873
VALUES LESS than (
TO_DATE( ' 2018-11-01 00:00:00' ,'SYYYY-MM-DD HH24:MI:SS'/*, 'NLS_CALENDAR=GREGORIAN'*/ )
) /* tablespace SDWO_DATA */
,partition SYS_P104273
VALUES LESS than (
TO_DATE( ' 2018-12-01 00:00:00' ,'SYYYY-MM-DD HH24:MI:SS'/*, 'NLS_CALENDAR=GREGORIAN'*/ )
) /* tablespace SDWO_DATA */
,partition SYS_P105533
VALUES LESS than (
TO_DATE( ' 2019-01-01 00:00:00' ,'SYYYY-MM-DD HH24:MI:SS'/*, 'NLS_CALENDAR=GREGORIAN'*/ )
) /* tablespace SDWO_DATA */
,partition SYS_P108153
VALUES LESS than (
TO_DATE( ' 2019-02-01 00:00:00' ,'SYYYY-MM-DD HH24:MI:SS'/*, 'NLS_CALENDAR=GREGORIAN'*/ )
) /* tablespace SDWO_DATA */
,partition SYS_P127173
VALUES LESS than (
TO_DATE( ' 2019-03-01 00:00:00' ,'SYYYY-MM-DD HH24:MI:SS'/*, 'NLS_CALENDAR=GREGORIAN'*/ )
) /* tablespace SDWO_DATA */
```



```
,partition SYS_P130313
VALUES LESS than (
TO_DATE( ' 2019-04-01 00:00:00' ,'SYYYY-MM-DD HH24:MI:SS'/*, 'NLS_CALENDAR=GREGORIAN'*/ )
) /* tablespace SDWO_DATA */
) ;
CREATE
index WO_INTEGRATE_LOG_N1
ON SD_WO.WO_INTEGRATE_LOG_T (BUSINESS_ID) LOCAL ;
CREATE
index WO_INTEGRATE_LOG_N2
ON SD_WO.WO_INTEGRATE_LOG_T (
CREATION_DATE
,BUSINESS_TYPE
) LOCAL ;
CREATE
index WO_INTEGRATE_LOG_N3
ON SD_WO.WO_INTEGRATE_LOG_T (
PROJECT_NUMBER
,BUSINESS_TYPE
) LOCAL ;
```

Input:

```
CREATE TABLE SD_WO.WO_INTEGRATE_LOG_T
(
  LOG_ID             NUMBER not null,
  PROJECT_NUMBER     VARCHAR2(40),
  MESSAGE_ID         VARCHAR2(100),
  BUSINESS_ID        VARCHAR2(100),
  BUSINESS_TYPE      VARCHAR2(100),
  INTEGRATE_CONTENT CLOB,
  OPERATION_RESULT  VARCHAR2(100),
  FAILED_MSG         VARCHAR2(4000),
  HOST_NAME          VARCHAR2(100) not null,
  CREATED_BY         NUMBER not null,
  CREATION_DATE      DATE not null,
  LAST_UPDATED_BY    NUMBER not null,
  LAST_UPDATE_DATE  DATE not null,
  SOURCE_CODE        VARCHAR2(100),
  TENANT_ID          NUMBER
)
partition by range (CREATION_DATE)
(
  partition P2018 values less than (TO_DATE(' 2018-10-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS',
'NLS_CALENDAR=GREGORIAN'))
    tablespace SDWO_DATA,
  partition SYS_P53873 values less than (TO_DATE(' 2018-11-01 00:00:00', 'SYYYY-MM-DD
HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN'))
    tablespace SDWO_DATA,
  partition SYS_P104273 values less than (TO_DATE(' 2018-12-01 00:00:00', 'SYYYY-MM-DD
HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN'))
    tablespace SDWO_DATA,
  partition SYS_P105533 values less than (TO_DATE(' 2019-01-01 00:00:00', 'SYYYY-MM-DD
HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN'))
    tablespace SDWO_DATA,
  partition SYS_P108153 values less than (TO_DATE(' 2019-02-01 00:00:00', 'SYYYY-MM-DD
HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN'))
    tablespace SDWO_DATA,
  partition SYS_P127173 values less than (TO_DATE(' 2019-03-01 00:00:00', 'SYYYY-MM-DD
HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN'))
    tablespace SDWO_DATA,
  partition SYS_P130313 values less than (TO_DATE(' 2019-04-01 00:00:00', 'SYYYY-MM-DD
HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN'))
    tablespace SDWO_DATA
);

alter table SD_WO.WO_INTEGRATE_LOG_T
  add constraint WO_INTEGRATE_LOG_PK primary key (LOG_ID);
create index SD_WO.WO_INTEGRATE_LOG_N1 on SD_WO.WO_INTEGRATE_LOG_T (BUSINESS_ID);
create index SD_WO.WO_INTEGRATE_LOG_N2 on SD_WO.WO_INTEGRATE_LOG_T (CREATION_DATE,
BUSINESS_TYPE);
```



```
create index SD_WO.WO_INTEGRATE_LOG_N3 on SD_WO.WO_INTEGRATE_LOG_T
(PROJECT_NUMBER, BUSINESS_TYPE);
```

Output:

```
CREATE TABLE SD_WO.WO_INTEGRATE_LOG_T
(
  LOG_ID             NUMBER not null,
  PROJECT_NUMBER     VARCHAR2(40),
  MESSAGE_ID         VARCHAR2(100),
  BUSINESS_ID        VARCHAR2(100),
  BUSINESS_TYPE      VARCHAR2(100),
  INTEGRATE_CONTENT CLOB,
  OPERATION_RESULT  VARCHAR2(100),
  FAILED_MSG         VARCHAR2(4000),
  HOST_NAME          VARCHAR2(100) not null,
  CREATED_BY         NUMBER not null,
  CREATION_DATE      DATE not null,
  LAST_UPDATED_BY    NUMBER not null,
  LAST_UPDATE_DATE  DATE not null,
  SOURCE_CODE        VARCHAR2(100),
  TENANT_ID          NUMBER
)
partition by range (CREATION_DATE)
(
  partition P2018 values less than (TO_DATE(' 2018-10-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS',
'NLS_CALENDAR=GREGORIAN'))
    tablespace SDWO_DATA,
  partition SYS_P53873 values less than (TO_DATE(' 2018-11-01 00:00:00', 'SYYYY-MM-DD
HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN'))
    tablespace SDWO_DATA,
  partition SYS_P104273 values less than (TO_DATE(' 2018-12-01 00:00:00', 'SYYYY-MM-DD
HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN'))
    tablespace SDWO_DATA,
  partition SYS_P105533 values less than (TO_DATE(' 2019-01-01 00:00:00', 'SYYYY-MM-DD
HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN'))
    tablespace SDWO_DATA,
  partition SYS_P108153 values less than (TO_DATE(' 2019-02-01 00:00:00', 'SYYYY-MM-DD
HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN'))
    tablespace SDWO_DATA,
  partition SYS_P127173 values less than (TO_DATE(' 2019-03-01 00:00:00', 'SYYYY-MM-DD
HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN'))
    tablespace SDWO_DATA,
  partition SYS_P130313 values less than (TO_DATE(' 2019-04-01 00:00:00', 'SYYYY-MM-DD
HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN'))
    tablespace SDWO_DATA
);

ALTER TABLE SD_WO.WO_INTEGRATE_LOG_T
  add constraint WO_INTEGRATE_LOG_PK primary key (LOG_ID);
create index SD_WO.WO_INTEGRATE_LOG_N1 on SD_WO.WO_INTEGRATE_LOG_T (BUSINESS_ID);
create index SD_WO.WO_INTEGRATE_LOG_N2 on SD_WO.WO_INTEGRATE_LOG_T (CREATION_DATE,
BUSINESS_TYPE);
create index SD_WO.WO_INTEGRATE_LOG_N3 on SD_WO.WO_INTEGRATE_LOG_T
(PROJECT_NUMBER, BUSINESS_TYPE);
```

Data Type

Remove the BYTE keyword from the data type.

Oracle Syntax	Syntax After Migration
CREATE TABLE TBL_ORACLE (ID Number, Name VARCHAR2(100 BYTE), ADDRESS VARCHAR2(200 BYTE));	CREATE TABLE TBL_ORACLE (ID NUMBER ,Name VARCHAR2 (100) ,ADDRESS VARCHAR2 (200)) ;

Partition (Comment Partition)

In configuration parameter for oracle "#Unique or primary key constraint for partitioned table" to comment_partition.

Oracle Syntax	Syntax After Migration

CREATE TABLE TBL_ORACLE
(
  ID Number,
  Name VARCHAR2(100 BYTE),
  ADDRESS VARCHAR2(200 BYTE)
)
TABLESPACE space1
PCTUSED 40
PCTFREE 0
INITRANS 1
MAXTRANS 255
NOLOGGING
PARTITION BY RANGE (ID)
(
  PARTITION PART_2010 VALUES LESS THAN (10)
  NOLOGGING,
  PARTITION PART_2011 VALUES LESS THAN (20)
  NOLOGGING ,
  PARTITION PART_2012 VALUES LESS THAN (MAXVALUE)
  NOLOGGING
)
ENABLE ROW MOVEMENT;
 	
ALTER TABLE TBL_ORACLE ADD CONSTRAINT SAMPLE_PK PRIMARY KEY (ID);
	CREATE UNLOGGED TABLE TBL_ORACLE
	(
	  ID NUMBER
	  ,Name VARCHAR2 (100)
	  ,ADDRESS VARCHAR2 (200)
	  ,CONSTRAINT SAMPLE_PK PRIMARY KEY (ID)
	)
	TABLESPACE space1
	  /*PCTUSED 40*/
	  PCTFREE 0
	  INITRANS 1
	  MAXTRANS 255
	/* PARTITION BY RANGE(ID)(PARTITION PART_2010 VALUES LESS THAN(10) ,
	PARTITION PART_2011 VALUES LESS THAN(20) ,
	PARTITION PART_2012 VALUES LESS THAN(MAXVALUE))
	ENABLE ROW MOVEMENT */
	;

Partition (Comment Constraint)

In configuration parameter for oracle "#Unique or primary key constraint for partitioned table" to comment_unique.

Oracle Syntax	Syntax After Migration

CREATE TABLE TBL_ORACLE
(
  ID Number,
  Name VARCHAR2(100 BYTE),
  ADDRESS VARCHAR2(200 BYTE)
)
TABLESPACE space1
PCTUSED 40
PCTFREE 0
INITRANS 1
MAXTRANS 255
NOLOGGING
PARTITION BY RANGE (ID)
(
  PARTITION PART_2010 VALUES LESS THAN (10)
  NOLOGGING,
  PARTITION PART_2011 VALUES LESS THAN (20)
  NOLOGGING ,
  PARTITION PART_2012 VALUES LESS THAN (MAXVALUE)
  NOLOGGING
)
ENABLE ROW MOVEMENT;
 	
ALTER TABLE TBL_ORACLE ADD CONSTRAINT SAMPLE_PK PRIMARY KEY (ID);
	CREATE UNLOGGED TABLE TBL_ORACLE
	(
	  ID NUMBER
	  ,Name VARCHAR2 (100)
	  ,ADDRESS VARCHAR2 (200)
	/*,CONSTRAINT SAMPLE_PK PRIMARY KEY (ID)*/
	)
	TABLESPACE space1
	  /*PCTUSED 40*/
	  PCTFREE 0
	  INITRANS 1
	  MAXTRANS 255
	PARTITION BY RANGE (ID)
	(
	  PARTITION PART_2010 VALUES LESS THAN (10)
	  ,PARTITION PART_2011 VALUES LESS THAN (20)
	  ,PARTITION PART_2012 VALUES LESS THAN (MAXVALUE)
	  ) ENABLE ROW MOVEMENT ;

Partition (I)

Comment ALTER TABLE TRUNCATE PARTITION for non-partitioned tables.

Oracle Syntax	Syntax After Migration

CREATE TABLE product_range
(
  product_id VARCHAR2(20),
  Product_Name VARCHAR2(50),
  Year_Manufacture DATE
)
partition by range (Year_Manufacture)
(
  partition Year_Manufacture values less than
(TO_DATE(' 2007-01-01 00:00:00', 'SYYYY-MM-DD
HH24:MI:SS'))
  pctfree 10
  initrans 1
);
 	
CREATE TABLE product_list
(
  product_id VARCHAR2(20),
  Product_Name VARCHAR2(50),
  Year_Manufacture vARCHAR2(10)
)
partition by list (Year_Manufacture)
(
  partition P_2020 VALUES (2020)
  pctfree 10
  initrans 1
);
 	
CREATE OR REPLACE PROCEDURE Range_test
IS
V_ID VARCHAR2(10);
BEGIN
EXECUTE IMMEDIATE 'ALTER TABLE product
TRUNCATE PARTITION PART'||V_ID;
NULL;
END;
/
CREATE OR REPLACE PROCEDURE List_test
IS
V_ID VARCHAR2(10);
BEGIN
EXECUTE IMMEDIATE 'ALTER TABLE product
TRUNCATE PARTITION PART'||V_ID;
NULL;
END;
/
	CREATE TABLE product_range
	(
	  product_id VARCHAR2(20),
	  Product_Name VARCHAR2(50),
	  Year_Manufacture DATE
	)
	partition by range (Year_Manufacture)
	(
	  partition Year_Manufacture values less than
	(TO_DATE(' 2007-01-01 00:00:00', 'SYYYY-MM-DD
	HH24:MI:SS'))
	  pctfree 10
	  initrans 1
	);
	
	CREATE TABLE product_list
	(
	  product_id VARCHAR2(20),
	  Product_Name VARCHAR2(50),
	  Year_Manufacture vARCHAR2(10)
	)
	/*partition by list (Year_Manufacture)
	(
	  partition P_2020 VALUES (2020)
	  pctfree 10
	  initrans 1
	)*/;
	
	CREATE OR REPLACE PROCEDURE Range_test
	IS
	V_ID VARCHAR2(10);
	BEGIN
	EXECUTE IMMEDIATE 'ALTER TABLE product
	TRUNCATE PARTITION PART'||V_ID;
	NULL;
	END;
	/
	CREATE OR REPLACE PROCEDURE List_test
	IS
	V_ID VARCHAR2(10);
	BEGIN
	/*EXECUTE IMMEDIATE 'ALTER TABLE product
	TRUNCATE PARTITION PART'||V_ID;*/
	NULL;
	END;
	/

Partition (II)

Delete data for ALTER TABLE TRUNCATE PARTITION for non-partitioned tables.

Oracle Syntax	Syntax After Migration

CREATE TABLE product_list
(
  product_id VARCHAR2(20),
  Product_Name VARCHAR2(50),
  Year_Manufacture vARCHAR2(10)
)
partition by list (Year_Manufacture)
(partition PART_2015 VALUES
(2011,2012,2013,2014,2015) pctfree 10 initrans 1 ,
 partition PART_2016 VALUES (2016) pctfree 10
initrans 1 ,
 partition PART_2017 VALUES (2017) pctfree 10
initrans 1 ,
 partition PART_2018 VALUES (2018) pctfree 10
initrans 1 ,
 partition PART_2019 VALUES (2019) pctfree 10
initrans 1 ,
 partition PART_2020 VALUES (2020) pctfree 10
initrans 1 ,
 PARTITION PART_unknown VALUES (DEFAULT));
 	
CREATE OR REPLACE PROCEDURE List_test
IS
V_ID VARCHAR2(10);
BEGIN
 EXECUTE IMMEDIATE 'ALTER TABLE product_list
TRUNCATE PARTITION PART_2020;
 NULL;
END;
/
 	
CREATE OR REPLACE PROCEDURE List_test
IS
V_ID VARCHAR2(10);
BEGIN
 EXECUTE IMMEDIATE 'ALTER TABLE product_list
TRUNCATE PARTITION PART_' || V_ID;
 NULL;
END;
/
	CREATE TABLE product_list
	(
	  product_id VARCHAR2(20),
	  Product_Name VARCHAR2(50),
	  Year_Manufacture vARCHAR2(10)
	)
	/*partition by list (Year_Manufacture)
	(partition PART_2015 VALUES
	(2011,2012,2013,2014,2015) pctfree 10 initrans 1 ,
	 partition PART_2016 VALUES (2016) pctfree 10
	initrans 1 ,
	 partition PART_2017 VALUES (2017) pctfree 10
	initrans 1 ,
	 partition PART_2018 VALUES (2018) pctfree 10
	initrans 1 ,
	 partition PART_2019 VALUES (2019) pctfree 10
	initrans 1 ,
	 partition PART_2020 VALUES (2020) pctfree 10
	initrans 1 ,
	 PARTITION PART_unknown VALUES
	(DEFAULT))*/;
	
	CREATE OR REPLACE PROCEDURE List_test
	IS
	V_ID VARCHAR2(10);
	BEGIN
	 EXECUTE IMMEDIATE 'ALTER TABLE product_list
	TRUNCATE PARTITION PART_' || V_ID;
	 NULL;
	END;
	/
	
	CREATE OR REPLACE PROCEDURE List_test
	IS
	V_ID VARCHAR2(10);
	BEGIN
	 /* EXECUTE IMMEDIATE 'ALTER TABLE
	product_list TRUNCATE PARTITION PART_' || V_ID;
	*/
	 IF 'PART_' || V_ID = 'PART_2015' THEN
	  DELETE FROM product_list WHERE
	Year_Manufacture IN (2011,2012,2013,2014,2015);
	 ELSIF 'PART_' || V_ID = 'PART_2016' THEN
	  DELETE FROM product_list WHERE
	Year_Manufacture IN (2016);
	 ELSIF 'PART_' || V_ID = 'PART_2017' THEN
	  DELETE FROM product_list WHERE
	Year_Manufacture IN (2017);
	 ELSIF 'PART_' || V_ID = 'PART_2018' THEN
	  DELETE FROM product_list WHERE
	Year_Manufacture IN (2018);
	 ELSIF 'PART_' || V_ID = 'PART_2019' THEN
	  DELETE FROM product_list WHERE
	Year_Manufacture IN (2019);
	 ELSIF 'PART_' || V_ID = 'PART_2020' THEN
	  DELETE FROM product_list WHERE
	Year_Manufacture IN (2020);
	 ELSE
	  DELETE FROM product_list WHERE
	Year_Manufacture NOT IN
	(2011,2012,2013,2014,2015,2016,2017,2018,2019,2
	020);
	 END IF;
	 NULL;

Oracle Syntax	Syntax After Migration
	END; /

SEGMENT CREATION

SEGMENT CREATION { IMMEDIATE | DEFERRED } is not supported in GaussDB(DWS), hence it is commented out in the migrated output. You need to set **commentStorageParameter** to **true**.

Input - TABLE with SEGMENT CREATION

```
CREATE TABLE T1
   ( MESSAGE_CODE VARCHAR2(50),
 MAIL_TITLE VARCHAR2(1000),
 MAIL_BODY VARCHAR2(1000),
 MAIL_ADDRESS VARCHAR2(1000),
 MAIL_ADDRESS_CC VARCHAR2(1000)
   ) SEGMENT CREATION DEFERRED
  PCTFREE 10 PCTUSED 0 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
  STORAGE( INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
  TABLESPACE Test ;
```

Output

```
CREATE TABLE T1
   ( MESSAGE_CODE VARCHAR2(50),
 MAIL_TITLE VARCHAR2(1000),
 MAIL_BODY VARCHAR2(1000),
 MAIL_ADDRESS VARCHAR2(1000),
 MAIL_ADDRESS_CC VARCHAR2(1000)
   ) /*SEGMENT CREATION DEFERRED */
  /*PCTFREE 10*/
/* PCTUSED 0 */
/*INITRANS 1 */
/*MAXTRANS 255 */
/* NOCOMPRESS LOGGING*/
/*  STORAGE( INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
  PCTINCREASE 0
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)*/
/*  TABLESPACE Test */;
```

STORAGE

Storage parameters including **BUFFER_POOL** and **MAXEXTENTS** are not supported in GaussDB(DWS). If **comment_storage_parameter** is set to **TRUE**, these parameters that appear in tables or indexes will be commented out during migration.

Input - TABLE with STORAGE

```
CREATE UNIQUE INDEX PK_BASE_APPR_STEP_DEF ON BASE_APPR_STEP_DEF (FLOW_ID, NODE_ID,
STEP_ID)
   PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
   STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
   PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
   BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
   TABLESPACE SPMS_DATA ;
```



```
CREATE TABLE UFP_MAIL
  ( MAIL_ID NUMBER(*,0),
 MAIL_TITLE VARCHAR2(1000),
 MAIL_BODY VARCHAR2(4000),
 STATUS VARCHAR2(50),
 CREATE_TIME DATE,
 SEND_TIME DATE,
 MAIL_ADDRESS CLOB,
 MAIL_CC CLOB,
 BASE_ID VARCHAR2(20),
 BASE_STATUS VARCHAR2(50),
 BASE_VERIFY VARCHAR2(20),
 BASE_LINK VARCHAR2(4000),
 MAIL_TYPE VARCHAR2(20),
 BLIND_COPY_TO CLOB,
 FILE_NAME VARCHAR2(4000),
 FULL_FILEPATH VARCHAR2(4000)
  ) SEGMENT CREATION IMMEDIATE
 PCTFREE 10 PCTUSED 0 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
 STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
 TABLESPACE SPMS_DATA
 LOB (MAIL_ADDRESS) STORE AS BASICFILE (
 TABLESPACE SPMS_DATA ENABLE STORAGE IN ROW CHUNK 8192 RETENTION
 NOCACHE LOGGING
 STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT))
 LOB (MAIL_CC) STORE AS BASICFILE (
 TABLESPACE SPMS_DATA ENABLE STORAGE IN ROW CHUNK 8192 RETENTION
 NOCACHE LOGGING
 STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT))
 LOB (BLIND_COPY_TO) STORE AS BASICFILE (
 TABLESPACE SPMS_DATA ENABLE STORAGE IN ROW CHUNK 8192 RETENTION
 NOCACHE LOGGING
 STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)) ;
```

Output

```
CREATE
    UNIQUE INDEX PK_BASE_APPR_STEP_DEF
        ON BASE_APPR_STEP_DEF (
        FLOW_ID
        ,NODE_ID
        ,STEP_ID
    ) /*PCTFREE 10*/
    /*INITRANS 2*/
    /*MAXTRANS 255*/
    /*COMPUTE STATISTICS*/
    /*STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645  FREELISTS 1
FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)*/
    /*TABLESPACE SPMS_DATA */
;
```

NOTE

If **comment_storage_parameter** is set **TRUE**, then storage parameters are commented.

STORE

The **STORE** keyword for the LOB column is not supported in GaussDB(DWS). Therefore, it is commented out in the migrated output.

Input - TABLE with STORE

```
CREATE TABLE CTP_PROC_LOG
   ( PORC_NAME VARCHAR2(100),
 LOG_TIME VARCHAR2(100),
 LOG_INFO CLOB
  ) SEGMENT CREATION IMMEDIATE
 PCTFREE 10 PCTUSED 0 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
 STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)
 TABLESPACE SPMS_DATA
 LOB (LOG_INFO) STORE AS BASICFILE (
 TABLESPACE SPMS_DATA ENABLE STORAGE IN ROW CHUNK 8192 RETENTION
 NOCACHE LOGGING
 STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645
 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)) ;
```

Output

```
CREATE
    TABLE
        CTP_PROC_LOG (
            PORC_NAME VARCHAR2 (100)
            ,LOG_TIME VARCHAR2 (100)
            ,LOG_INFO CLOB
        ) /*SEGMENT CREATION IMMEDIATE*/
        /*PCTFREE 10*/
        /*PCTUSED 0*/
        /*INITRANS 1*/
        /*MAXTRANS 255*/
        /*NOCOMPRESS*/
        /*LOGGING*/
        /*STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645  FREELISTS 1
FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)*/
        /*TABLESPACE SPMS_DATA */
        /*LOB (LOG_INFO) STORE AS BASICFILE ( TABLESPACE SPMS_DATA ENABLE STORAGE IN ROW
CHUNK 8192 RETENTION NOCACHE LOGGING STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1
MAXEXTENTS 2147483645  FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE
DEFAULT CELL_FLASH_CACHE DEFAULT))*/
;
```

PCTINCREASE

The storage parameter **PCTINCREASE** is not supported for all the tables. In addition, all storage parameters (like pctfree, minextents, maxextents) are not allowed for partitioned tables.

Input - TABLE with PCTINCREASE

```
CREATE TABLE tab1 (
          col1 < datatype >
        , col2 < datatype >
        , ...
        , colN < datatype > )
       TABLESPACE testts
       PCTFREE 10 INITRANS 1 MAXTRANS
       255
       /* STORAGE (
       INITIAL 5 M NEXT 5 M MINEXTENTS 1 MAXEXTENTS UNLIMITED PCTINCREASE 0 );*/
```

Output

```
CREATE TABLE tab1 (
          col1 < datatype >
        , col2 < datatype >
```



```
        , ...
        , colN < datatype >  )
          TABLESPACE testts
          PCTFREE 10 INITRANS 1 MAXTRANS 255
          /* STORAGE (
          INITIAL 5 M NEXT 5 M MINEXTENTS 1 MAXEXTENTS
          UNLIMITED );*/
```

FOREIGN KEY

A foreign key is a way to enforce referential integrity within an Oracle database. A foreign key means that values in one table must also appear in another table. The referenced table is called the parent table while the table with the foreign key is called the child table. The foreign key in the child table will generally reference a primary key in the parent table. A foreign key can be defined in either a CREATE TABLE statement or an ALTER TABLE statement.

A foreign key constraint must be established with the REFERENCE clause. An inline constraint clause appears as part of the column definition clause or the object properties clause. An out-of-line constraint appears as part of a relational properties clause or the object properties clause.

If the configuration parameter **foreignKeyHandler** is set to **true** (default value), then the tool will migrate these statements into commented statements.

DSC supports inline and out-of-line foreign key constraints as shown in the following examples.

Input - Foreign Key with inline constraint in CREATE TABLE

```
CREATE TABLE orders (
    order_no    INT  NOT NULL PRIMARY KEY,
    order_date   DATE  NOT NULL,
    cust_id    INT
 [CONSTRAINT fk_orders_cust]
    REFERENCES customers(cust_id)
   [ON DELETE SET NULL]
 [INITIALLY DEFERRED]
 [ENABLE NOVALIDATE]
);
```

Output

```
CREATE TABLE orders (
    order_no    INT  NOT NULL PRIMARY KEY,
    order_date   DATE  NOT NULL,
    cust_id    INT
/*
  [CONSTRAINT fk_orders_cust]
   REFERENCES customers(cust_id)
   [ON DELETE SET NULL]
   [INITIALLY DEFERRED]
   [ENABLE NOVALIDATE] */
);
```

Input - Foreign Key with out-of-line constraint in CREATE TABLE

```
CREATE TABLE customers (
       cust_id   INT   NOT NULL,
      cust_name  VARCHAR(64) NOT NULL,
      cust_addr  VARCHAR(256),
 cust_contact_no  VARCHAR(16),
    PRIMARY KEY (cust_id)
);
```



```
CREATE TABLE orders (
    order_no    INT  NOT NULL,
    order_date  DATE  NOT NULL,
    cust_id     INT  NOT NULL,
    PRIMARY KEY (order_no),
     CONSTRAINT fk_orders_cust
    FOREIGN KEY (cust_id)
     REFERENCES customers(cust_id)
      ON DELETE CASCADE
);
```

Output

```
CREATE TABLE customers (
       cust_id   INT   NOT NULL,
      cust_name  VARCHAR(64) NOT NULL,
      cust_addr  VARCHAR(256),
 cust_contact_no  VARCHAR(16),
    PRIMARY KEY (cust_id)
);

CREATE TABLE orders (
     order_no  INT  NOT NULL,
   order_date  DATE  NOT NULL,
      cust_id  INT  NOT NULL,
    PRIMARY KEY (order_no) /*,
     CONSTRAINT fk_orders_cust
    FOREIGN KEY (cust_id)
     REFERENCES customers(cust_id)
      ON DELETE CASCADE */
);
```

LONG Data Type

Columns defined as LONG can store variable-length character data containing up to two gigabytes of information. The tool supports LONG data types in TABLE structure and PL/SQL.

Input - LONG data type in table structure

```
CREATE TABLE project ( proj_cd INT
                     , proj_name VARCHAR2(32)
                     , dept_no INT
                     , proj_det LONG );
```

Output

```
CREATE TABLE project ( proj_cd INT
                      , proj_name VARCHAR2(32)
                      , dept_no INT
                      , proj_det TEXT );
```

Input - LONG data type in PL/SQL

```
CREATE OR REPLACE FUNCTION fn_proj_det
                       ( i_proj_cd INT )
RETURN LONG
IS
   v_proj_det LONG;
BEGIN
    SELECT proj_det
      INTO v_proj_det
      FROM project
     WHERE proj_cd = i_proj_cd;

   RETURN v_proj_det;
END;
/
```


Output

```
CREATE OR REPLACE FUNCTION fn_proj_det
                        ( i_proj_cd INT )
RETURN TEXT
IS
  v_proj_det TEXT;
BEGIN
    SELECT proj_det
      INTO v_proj_det
      FROM project
     WHERE proj_cd = i_proj_cd;

  RETURN v_proj_det;
END;
/
```

TYPE

MDSYS.MBRCOORDLIST should be replaced with CLOB.

Oracle Syntax	Syntax After Migration

create table product_part
(
  partid   VARCHAR2(24),
  mbrcoords MDSYS.MBRCOORDLIST
);
	CREATE TABLE product_part
	(
	  partid VARCHAR2(24),
	  mbrcoords CLOB
	);

MDSYS.SDO_GEOMETRY should be replaced with CLOB.

Oracle Syntax	Syntax After Migration

create table product_part
(
  partid   VARCHAR2(24),
  shape MDSYS.SDO_GEOMETRY
);
	CREATE TABLE product_part
	(
	  partid VARCHAR2(24),
	  shape CLOB
	);

GEOMETRY should be replaced with CLOB.

Oracle Syntax	Syntax After Migration

create table product_part
(
  partid   VARCHAR2(24),
  shape GEOMETRY
);
	CREATE TABLE product_part
	(
	  partid VARCHAR2(24),
	  shape CLOB
	);

Columns

xmax, **xmin**, **left**, **right** and **maxvalue** are GaussDB(DWS) keywords and should be concatenated with double quotes in upper case.

Oracle Syntax	Syntax After Migration

create table product
(
  xmax VARCHAR2(20),
  xmin VARCHAR2(50),
  left VARCHAR2(50),
  right VARCHAR2(50),
  maxvalue VARCHAR2(50)
);
	CREATE TABLE product1
	(
	  "XMAX" VARCHAR2(20),
	  "XMIN" VARCHAR2(50),
	  "LEFT" VARCHAR2(50),
	  "RIGHT" VARCHAR2(50),
	  "MAXVALUE" VARCHAR2(50)
	);

Interval Partition

Partition should be commented for interval partition.

Oracle Syntax	Syntax After Migration

CREATE TABLE product
(
  product_id VARCHAR2(20),
  product_name VARCHAR2(50),
  manufacture_month DATE
)
partition by range (manufacture_month) interval (NUMTODSINTERVAL (1, 'MONTH'))
(
  partition T_PARTITION_2018_11_LESS values less than (TO_DATE(' 2018-11-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS')));
	CREATE TABLE product
	(
	  product_id VARCHAR2(20),
	  product_name VARCHAR2(50),
	  manufacture_month DATE
	)
	/*partition by range (manufacture_month) interval (NUMTODSINTERVAL (1, 'MONTH'))
	(
	  partition T_PARTITION_2018_11_LESS values less than (TO_DATE(' 2018-11-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS')))*/;

6.6.2.2 Temporary Tables

GaussDB(DWS) does not support **GLOBAL TEMPORARY TABLE**, It migrates **GLOBAL TEMPORARY TABLE** to **LOCAL TEMPORARY TABLE**.

ON COMMIT DELETE ROWS is also not supported and will be migrated to **ON COMMIT PRESERVE ROWS**.

The following is an example of the syntax of a temporary table before and after migration.

Pre-migration

Figure 6-6 GLOBAL TEMPORARY TABLE and ON COMMIT DELETE ROWS

```
CREATE
    GLOBAL TEMPORARY TABLE
        schema1.temp_tbl1 (
            col1 VARCHAR2 (400)
            ,col2 DATE NOT NULL
        )
            ON COMMIT DELETE ROWS
;
```

```
CREATE
    GLOBAL TEMPORARY TABLE
```



```
            schema1.temp_tbl1
           (
                col1 VARCHAR2 (400),
                col2 DATE NOT NULL
            )
              ON COMMIT DELETE ROWS
;
```

Post-migration

Figure 6-7 LOCAL TEMPORARY TABLE and ON COMMIT PRESERVE ROWS

```
CREATE
    LOCAL TEMPORARY TABLE
        schema1_temp_tbl1 (
            col1 VARCHAR2 (400)
            ,col2 DATE NOT NULL
        )
            ON COMMIT PRESERVE ROWS
;
```

```
CREATE
    LOCAL TEMPORARY TABLE
            schema1_temp_tbl1
           (
                col1 VARCHAR2 (400),
                col2 DATE NOT NULL
            )
              ON COMMIT PRESERVE ROWS
;
```

6.6.2.3 Global Temporary Tables

Global temporary tables are converted to local temporary tables.

Input - GLOBAL TEMPORARY TABLE

```
CREATE GLOBAL TEMPORARY TABLE
"Pack1"."GLOBAL_TEMP_TABLE"

( "ID" VARCHAR2(8)

) ON COMMIT DELETE ROWS ;
```

Output

```
CREATE
    LOCAL TEMPORARY TABLE

"Pack1_GLOBAL_TEMP_TABLE" (

"ID" VARCHAR2 (8)
        )
         ON COMMIT PRESERVE ROWS ;
```

6.6.2.4 Indexes

When an index is created in GaussDB(DWS), a schema name cannot be specified along with the index name. The index will be automatically created in the schema where the index table is created.

Figure 6-8 Input: INDEX

```
CREATE
    INDEX scott.ix_tab1_col1
        ON scott.tab1 (col1) tablespace users pctfree 10 initrans 2 maxtrans 255 storage (
        initial 256 K NEXT 256 K minextents 1 maxextents unlimited
    )
```

Figure 6-9 Output: INDEX

```
CREATE
    INDEX ix_tab1_col1
        ON scott.tab1 (col1) tablespace users pctfree 10 initrans 2 maxtrans 255 storage (
        initial 256 K NEXT 256 K minextents 1 maxextents unlimited
    )
```

Input - Function-based indexes by using CASE

A function-based index is an index that is created on the results of a function or expression on top of a column.

Output

```
CREATE
   UNIQUE index GCC_RSRC_ASSIGN_U1
      ON GCC_PLAN.GCC_RSRC_ASSIGN_T (
      (CASE
         WHEN( ENABLE_FLAG = 'Y' AND ASSIGN_TYPE = '13' AND WORK_ORDER_ID IS NOT NULL )
         THEN WORK_ORDER_ID
         ELSE NULL
      END)
   ) ;
```

NOTE

The expression or function needs to be put inside brackets.

Input - Function-based indexes by using DECODE

```
CREATE UNIQUE index GCC_PLAN_N2
      ON GCC_PLAN.GCC_PLAN_T (
      DECODE (
         ENABLE_FLAG
         ,'Y'
         ,BUSINESS_ID
         ,NULL
      )
   ) ;
```

Output

```
CREATE UNIQUE index GCC_PLAN_N2
      ON GCC_PLAN.GCC_PLAN_T (
      (DECODE (
         ENABLE_FLAG
         ,'Y'
         ,BUSINESS_ID
         ,NULL
      ))
   ) ;
```

NOTE

The expression or function needs to be put inside brackets.

ORA_HASH

ORA_HASH is a function that computes a hash value for a given expression or column. If this function is specified on the column(s) in the CREATE INDEX statement, this function will be removed.

Input

```
CREATE INDEX SD_WO.WO_WORK_ORDER_T_N3 on SD_WO.WO_WORK_ORDER_T (PROJECT_NUMBER,
ORA_HASH(WORK_ORDER_NAME));
```

Output

```
CREATE
index WO_WORK_ORDER_T_N3
ON SD_WO.WO_WORK_ORDER_T (
PROJECT_NUMBER
,ORA_HASH( WORK_ORDER_NAME )
) ;
```

DECODE

If DECODE function in the CREATE INDEX statement is used as a part of a column, the following error will be reported: "syntax error at or near 'DECODE' (Script - gcc_plan_t.sql)".

Input

```
CREATE UNIQUE index GCC_PLAN.GCC_PLAN_N2 on GCC_PLAN.GCC_PLAN_T
(DECODE(ENABLE_FLAG,'Y',BUSINESS_ID,NULL));
```

Output

```
CREATE
UNIQUE index GCC_PLAN_N2
ON GCC_PLAN.GCC_PLAN_T (
DECODE (
ENABLE_FLAG
,'Y'
,BUSINESS_ID
,NULL
)
) ;
```

CASE statement

The CASE statement is not supported in the CREATE INDEX statement.

Input

```
CREATE
    UNIQUE index GCC_RSRC_ASSIGN_U1
        ON GCC_PLAN.GCC_RSRC_ASSIGN_T (
        (CASE
            WHEN( ENABLE_FLAG = 'Y' AND ASSIGN_TYPE = '13' AND WORK_ORDER_ID IS NOT NULL )
            THEN WORK_ORDER_ID
            ELSE NULL
        END)
    ) ;
```

Output

```
CREATE UNIQUE INDEX gcc_rsrc_assign_u1
  ON gcc_plan.gcc_rsrc_assign_t ( (( CASE
                                        WHEN( enable_flag = 'Y'
                                              AND assign_type = '13'
                                              AND work_order_id IS NOT NULL )
                                    THEN work_order_id
                                    ELSE NULL END )) );
```


6.6.2.5 Views

A view is a logical table based on one or more tables or views. A view itself contains no data.

In the source file, if the table names are not qualified with the schema name, then the target file is modified such that the table is also qualified with the same schema name as that of the view.

The following is an example of the syntax before and after view migration.

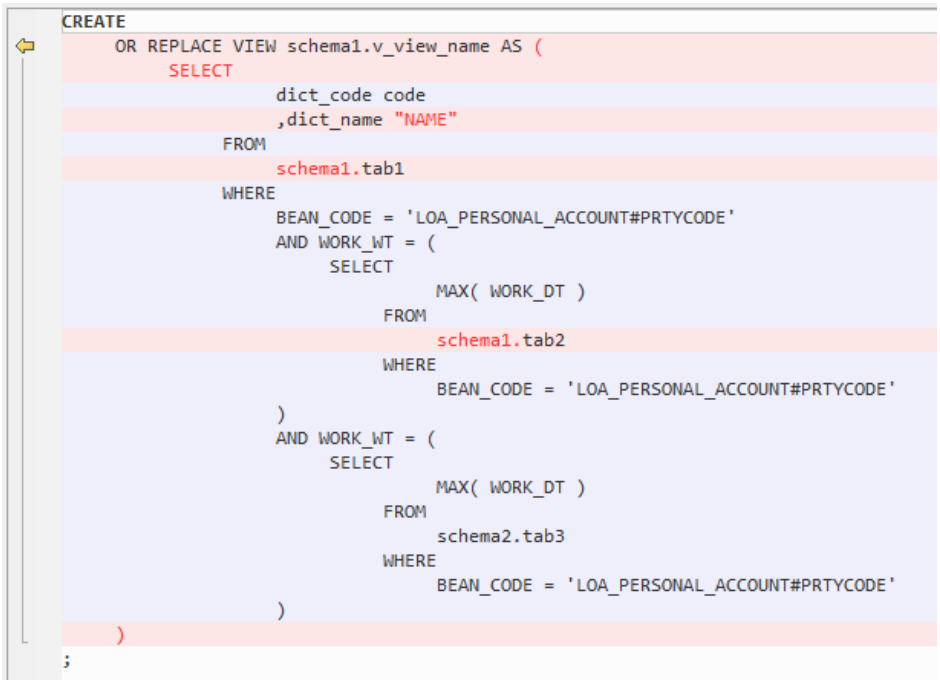
Table Name (Without Schema Name)

Figure 6-10 Input: tab1 and tab2

```
CREATE
    OR REPLACE VIEW schema1.v_view_name AS SELECT
                dict_code code
                ,dict_name name
          FROM
                tab1
          WHERE
                BEAN_CODE = 'LOA_PERSONAL_ACCOUNT#PRTYCODE'
                AND WORK_WT = (
                     SELECT
                               MAX( WORK_DT )
                          FROM
                               tab2
                          WHERE
                               BEAN_CODE = 'LOA_PERSONAL_ACCOUNT#PRTYCODE'
                )
                AND WORK_WT = (
                     SELECT
                               MAX( WORK_DT )
                          FROM
                               schema2.tab3
                          WHERE
                               BEAN_CODE = 'LOA_PERSONAL_ACCOUNT#PRTYCODE'
                )
;
```


Table Name (With Schema Name)

Figure 6-11 Output: view schema names



```
CREATE
    OR REPLACE VIEW schema1.v_view_name AS (
        SELECT
                dict_code code
                ,dict_name "NAME"
            FROM
                schema1.tab1
            WHERE
                BEAN_CODE = 'LOA_PERSONAL_ACCOUNT#PRTYCODE'
                AND WORK_WT = (
                    SELECT
                            MAX( WORK_DT )
                        FROM
                            schema1.tab2
                        WHERE
                            BEAN_CODE = 'LOA_PERSONAL_ACCOUNT#PRTYCODE'
                )
                AND WORK_WT = (
                    SELECT
                            MAX( WORK_DT )
                        FROM
                            schema2.tab3
                        WHERE
                            BEAN_CODE = 'LOA_PERSONAL_ACCOUNT#PRTYCODE'
                )
    )
;
```

6.6.2.6 Sequences

A sequence is an Oracle object used to generate a number sequence. This can be useful when you need to create an autonumber column to act as a primary key.

If **MigSupportSequence** is set to **true** (default), a sequence is created in the **PUBLIC** schema.

NOTE

- **CACHE** and **ORDER** cannot be migrated.
- In Oracle, the maximum value of **MAXVALUE** can be set to **999999999999999999999999999**. In GaussDB(DWS), the maximum value of **MAXVALUE** can be set to **9223372036854775807**.
- Before migrating a sequence, copy the content in the **sequence_scripts.sql** file and paste it to execute the script in all the target databases. For details, see **Executing Custom DB Scripts**.

Sequence

Input - CREATE SEQUENCE

```
CREATE SEQUENCE GROUP_DEF_SEQUENCE
  minvalue 1
  maxvalue 10000000000000000000
  start with 1152
  increment by 1
  cache 50
  order;
```

Output


```
INSERT
    INTO
        PUBLIC.MIG_SEQ_TABLE (
            SCHEMA_NAME
            ,SEQUENCE_NAME
            ,START_WITH
            ,INCREMENT_BY
            ,MIN_VALUE
            ,MAX_VALUE
            ,CYCLE_I
            ,CACHE
            ,ORDER_I
        )
    VALUES (
        UPPER( current_schema ( ) )
        ,UPPER( 'GROUP_DEF_SEQUENCE' )
        ,1152
        ,1
        ,1
        ,9223372036854775807
        ,FALSE
        ,20
        ,FALSE
    )
;
```

SEQUENCE with NOCACHE

Input - **CREATE SEQUENCE** with **NOCACHE**

```
CREATE SEQUENCE customers_seq
 START WITH 1000
 INCREMENT BY   1
 NOCACHE
 NOCYCLE;
```

Output

```
INSERT
    INTO
        PUBLIC.MIG_SEQ_TABLE (
            SCHEMA_NAME
            ,SEQUENCE_NAME
            ,START_WITH
            ,INCREMENT_BY
            ,MIN_VALUE
            ,MAX_VALUE
            ,CYCLE_I
            ,CACHE
            ,ORDER_I
        )
    VALUES (
        UPPER( current_schema ( ) )
        ,UPPER( 'customers_seq' )
        ,1000
        ,1
        ,1
        ,9999999999999999999999999999
        ,FALSE
        ,20
        ,FALSE
    )
;
```

Input - CREATE SEQUENCE with a specified schema name

Input - **CREATE SEQUENCE** with a specified schema name


```
CREATE SEQUENCE scott.seq_customers
 START WITH 1000 INCREMENT BY 1
 MINVALUE 1000 MAXVALUE 999999999999999
 CACHE 20 CYCLE ORDER;
```

Output

```
INSERT
    INTO
        PUBLIC.MIG_SEQ_TABLE (
            SCHEMA_NAME
            ,SEQUENCE_NAME
            ,START_WITH
            ,INCREMENT_BY
            ,MIN_VALUE
            ,MAX_VALUE
            ,CYCLE_I
            ,CACHE
            ,ORDER_I
        )
    VALUES (
        UPPER( 'scott' )
        ,UPPER( 'seq_customers' )
        ,1000
        ,1
        ,1000
        ,999999999999999
        ,TRUE
        ,20
        ,FALSE
    )
;
```

CREATE SEQUENCE with a Default Value

Input - **SEQUENCE** with a default value

```
CREATE SEQUENCE seq_orders;
```

Output

```
INSERT
    INTO
        PUBLIC.MIG_SEQ_TABLE (
            SCHEMA_NAME
            ,SEQUENCE_NAME
            ,START_WITH
            ,INCREMENT_BY
            ,MIN_VALUE
            ,MAX_VALUE
            ,CYCLE_I
            ,CACHE
            ,ORDER_I
        )
    VALUES (
        UPPER( current_schema ( ) )
        ,UPPER( 'seq_orders' )
        ,1
        ,1
        ,1
        ,9999999999999999999999999999
        ,FALSE
        ,20
        ,FALSE
    )
;
```


NEXTVAL

To migrate the NEXTVAL function, a custom function is provided for generating the next value based on **increment_by**, **max_value**, **min_value**, and **cycle**. During the DSC installation, this function should be created in all the databases where the migration is to be performed.

NEXTVAL supports all GaussDB(DWS) versions.

NEXTVAL is a system function of Oracle and is not implicitly supported by GaussDB(DWS). To support this function, DSC creates a **NEXTVAL** function in the **PUBLIC** schema. The **PUBLIC.NEXTVAL** function is used in the migrated statements.

NOTE

If **MigSupportSequence** is set to **true**, NEXTVAL is migrated to PUBLIC.NEXTVAL('[schema].sequence').

If **MigSupportSequence** is set to **false**, NEXTVAL is migrated to NEXTVAL('[schema].sequence').

Before migrating the NEXTVAL function, copy the content in the **sequence_scripts.sql** file and paste it to execute the script in all the target databases. For details, see **Executing Custom DB Scripts**.

Input - NEXTVAL

```
[schema.]sequence.NEXTVAL
```

Output

```
PUBLIC.nextval('[schema.]sequence')
```

Input - NEXTVAL

```
SELECT
        EMP_ID_SEQ.NEXTVAL INTO
            SEQ_NUM
        FROM
            dual
;
```

Output

```
SELECT
        PUBLIC.NEXTVAL ('EMP_ID_SEQ') INTO
            SEQ_NUM
        FROM
            dual
;
```

CURRVAL

To migrate the CURRVAL function, you can customize one to return the current value of a sequence. During the DSC installation, this function should be created in all the databases where the migration is to be performed.

CURRVAL is a system function of Oracle and is not implicitly supported by GaussDB(DWS). To support this function, DSC creates a **CURRVAL** function in the **PUBLIC** schema. The **PUBLIC.CURRVAL** function is used in the migrated statements.

NOTE

If **MigSupportSequence** is set to **true**, CURRVAL is migrated to PUBLIC.CURRVAL('[schema].sequence').

If **MigSupportSequence** is set to **false**, CURRVAL is migrated to CURRVAL('[schema].sequence').

Before migrating the NEXTVAL function, copy the content in the **sequence_scripts.sql** file and paste it to execute the script in all the target databases. For details, see **Executing Custom DB Scripts**.

Input - CURRVAL

```
[schema.]sequence.CURRVAL
```

Output

```
currval('[schema.]sequence')
```

Input - CURRVAL

```
INSERT
    INTO
        Line_items_tab (
            Orderno
            ,Partno
            ,Quantity
        )
    VALUES (
        Order_seq.CURRVAL
        ,20321
        ,3
    )
;
```

Output

```
INSERT
    INTO
        Line_items_tab (
            Orderno
            ,Partno
            ,Quantity
        ) SELECT
                PUBLIC.CURRVAL ('Order_seq')
                ,20321
                ,3
;
```

6.6.2.7 PURGE

In Oracle, the **DROP TABLE** statement moves a table to the recycle bin, allowing for potential recovery. In contrast, the **PURGE** statement permanently deletes a table or index from the recycle bin, releasing all associated space. It can also empty the entire recycle bin or permanently remove specific contents from a deleted tablespace.

After migration using **Oracle** syntax, the query does not include **PURGE**.

The following examples illustrate the syntax changes for **PURGE** before and after migration.

Pre-migration

Figure 6-12 Input: statement containing PURGE

```
Execute immediate 'Drop table
table1 purge' ;

drop table test.emp purge ;
```

Post-migration

Figure 6-13 Output: statement without PURGE

```
Execute immediate 'Drop table table1' ;


drop table test.emp ;
```

6.6.2.8 Database Keywords

DSC supports GaussDB(DWS) keywords, such as **NAME**, **LIMIT**, **OWNER**, **KEY**, and **CAST**. These keywords must be enclosed in double quotation marks.

GaussDB(DWS) Keywords (NAME, VERSION, LABEL, and POSITION)

The keywords **NAME**, **VERSION**, **LABEL**, and **POSITION** are changed to **AS***Keyword*.

Input – NAME, VERSION, LABEL, POSITION

```
SELECT id, NAME,label,description
       FROM (SELECT a.id             id,
                    b.NAME           NAME,
                    b.description    description,
                    b.default_label  label,
                    ROWNUM           ROW_ID
             FROM CTP_ITEM A
             LEFT OUTER JOIN CTP_ITEM_NLS B ON A.ID = B.ID
                                       AND B.LOCALE = i_language
            ORDER BY a.id ASC)
      WHERE ROW_ID >= to_number(begNum)
        AND ROW_ID < to_number(begNum) + to_number(fetchNum);

SELECT DISTINCT REPLACE(VERSION,' ','') ID, VERSION TEXT
       FROM (SELECT T1.SOFTASSETS_NAME, T2.VERSION
             FROM SPMS_SOFT_ASSETS T1, SPMS_SYSSOFT_ASSETS T2
            WHERE T1.SOFTASSETS_ID = T2.SOFTASSETS_ID)
        WHERE SOFTASSETS_NAME = I_SOFT_NAME;

SELECT COUNTRY, AMOUNT
        FROM (SELECT '' COUNTRY || '' AMOUNT, '1' POSITION
              FROM DUAL )
       ORDER BY POSITION;
```

Output

```
SELECT id,NAME,label,description FROM (
 SELECT a.id id,b.NAME AS NAME,
 b.description description
```



```
 ,b.default_label AS label,
 ROW_NUMBER( ) OVER( ) ROW_ID
 FROM CTP_ITEM A LEFT OUTER JOIN
 CTP_ITEM_NLS B
 ON A.ID = B.ID AND
 B.LOCALE = i_language
 ORDER BY a.id ASC) WHERE
 ROW_ID >= to_number( begNum )
 AND
 ROW_ID < to_number( begNum ) + to_number( fetchNum )
;

SELECT
  DISTINCT REPLACE( VERSION ,' ' ,'' ) ID
    ,VERSION AS TEXT
    FROM
      (
       SELECT
       T1.SOFTASSETS_NAME
       ,T2.VERSION
       FROM
      SPMS_SOFT_ASSETS T1
     ,SPMS_SYSSOFT_ASSETS T2
      WHERE
      T1.SOFTASSETS_ID = T2.SOFTASSETS_ID
     )
     WHERE SOFTASSETS_NAME = I_SOFT_NAME ;



SELECT COUNTRY ,AMOUNT
FROM ( SELECT '' COUNTRY || '' AMOUNT
       ,'1' AS POSITION
         FROM
         DUAL
      )
    ORDER BY
    POSITION
;
```

TEXT & YEAR

Input – TEXT, YEAR

```
SELECT
  NAME,
  VALUE,
  DESCRIPTION TEXT,
  JOINED YEAR,
  LIMIT
FROM
  EMPLOYEE;

SELECT
  NAME,
  TEXT,
  YEAR,
  VALUE,
  DESCRIPTION,
  LIMIT
FROM
  EMPLOYEE_DETAILS;
```

Output

```
SELECT
  "NAME",
  VALUE,
  DESCRIPTION AS TEXT,
```



```
 JOINED AS YEAR,
 "LIMIT"
FROM
 EMPLOYEE;

SELECT
 "NAME",
 "TEXT",
 "YEAR",
 VALUE,
 DESCRIPTION,
 "LIMIT"
FROM
 EMPLOYEE_DETAILS;
```

NAME and LIMIT

Input: GaussDB(DWS) keywords NAME and LIMIT

```
CREATE TABLE NAME
     ( NAME VARCHAR2(50) NOT NULL
     , VALUE VARCHAR2(255)
     , DESCRIPTION VARCHAR2(4000)
     , LIMIT NUMBER(9)
     )
 /*TABLESPACE users*/
 pctfree 10   initrans 1   maxtrans
 255
    storage ( initial 256K next 256K
    minextents 1 maxextents
    unlimited );

SELECT NAME, VALUE, DESCRIPTION, LIMIT
  FROM NAME;
```

Output

```
CREATE TABLE "NAME"
        ( "NAME" VARCHAR2 (50) NOT NULL
        , VALUE VARCHAR2 (255)
        , DESCRIPTION VARCHAR2 (4000)
        , "LIMIT" NUMBER (9)
        )
     /*TABLESPACE users*/
     pctfree 10 initrans 1 maxtrans 255
     storage ( initial 256 K NEXT 256 K minextents 1
     maxextents unlimited );

SELECT "NAME", VALUE, DESCRIPTION, "LIMIT"
  FROM "NAME";
```

OWNER

Bulk Operations

Input: Use SELECT to query the GaussDB(DWS) keyword OWNER

```
SELECT
        owner
    FROM
        Test_Col;
```

Output

```
SELECT
        "OWNER"
    FROM
        Test_Col;
```


Input: Use DELETE to query the GaussDB(DWS) keyword OWNER

```
DELETE FROM emp14
    WHERE
        ename = 'Owner';
```

Input

```
DELETE FROM emp14
    WHERE
        ename = 'Owner'
```

KEY

Blogic Operations

Input: GaussDB(DWS) keyword KEY

```
CREATE
    OR REPLACE FUNCTION myfct RETURN VARCHAR2 parallel_enable IS res VARCHAR2 ( 200 ) ;
    BEGIN
        res := 100 ;
        INSERT INTO emp18 RW ( RW.empno ,RW.ename ) SELECT
            res ,RWN.ename KEY
        FROM
            emp16 RWN ;
            COMMIT ;
        RETURN res ;
END ;
/
```

Output

```
CREATE
    OR REPLACE FUNCTION myfct RETURN VARCHAR2 IS res VARCHAR2 ( 200 ) ;
    BEGIN
        res := 100 ;
        INSERT INTO emp18 ( empno ,ename ) SELECT
            res ,RWN.ename "KEY"
        FROM
            emp16 RWN ;
            /* COMMIT; */
        null ;
        RETURN res ;
END ;
```

Range, Account and Language

When the GaussDB(DWS) keyword is used as the alias of any column in the **SELECT** list and AS is not used, define the alias in the **AS***Keyword* format.

Input

```
CREATE
    OR REPLACE /*FORCE*/
    VIEW SAD.FND_TERRITORIES_TL_V (
        TERRITORY_CODE
        ,TERRITORY_SHORT_NAME
        ,LANGUAGE
        ,Account
        ,Range
        ,LAST_UPDATED_BY
        ,LAST_UPDATE_DATE
        ,LAST_UPDATE_LOGIN
        ,DESCRIPTION
        ,SOURCE_LANG
        ,ISO_NUMERIC_CODE
```



```
) AS SELECT
        t.TERRITORY_CODE
        ,t.TERRITORY_SHORT_NAME
        ,t.LANGUAGE
        ,t.Account
        ,t.Range
        ,t.LAST_UPDATED_BY
        ,t.LAST_UPDATE_DATE
        ,t.LAST_UPDATE_LOGIN
        ,t.DESCRIPTION
        ,t.SOURCE_LANG
        ,t.ISO_NUMERIC_CODE
    FROM
        fnd_territories_tl t
UNION
ALL SELECT
        'SS' TERRITORY_CODE
        ,'Normal Country' TERRITORY_SHORT_NAME
        ,NULL LANGUAGE
        ,NULL Account
        ,NULL Range
        ,NULL LAST_UPDATED_BY
        ,NULL LAST_UPDATE_DATE
        ,NULL LAST_UPDATE_LOGIN
        ,NULL DESCRIPTION
        ,NULL SOURCE_LANG
        ,NULL ISO_NUMERIC_CODE
    FROM
        DUAL ;
```

Output

```
CREATE
    OR REPLACE /*FORCE*/
    VIEW SAD.FND_TERRITORIES_TL_V (
        TERRITORY_CODE
        ,TERRITORY_SHORT_NAME
        ,LANGUAGE
        ,CREATED_BY
        ,CREATION_DATE
        ,LAST_UPDATED_BY
        ,LAST_UPDATE_DATE
        ,LAST_UPDATE_LOGIN
        ,DESCRIPTION
        ,SOURCE_LANG
        ,ISO_NUMERIC_CODE
    ) AS SELECT
        t.TERRITORY_CODE
        ,t.TERRITORY_SHORT_NAME
        ,t.LANGUAGE
        ,t.CREATED_BY
        ,t.CREATION_DATE
        ,t.LAST_UPDATED_BY
        ,t.LAST_UPDATE_DATE
        ,t.LAST_UPDATE_LOGIN
        ,t.DESCRIPTION
        ,t.SOURCE_LANG
        ,t.ISO_NUMERIC_CODE
    FROM
        fnd_territories_tl t
    UNION
    ALL SELECT
        'SS' TERRITORY_CODE
        ,'Normal Country' TERRITORY_SHORT_NAME
        ,NULL AS LANGUAGE
        ,NULL CREATED_BY
        ,NULL CREATION_DATE
        ,NULL LAST_UPDATED_BY
        ,NULL LAST_UPDATE_DATE
        ,NULL LAST_UPDATE_LOGIN
```



```
            ,NULL DESCRIPTION
            ,NULL SOURCE_LANG
            ,NULL ISO_NUMERIC_CODE
        FROM
            DUAL ;
```

Primary Key and Unique Key

If primary and unique keys are declared on table creation, only the primary key needs to consider for migration.

```
create table SD_WO.WO_DU_TRIGGER_REVENUE_T
(
  TRIGGER_REVENUE_ID NUMBER not null,
  PROJECT_NUMBER     VARCHAR2(40),
  DU_ID              NUMBER,
  STANDARD_MS_CODE   VARCHAR2(100),
  TRIGGER_STATUS     NUMBER,
  TRIGGER_MSG        VARCHAR2(4000),
  BATCH_NUMBER       NUMBER,
  PROCESS_STATUS     NUMBER,
  ENABLE_FLAG        CHAR(1) default 'Y',
  CREATED_BY         NUMBER,
  CREATION_DATE      DATE,
  LAST_UPDATE_BY     NUMBER,
  LAST_UPDATE_DATE   DATE
)
;

alter table SD_WO.WO_DU_TRIGGER_REVENUE_T
  add constraint WO_DU_TRIGGER_REVENUE_PK primary key (TRIGGER_REVENUE_ID);
alter table SD_WO.WO_DU_TRIGGER_REVENUE_T
  add constraint WO_DU_TRIGGER_REVENUE_N1 unique (DU_ID, STANDARD_MS_CODE);
```

Output

```
CREATE
    TABLE
        SD_WO.WO_DU_TRIGGER_REVENUE_T (
            TRIGGER_REVENUE_ID NUMBER NOT NULL
            ,PROJECT_NUMBER VARCHAR2 (40)
            ,DU_ID NUMBER
            ,STANDARD_MS_CODE VARCHAR2 (100)
            ,TRIGGER_STATUS NUMBER
            ,TRIGGER_MSG VARCHAR2 (4000)
            ,BATCH_NUMBER NUMBER
            ,PROCESS_STATUS NUMBER
            ,ENABLE_FLAG CHAR( 1 ) DEFAULT 'Y'
            ,CREATED_BY NUMBER
            ,CREATION_DATE DATE
            ,LAST_UPDATE_BY NUMBER
            ,LAST_UPDATE_DATE DATE
            ,CONSTRAINT WO_DU_TRIGGER_REVENUE_PK PRIMARY KEY (TRIGGER_REVENUE_ID)
        ) ;
```

PROMPT

PROMPT should be converted to **\ECHO** supported by GaussDB(DWS).

Oracle Syntax	Syntax after Migration

prompt
prompt Creating table product
prompt ==============================
prompt
create table product
(
  product_id   VARCHAR2(20),
  product_name   VARCHAR2(50)
);
	\echo
	\echo Creating table product
	\echo ==============================
	\echo
	CREATE TABLE product
	(
	  product_id VARCHAR2(20),
	  product_name VARCHAR2(50)
	);

6.6.3 COMPRESS Phrase

This function is used to comment out the **COMPRESS** phrase by default during the migration.

Input – COMPRESS Phrase

```
CREATE TABLE test_tab (
  id            NUMBER(10)    NOT NULL,
  description   VARCHAR2(100) NOT NULL,
  created_date  DATE          NOT NULL,
  created_by    VARCHAR2(50)  NOT NULL,
  updated_date  DATE,
  updated_by    VARCHAR2(50)
)
NOCOMPRESS
PARTITION BY RANGE (created_date) (
  PARTITION test_tab_q1 VALUES LESS THAN (TO_DATE('01/04/2003', 'DD/MM/YYYY')) COMPRESS,
  PARTITION test_tab_q2 VALUES LESS THAN (MAXVALUE)
);
```

Output

```
CREATE
    TABLE
        test_tab (
            id NUMBER (10) NOT NULL
            ,description VARCHAR2 (100) NOT NULL
            ,created_date DATE NOT NULL
            ,created_by VARCHAR2 (50) NOT NULL
            ,updated_date DATE
            ,updated_by VARCHAR2 (50)
        ) /*NOCOMPRESS*/
        PARTITION BY RANGE (created_date) (
            PARTITION test_tab_q1
            VALUES LESS THAN (
                TO_DATE( '01/04/2003' ,'DD/MM/YYYY' )
            ) /*COMPRESS*/
            ,PARTITION test_tab_q2
            VALUES LESS THAN (MAXVALUE)
) ;
```

6.6.4 Bitmap Index

There is a configuration parameter that is introduced for this feature named **BitmapIndexSupport** which default value is **comment**, then the sample input and output are as follows:

Input – Bitmap index


```
CREATE BITMAP INDEX
emp_bitmap_idx
ON index_demo (gender);
```

Output

```
/*CREATE BITMAP INDEX emp_bitmap_idx ON index_demo (gender);*/
```

However, if the configuration parameter is set to **BTREE**, then the output is as follows:

Output

```
CREATE
    /*bitmap*/
    INDEX emp_bitmap_idx
        ON index_demo
        USING btree (gender) ;
```

6.6.5 Custom Tablespace

This section describes the syntax for migrating a custom tablespace.

Input – custom tablespace

```
CREATE
    TABLE
        SEAS_VERSION_DDL_REL_ORA (
            VERSION_ORA_ID VARCHAR2 (20)
            ,TAB_OBJ_ID VARCHAR2 (20)
            ,AUDIT_ID VARCHAR2 (20)
            ,DDL_SYS CLOB
            ,DDL_USER CLOB
            ,IF_CONFORM VARCHAR2 (3)
            ,DDL_TYPE_SYS VARCHAR2 (5)
            ,DDL_REN_REASON_SYS VARCHAR2 (4000)
            ,DDL_ERR_SYS VARCHAR2 (4000)
        ) SEGMENT CREATION IMMEDIATE PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
NOCOMPRESS LOGGING STORAGE (
            INITIAL 655360 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645 PCTINCREASE 0
FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
DEFAULT
        ) TABLESPACE DRMS LOB (DDL_SYS) STORE AS BASICFILE (
            TABLESPACE DRMS ENABLE STORAGE IN ROW CHUNK 8192 RETENTION NOCACHE LOGGING
STORAGE (
                INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645 PCTINCREASE 0
FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
DEFAULT
            )
        ) LOB (DDL_USER) STORE AS BASICFILE (
            TABLESPACE DRMS ENABLE STORAGE IN ROW CHUNK 8192 RETENTION NOCACHE LOGGING
STORAGE (
                INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645 PCTINCREASE 0
FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
DEFAULT
            )
        )
```

Output

```
CREATE
    TABLE
        SEAS_VERSION_DDL_REL_ORA (
            VERSION_ORA_ID VARCHAR2 (20)
            ,TAB_OBJ_ID VARCHAR2 (20)
            ,AUDIT_ID VARCHAR2 (20)
            ,DDL_SYS CLOB
            ,DDL_USER CLOB
```



```
            ,IF_CONFORM VARCHAR2 (3)
            ,DDL_TYPE_SYS VARCHAR2 (5)
            ,DDL_REN_REASON_SYS VARCHAR2 (4000)
            ,DDL_ERR_SYS VARCHAR2 (4000)
        ) /*SEGMENT CREATION IMMEDIATE*/
        /*PCTFREE 10*/
        /*PCTUSED 40*/
        /*INITRANS 1*/
        /*MAXTRANS 255*/
        /*NOCOMPRESS*/
        /*LOGGING*/
        /*STORAGE(INITIAL 655360 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645  FREELISTS 1
FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE DEFAULT)*/
        /*TABLESPACE DRMS */
        /*LOB (DDL_SYS) STORE AS BASICFILE ( TABLESPACE DRMS ENABLE STORAGE IN ROW CHUNK
8192 RETENTION NOCACHE LOGGING STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1
MAXEXTENTS 2147483645  FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE
DEFAULT CELL_FLASH_CACHE DEFAULT))*/
        /*LOB (DDL_USER) STORE AS BASICFILE ( TABLESPACE DRMS ENABLE STORAGE IN ROW CHUNK
8192 RETENTION NOCACHE LOGGING STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1
MAXEXTENTS 2147483645  FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT FLASH_CACHE
DEFAULT CELL_FLASH_CACHE DEFAULT))*/
;
```

6.6.6 Supplemental Log Data

Supplemental columns can be recorded in redo log files. The process of recording these additional columns is called supplemental logging. Oracle supports this function, but GaussDB(DWS) does not.

Input

```
CREATE TABLE sad.fnd_lookup_values_t
        (
  lookup_code_id NUMBER NOT NULL /* ENABLE */
  ,lookup_code VARCHAR2 (40) NOT NULL /* ENABLE */
  ,meaning       VARCHAR2 (100)
  ,other_meaning VARCHAR2 (100)
  ,order_by_no   NUMBER
  ,start_time    DATE DEFAULT SYSDATE NOT NULL /* ENABLE */
  ,end_time    DATE
  ,enable_flag CHAR( 1 ) DEFAULT 'Y' NOT NULL /* ENABLE */
  ,disable_date DATE
  ,created_by   NUMBER ( 15 ,0 ) NOT NULL /* ENABLE */
  ,creation_date DATE NOT NULL /* ENABLE */
  ,last_updated_by NUMBER ( 15 ,0 ) NOT NULL /* ENABLE */
  ,last_update_date DATE NOT NULL /* ENABLE */
  ,last_update_login NUMBER ( 15 ,0 ) DEFAULT 0 NOT NULL /* ENABLE */
  ,description    VARCHAR2 (500)
  ,lookup_type_id NUMBER NOT NULL/* ENABLE */
  ,attribute4 VARCHAR2 (250)
  ,supplemental log data (ALL) COLUMNS
  ) ;
```

Output

```
CREATE TABLE sad.fnd_lookup_values_t
        (
  lookup_code_id NUMBER NOT NULL /* ENABLE */
  ,lookup_code VARCHAR2 (40) NOT NULL /* ENABLE */
  ,meaning       VARCHAR2 (100)
  ,other_meaning VARCHAR2 (100)
  ,order_by_no   NUMBER
  ,start_time    DATE DEFAULT SYSDATE NOT NULL /* ENABLE */
  ,end_time    DATE
  ,enable_flag CHAR( 1 ) DEFAULT 'Y' NOT NULL /* ENABLE */
  ,disable_date DATE
  ,created_by   NUMBER ( 15 ,0 ) NOT NULL /* ENABLE */
```



```
,creation_date DATE NOT NULL /* ENABLE */
,last_updated_by NUMBER ( 15 ,0 ) NOT NULL /* ENABLE */
,last_update_date DATE NOT NULL /* ENABLE */
,last_update_login NUMBER ( 15 ,0 ) DEFAULT 0 NOT NULL /* ENABLE */
,description    VARCHAR2 (500)
,lookup_type_id NUMBER NOT NULL/* ENABLE */
,attribute4 VARCHAR2 (250)
/* ,supplemental log data (ALL) COLUMNS */
) ;
```

NOTE

The **SUPPLEMENTAL LOG DATA** functions that are not supported by GaussDB need to be commented out.

SUPPLEMENTAL LOG DATA is not supported by **CREATE TABLE** and needs to be commented out.

Input

```
CREATE TABLE SAD.FND_DATA_CHANGE_LOGS_T
  (    LOGID NUMBER,
       TABLE_NAME VARCHAR2(40) NOT NULL ENABLE,
       TABLE_KEY_COLUMNS VARCHAR2(200),
       TABLE_KEY_VALUES VARCHAR2(200),
       COLUMN_NAME VARCHAR2(40) NOT NULL ENABLE,
       COLUMN_CHANGE_FROM_VALUE VARCHAR2(200),
       COLUMN_CHANGE_TO_VALUE VARCHAR2(200),
       DESCRIPTION VARCHAR2(500),
        SUPPLEMENTAL LOG DATA (ALL) COLUMNS
  );
```

Output

```
CREATE TABLE sad.fnd_data_change_logs_t
  (
    logid                    NUMBER
    ,table_name              VARCHAR2 (40) NOT NULL /* ENABLE */
    ,table_key_columns       VARCHAR2 (200)
    ,table_key_values        VARCHAR2 (200)
    ,column_name             VARCHAR2 (40) NOT NULL /* ENABLE */
    ,column_change_from_value VARCHAR2 (200)
    ,column_change_to_value   VARCHAR2 (200)
    ,description             VARCHAR2 (500)
    /*, SUPPLEMENTAL LOG DATA (ALL) COLUMNS*/
  )
```

6.6.7 LONG RAW

"Data type LONG RAW" is not supported in the CREATE TABLE statement. LONG RAW data type needs to be replaced with Byte.

Input

```
CREATE TABLE SAD.WORKFLOWDEFS
   ( ID NUMBER(*,0),
 WF_NAME VARCHAR2(200),
 WF_DEFINITION LONG RAW,
 WF_VERSION NUMBER(*,0),
 WF_PUBLISH CHAR(1),
 WF_MAINFLOW CHAR(1),
 WF_APP_NAME VARCHAR2(20),
 CREATED_BY NUMBER,
 CREATION_DATE DATE,
 LAST_UPDATED_BY NUMBER,
 LAST_UPDATE_DATE DATE,
 WFDESC VARCHAR2(2000)
  );
```


Output

```
CREATE TABLE sad.workflowdefs
 (
   id                  NUMBER (38, 0),
   wf_name             VARCHAR2 (200),
   wf_definition       BYTEA,
   wf_version          NUMBER (38, 0),
   wf_publish          CHAR(1),
   wf_mainflow         CHAR(1),
   wf_app_name         VARCHAR2 (20),
   created_by          NUMBER,
   creation_date       DATE,
   last_updated_by     NUMBER,
   last_update_date DATE,
   wfdesc              VARCHAR2 (2000)
 );
```

6.6.8 SYS_GUID

SYS_GUID is a built-in function which returns the Global Unique Identifier (GUID) for a row in a table. It accepts no arguments and returns a RAW value of 16 bytes.

Input

```
CREATE TABLE sad.fnd_data_change_logs_t
 (
   logid                   NUMBER,
   table_name              VARCHAR2 (40) NOT NULL /* ENABLE */
   ,table_key_columns      VARCHAR2 (200),
table_key_values        VARCHAR2 (200),
column_name             VARCHAR2 (40) NOT NULL /* ENABLE */
,column_change_from_value VARCHAR2 (200),
column_change_to_value  VARCHAR2 (200),
organization_id         NUMBER,
created_by              NUMBER (15, 0) NOT NULL /* ENABLE */
,creation_date          DATE NOT NULL /* ENABLE */
,last_updated_by        NUMBER (15, 0) NOT NULL /* ENABLE */
,last_update_date       DATE NOT NULL /* ENABLE */
,last_update_login      NUMBER (15, 0) DEFAULT 0 NOT NULL /* ENABLE */
,description            VARCHAR2 (500),
sys_id                  VARCHAR2 (32) DEFAULT Sys_guid( )
/*, SUPPLEMENTAL LOG DATA (ALL) COLUMNS*/
 );
```

Output

```
CREATE TABLE sad.fnd_data_change_logs_t
 (
   logid                   NUMBER,
   table_name              VARCHAR2 (40) NOT NULL /* ENABLE */
   ,table_key_columns      VARCHAR2 (200),
table_key_values        VARCHAR2 (200),
column_name             VARCHAR2 (40) NOT NULL /* ENABLE */
,column_change_from_value VARCHAR2 (200),
column_change_to_value  VARCHAR2 (200),
organization_id         NUMBER,
created_by              NUMBER (15, 0) NOT NULL /* ENABLE */
,creation_date          DATE NOT NULL /* ENABLE */
,last_updated_by        NUMBER (15, 0) NOT NULL /* ENABLE */
,last_update_date       DATE NOT NULL /* ENABLE */
,last_update_login      NUMBER (15, 0) DEFAULT 0 NOT NULL /* ENABLE */
,description            VARCHAR2 (500),
sys_id                  VARCHAR2 (32) DEFAULT MIG_ORA_EXT.Sys_guid( )
/*, SUPPLEMENTAL LOG DATA (ALL) COLUMNS*/
 );
```


6.6.9 DML

This section describes the migration syntax of Oracle DML. The migration syntax decides how the keywords/features are migrated.

For details, see the following topics:

SELECT

INSERT

MERGE

SELECT

Overview

The Oracle **SELECT** statement starts a query, with an optional ORDER BY clause. The clause is used to retrieve records from one or more tables in a database.

Input - SELECT

```
SELECT col1, col2
  FROM tab1;
```

Output

```
SELECT col1, col2
  FROM tab1;
```

1. **Order of Clauses**

   The **HAVING** clause must follow the **GROUP BY** clause. However, Oracle allows **HAVING** to be in front of or behind the **GROUP BY** clause. In the target database, the **HAVING** clause is moved to behind the **GROUP BY** clause.

   Figure 6-14 Input - Order of Clauses

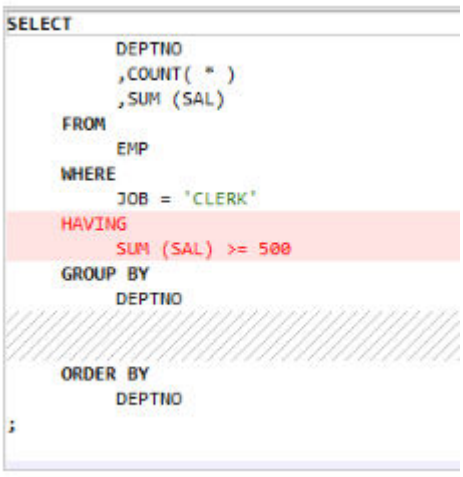
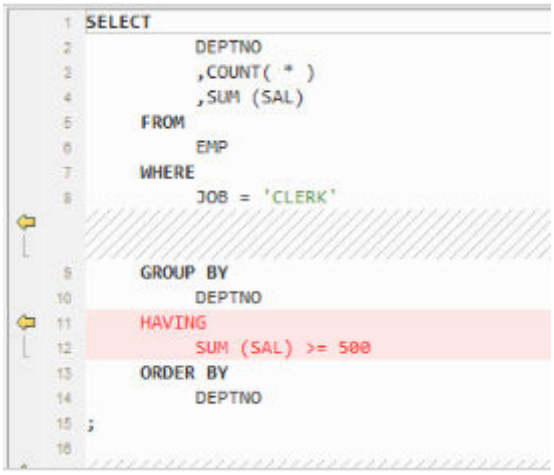


Figure 6-15 Output - Order of Clauses



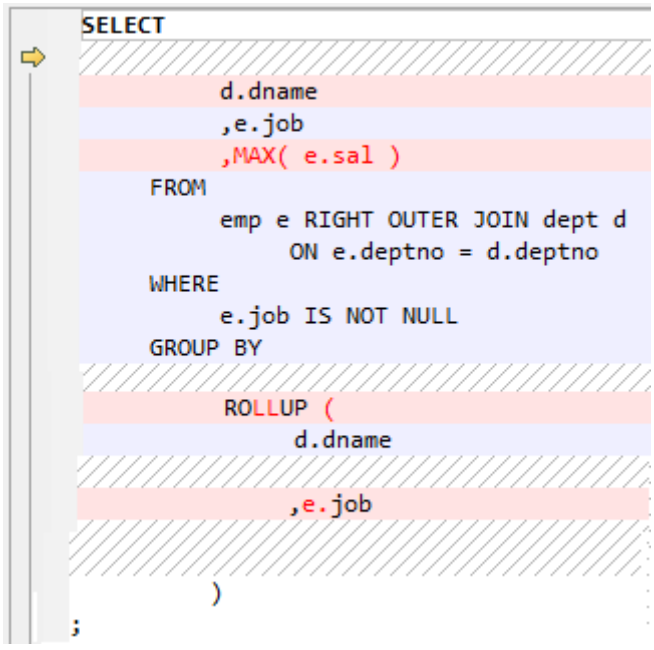
```
SELECT
        DEPTNO
        ,COUNT( * )
        ,SUM (SAL)
    FROM
        EMP
    WHERE
        JOB = 'CLERK'

    GROUP BY
        DEPTNO
    HAVING
        SUM (SAL) >= 500
    ORDER BY
        DEPTNO
;
```

2. **Extended Group By Clause**

   The **GROUP BY** clause can be specified if you want the database to group the selected rows based on the value of expr(s). If this clause contains **CUBE**, **ROLLUP**, or **GROUPING SETS** extensions, then the database produces super-aggregate groupings in addition to the regular groupings. These features are not supported by GaussDB(DWS) but can be enabled using the **UNION ALL** operator.

Figure 6-16 Input - Extended group by clause



```
SELECT
        d.dname
        ,e.job
        ,MAX( e.sal )
    FROM
        emp e RIGHT OUTER JOIN dept d
            ON e.deptno = d.deptno
    WHERE
        e.job IS NOT NULL
    GROUP BY
        ROLLUP (
            d.dname
            ,e.job
        )
;
```


Figure 6-17 Output - Extended group by clause

```
SELECT
        dname
        ,job
        ,ColumnAlias1
    FROM
        (
            SELECT
                    MAX( e.sal ) AS ColumnAlias1
                    ,d.dname
                    ,e.job
                FROM
                    emp e RIGHT OUTER JOIN dept d
                        ON e.deptno = d.deptno
                WHERE
                    e.job IS NOT NULL
                GROUP BY
                    d.dname ,e.job
            UNION
            ALL SELECT
                    MAX( e.sal ) AS ColumnAlias1
                    ,d.dname
                    ,NULL AS job
                FROM
                    emp e RIGHT OUTER JOIN dept d
                        ON e.deptno = d.deptno
                WHERE
                    e.job IS NOT NULL
                GROUP BY
                    d.dname
            UNION
            ALL SELECT
                    MAX( e.sal ) AS ColumnAlias1
                    ,NULL AS dname
                    ,NULL AS job
                FROM
                    emp e RIGHT OUTER JOIN dept d
                        ON e.deptno = d.deptno
                WHERE
                    e.job IS NOT NULL
        )
;
```

GROUPING_ID and ROLLUP

GROUPING_ID returns a number that corresponds to the **GROUPING** bit vector associated with a row. **GROUPING_ID** is applicable only in a **SELECT** statement containing a **GROUP BY** extension, such as the **ROLLUP** operator and **GROUPING** function. In queries with multiple **GROUP BY** expressions, determining the **GROUP BY** level of a particular row requires multiple **GROUPING** functions, which may complicate SQL statements. In such scenarios, **GROUPING_ID** is used to avoid statement complexity.

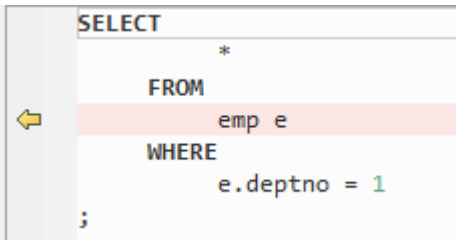
3. **Table Name Inside Brackets**

   Table names do not need to be specified within parentheses. However, allows using brackets.

Figure 6-18 Input - Table name inside brackets

```
SELECT
        *
    FROM
        (emp) e
    WHERE
        e.deptno = 1
;
```


Figure 6-19 Output - Table name inside brackets



4. **UNIQUE Keyword**

   Unique keyword is migrated as Distinct keyword.

   Input - SELECT UNIQUE

```
SELECT UNIQUE a.item_id id,
                a.menu_id parent_id,a.serialno menu_order
            FROM ctp_menu_item_rel a WHERE
  a.item_id IN(SELECT UNIQUE id FROM ctp_temp_item_table);
```

   Output

```
SELECT DISTINCT a.item_id id,
  a.menu_id parent_id,a.serialno menu_order
    FROM ctp_menu_item_rel a WHERE
     a.item_id IN(SELECT UNIQUE id FROM ctp_temp_item_table);
```

5. **USERENY**

   Input - CLIENT_INFO

   Returns user session information.

```
SELECT 1
FROM   sp_ht ht
WHERE  ht.hth = pi_contract_number
      /*  AND ht.contract_status = 2  --delete by leinian 2014-03-03(ECO)  */
      AND ht.contract_status IN ( 1, 2 ) /*  add by leinian 2014-03-20(ECO)  */
      AND Nvl(ht.s3_pilot_flag, 'N') = 'N'
      AND NOT EXISTS (SELECT 1
                      FROM   asms.asms_lookup_values alv
                      WHERE  alv.type_code = 'HTLX_LOAN'
                             AND ht.htlx = alv.code)
      AND ht.duty_erp_ou_id = To_number(Nvl(Rtrim(Ltrim(Substr(Userenv(
                                                'client_info'),
                                                1,
                                                8))), 218))
      AND ht.source_code = 'ECONTRACT'
      AND ht.needing_engineering_service IS NOT NULL
      AND ht.khm != '28060'
      AND ht.htlx != '111' ;
```

   Output

```
SELECT
        1
    FROM
        sp_ht ht
    WHERE
        ht.hth = pi_contract_number /*   AND ht.contract_status = 2  --delete by leinian
2014-03-03(ECO)   */
        AND ht.contract_status IN (
            1
            ,2
        ) /*   add by leinian 2014-03-20(ECO)   */
        AND Nvl( ht.s3_pilot_flag ,'N' ) = 'N'
        AND NOT EXISTS (
            SELECT
                    1
                FROM
                    asms.asms_lookup_values alv
```



```
                WHERE
                    alv.type_code = 'HTLX_LOAN'
                    AND ht.htlx = alv.code
        )
        AND ht.duty_erp_ou_id = To_number( Nvl( Rtrim( Ltrim( SUBSTR( MIG_ORA_EXT.USERENV
( 'client_info' ) ,1 ,8 ) ) ) ,218 ) )
        AND ht.source_code = 'ECONTRACT'
        AND ht.needing_engineering_service IS NOT NULL
        AND ht.khm != '28060'
        AND ht.htlx != '111' ;
```

USERENV('CLIENT_INFO)

After the function in the package is converted, the function tag is not deleted. The **svproduct_is_for_pa** function in **4_sad_lookup_contract_pkg.bdy** is used.

USERENV('CLIENT_INFO')

USERENV used during the migration process. Migration fails due to the tool.

```
SELECT 1
FROM   sp_ht ht
WHERE  ht.hth = pi_contract_number
/*  AND ht.contract_status = 2  --delete by leinian 2014-03-03(ECO)  */
AND ht.contract_status IN ( 1, 2 ) /*  add by leinian 2014-03-20(ECO)  */
AND Nvl(ht.s3_pilot_flag, 'N') = 'N'
/*  add by yangyirui 2012-09-10: S3 Data is not provided for the contract cutover.  */
AND NOT EXISTS (SELECT 1
FROM   asms.asms_lookup_values alv
WHERE  alv.type_code = 'HTLX_LOAN'
AND ht.htlx = alv.code)
AND ht.duty_erp_ou_id = To_number(Nvl(Rtrim(Ltrim(Substr(Userenv(
'client_info'),
1,
8))), 218))
AND ht.source_code = 'ECONTRACT'
AND ht.needing_engineering_service IS NOT NULL
AND ht.khm != '28060'
AND ht.htlx != '111'
```

Input

```
Error message :client_info argument for USERENV function is not supported by the DSC.

4_sad_lookup_contract_pkg

================================
CREATE OR REPLACE PACKAGE BODY SAD.sad_lookup_contract_pkg IS
 FUNCTION svproduct_is_for_pa(pi_contract_number IN VARCHAR2) RETURN VARCHAR2 IS
   l_flag VARCHAR2(1) := 'N';
  BEGIN
   FOR rec_lookup IN (SELECT 1
     FROM asms.asms_lookup_values alv
    WHERE alv.type_code = 'HTLX_LOAN'
      AND alv.duty_erp_ou_id = to_number(nvl(rtrim(ltrim(substr(userenv('client_info'), 1, 8))), 218))
                  )
   LOOP
    l_flag := 'Y';
   END LOOP;

   RETURN l_flag;
  END svproduct_is_for_pa;
END sad_lookup_contract_pkg;
/
```

Output

```
CREATE OR replace FUNCTION sad_lookup_contract_pkg.Svproduct_is_for_pa (
pi_contract_number IN VARCHAR2)
RETURN VARCHAR2
IS
  l_flag VARCHAR2 ( 1 ) := 'N';
BEGIN
```



```
    FOR rec_lookup IN (SELECT 1
                       FROM   asms.asms_lookup_values alv
                       WHERE  alv.type_code = 'HTLX_LOAN'
                              AND alv.duty_erp_ou_id = To_number(Nvl(
                                                       Rtrim(Ltrim(Substr(
                                  mig_ora_ext.Userenv (
                                  'client_info'), 1, 8))
                                                       ),
                                                                  218)
                                                       ))
    LOOP
        l_flag := 'Y';
    END LOOP;

    RETURN l_flag;
END;

/
```

INSERT

Overview

The Oracle **INSERT** statement is used to insert a single record or multiple records into a table.

NOLOGGING

NOLOGGING is commented from the inserted script.

Oracle Syntax	Syntax After Migration
INSERT INTO TBL_ORACLE NOLOGGING SELECT emp_id, emp_name FROM emp;	INSERT INTO TBL_ORACLE /*NOLOGGING*/ SELECT emp_id, emp_name FROM emp;

1. **INSERT ALL**

   The Oracle **INSERT ALL** statement is used to add multiple rows using a single **INSERT** statement. The rows can be inserted into either a single table or multiple tables. The target query is converted as a common table expression (CTE).

Figure 6-20 Input - INSERT ALL

```
INSERT
     ALL INTO
          ap_cust
     VALUES (
          customer_id
          ,program_id
          ,delivered_date
     ) INTO
          ap_orders (
               ord_dt
               ,Prg_id
          )
     VALUES (
          order_date
          ,program_id
     ) SELECT
               program_id
               ,delivered_date
               ,customer_id
               ,order_date
          FROM
               ORDER
          WHERE
               deptno = 10
```


Figure 6-21 Output - Insert All

```
WITH Sel AS (
     SELECT
               program_id
               ,delivered_date
               ,customer_id
               ,order_date
          FROM
               ORDER
          WHERE
               deptno = 10
)
,ins1 AS (
     INSERT
          INTO
               ap_cust (
                    SELECT
                              customer_id
                              ,program_id
                              ,delivered_date
                         FROM
                              Sel
               ) returning *
)
INSERT
     INTO
          ap_orders (
               ord_dt
               ,Prg_id
          ) (
     ) (
          SELECT
                    order_date
                    ,program_id
               FROM
                    Sel
     )
```

2. **INSERT FIRST**

   The Oracle **INSERT FIRST** is used to execute an INSERT statement when the first condition is true; other statements are ignored. The target query is converted as a CTE.

Figure 6-22 Input - Insert first

```
INSERT
      FIRST WHEN deptno <= 10
      THEN INTO
            emp12 WHEN comm > 500
            THEN INTO
                  emp13 SELECT

                                    empno
                                    ,ename
                                    ,job
                                    ,mgr
                                    ,hiredate
                                    ,sal
                                    ,comm
                                    ,deptno
                              FROM
                                    emp
                              WHERE
                                    deptno IS NOT NULL
```

Figure 6-23 Output - Insert first

```
WITH Sel AS (
      SELECT
                  ROW_NUMBER( ) OVER( ) AS Ins_First_RN
                  ,empno
                  ,ename
                  ,job
                  ,mgr
                  ,hiredate
                  ,sal
                  ,comm
                  ,deptno
            FROM
                  emp
            WHERE
                  deptno IS NOT NULL
)
,ins1 AS (
     INSERT
          INTO
               emp12 (
                    SELECT
                              empno
                              ,ename
                              ,job
                              ,mgr
                              ,hiredate
                              ,sal
                              ,comm
                              ,deptno
                         FROM
                              Sel
                         WHERE
                              deptno <= 10
               ) returning 1
)
INSERT
      INTO
           emp13 (
                SELECT
                         empno
                         ,ename
                         ,job
                         ,mgr
                         ,hiredate
                         ,sal
                         ,comm
                         ,deptno
                    FROM
                         (
                              SELECT
                                   *
                                   FROM
                                        Sel
                                   WHERE
                                        comm > 500
                         ) s1 LEFT JOIN (
                        SELECT
                              Ins_First_RN
                          FROM
                              Sel
                          WHERE
                               deptno <= 10
                    ) s2
                         ON s1.Ins_First_RN = s2.Ins_First_RN
               WHERE
                    s2.Ins_First_RN IS NULL
)
```


3. **INSERT with Table Alias**

   The Oracle **table aliases** are used to clarify and improve readability when referring to a table in a query by assigning it a name or code. **INSERT with table alias** can be used with **INSERT INTO** statement. The tool supports the migration of **INSERT INTO** statements with **table alias**.

   a. **Blogic Operations**

   Input - INSERT with Table Alias

```
CREATE
    OR REPLACE FUNCTION myfct RETURN VARCHAR2 IS res VARCHAR2 ( 200 ) ;
    BEGIN
        res := 100 ;
        INSERT INTO emp18 RW ( RW.empno ,RW.ename ) SELECT
            res ,RWN.ename
        FROM
            emp16 RWN ;
            COMMIT ;
        RETURN res ;
END ;
/
```

   Output

```
CREATE
    OR REPLACE FUNCTION myfct RETURN VARCHAR2 IS res VARCHAR2 ( 200 ) ;
    BEGIN
        res := 100 ;
        INSERT INTO emp18 ( empno ,ename ) SELECT
            res ,RWN.ename
        FROM
            emp16 RWN ;
            /* COMMIT ; */
        null ;
        RETURN res ;
END ;
/
```

   b. **Bulk Operations**

   Input - INSERT with Table Alias

```
INSERT
    INTO
        Public.emp14 ats (
            ats.empno
            ,ats.ename
        )
    VALUES (
        3
        ,'Categories'
    )
;
```

   Output

```
INSERT
    INTO
        Public.emp14 (
            empno
            ,ename
        ) SELECT
                3
                ,'Categories'
;
```

   Input - INSERT with Table Alias

```
INSERT
    INTO
        "abc" . "emp18" wmc (
            wmc.empno
```



```
        ,wmc.ename
    ) SELECT
            wmc.empno
            ,wm_concat (wmc.ename) AS eName
        FROM
            emp16 wmc
        GROUP BY
            empno
;
```

Output

```
INSERT
    INTO
        "abc" . "emp18" (
            empno
            ,ename
        ) SELECT
                wmc.empno
                ,STRING_AGG (
                    wmc.ename
                    ,','
                ) AS eName
            FROM
                emp16 wmc
            GROUP BY
                empno
;
```

Input - INSERT with Table Alias

```
INSERT
    INTO
        emp14 "TABLE" (
            "TABLE" .empno
            ,ename
        ) SELECT
                empno
                ,ename
            FROM
                emp12
            WHERE
                emp12.salary > (
                    SELECT
                            MAX( salary )
                        FROM
                            emp13 "TABLE"
                        WHERE
                            "TABLE" .empno > 5
                )
;
```

Output

```
INSERT
    INTO
        emp14 (
            empno
            ,ename
        ) SELECT
                empno
                ,ename
            FROM
                emp12
            WHERE
                emp12.salary > (
                    SELECT
                            MAX( salary )
                        FROM
                            emp13 "TABLE"
                        WHERE
                            "TABLE" .empno > 5
```



```
            )
;
```

MERGE

MERGE is an ANSI-compliant SQL syntax operator used to select rows from one or more sources for updating or inserting a table or view. The criteria for updating or inserting the target table or view can be specified.

DSC uses multiple methods to migrate **MERGE** to SQL statements compatible with GaussDB(DWS).

Configure parameter **mergeImplementation** as follows:

- Set to **With** by default. In this option, the target query is converted as a CTE.

Figure 6-24 Input - MERGE

```
MERGE INTO
     student a
          USING (
          SELECT
                    id
                    ,sname
                    ,score
               FROM
                    student_n
     ) b
          ON( a.id = b.id ) WHEN MATCHED
     THEN UPDATE

          SET
               a.sname = b.sname
               ,a.score = b.score DELETE


                    WHERE


                              a.score < 640

;
```


Figure 6-25 Output - MERGE

```
WITH b AS (
      SELECT
                  id
                  ,sname
                  ,score
            FROM
                  student_n
)
,UPD_REC AS (
      UPDATE
                  student a
            SET
                  a.sname = b.sname
                  ,a.score = b.score
            FROM
                  b
            WHERE
                  a.id = b.id returning a. *
) DELETE FROM student a
      USING b
      WHERE
            a.score < 640
            AND a.id = b.id
;
```

- Set to **SPLIT**. In this option, the **MERGE** statement is split into multiple **INSERT** and **UPDATE** statements.

Figure 6-26 Input - MERGE

```
MERGE INTO employees01 e

USING (SELECT empid, ename, startdate, address
          FROM hr_records
       WHERE empid > 100) h

ON (e.id = h.empid)
WHEN MATCHED THEN
UPDATE SET e.address = h.address
        , e.ename = h.ename
WHEN NOT MATCHED THEN
INSERT (empid,ename,startdate,address)
VALUES (h.empid,h.ename,h.startdate,h.address);
```


Figure 6-27 Output - MERGE

```
UPDATE  employees01 e
    SET e.address = h.address
        , e.ename = h.ename
  FROM ( SELECT empid, ename, startdate, address
              FROM  hr_records
             WHERE  empid > 100
        ) h
 WHERE e.id = h.empid;


INSERT INTO employees01 ( empid, ename, startdate, address )
SELECT h.empid, h.ename, h.startdate, h.address
 FROM ( SELECT empid, ename, startdate, address
             FROM  hr_records
            WHERE  empid > 100
        ) h LEFT OUTER JOIN employees01 e
     ON e.id = h.empid
   WHERE e.id IS NULL;
```

6.6.10 Pseudo Columns

This section contains the migration syntax of Oracle Pseudo Columns. The migration syntax decides how the keywords/features are migrated.

A **pseudo column** is similar to a table column, but is not actually stored in the table. User can select values from pseudo columns, but cannot insert, update, or delete values in the pseudo columns.

ROWID

The **ROWID** pseudo column returns the address of a specific row.

Figure 6-28 Input - ROWID

```
SELECT
            empid
            ,ename
            ,ROWID
      FROM
            employees
;
```

Figure 6-29 Output - ROWID

```
SELECT
          empid
          ,ename
          ,CAST( ( xc_node_id || '#' || tableoid || '#' || ctid ) AS TEXT ) AS rowid
     FROM
          employees
;
```


ROWNUM

For each row of data returned by the query, the **ROWNUM** pseudo column returns a number, indicating the order with which the Oracle database selects rows from a table or a group of joined rows. The value of **ROWNUM** in the first row is **1**, the value of **ROWNUM** in the second row is **2**, and so on.

Figure 6-30 Input - ROWNUM

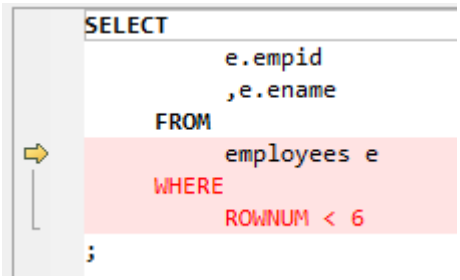
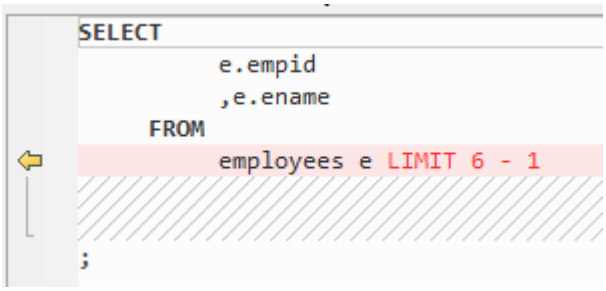


Figure 6-31 Output - ROWNUM



Input-ROWNUM with UPDATE

When executing **UPDATE**, if ROWNUM with some value (integer) is used, the system will UPDATE records using the operator near ROWNUM accordingly.

```
UPDATE SCMS_MSGPOOL_LST
          SET MSG_STD = '11'
        WHERE UNISEQNO = IN_OUNISEQNO
          AND MSG_TYP1 IN ('MT103', 'MT199')
          AND ROWNUM = 1;
```

Output

```
UPDATE SCMS_MSGPOOL_LST
  SET MSG_STD = '11'
WHERE (xc_node_id,ctid) in (select xc_node_id, ctid
      from SCMS_MSGPOOL_LST
      where UNISEQNO = IN_OUNISEQNO
      AND MSG_TYP1 IN ('MT103', 'MT199')
      LIMIT 1)
```

Input-ROWNUM with DELETE

When executing DELETED, if ROWNUM with some value (integer) is used, system will DELETE records using the operator near ROWNUM accordingly.

```
delete from test1
where c1='abc' and rownum = 1;
```

Output


```
delete from test1 where (xc_node_id,ctid) in (select xc_node_id, ctid from test1 where c1='abc' limit 1);
```

Input - UPDATE with ROWNUM

The UPDATE and DELETE scripts migrated using **ROWNUM** contain **LIMIT**, which is not supported by GaussDB(DWS).

```
UPDATE SCMS_MSGPOOL_LST
 SET MSG_STD = '11'
 WHERE UNISEQNO = IN_OUNISEQNO
 AND MSG_TYP1 IN ('MT103', 'MT199')
 AND ROWNUM = 1;
```

Output

```
UPDATE SCMS_MSGPOOL_LST
  SET MSG_STD = '11'
WHERE (xc_node_id, ctid) = ( SELECT xc_node_id, ctid
         FROM SCMS_MSGPOOL_LST
        WHERE UNISEQNO = IN_OUNISEQNO
       AND MSG_TYP1 IN ('MT103', 'MT199')
        LIMIT 1
     );
```

Input - DELETE with ROWNUM

```
DELETE FROM SPMS_APP_PUBLISH
 WHERE NOVA_NO = IN_NOVA_NO
  AND DELIVERY_TYPE = '1'
  AND PUBLISH_DATE = IN_PUBLISH_DATE
  AND ROWNUM = 1;
```

Output

```
DELETE FROM SPMS_APP_PUBLISH
 WHERE (xc_node_id, ctid) IN (SELECT xc_node_id, ctid
         FROM SPMS_APP_PUBLISH
        WHERE NOVA_NO = IN_NOVA_NO
       AND DELIVERY_TYPE = '1'
       AND PUBLISH_DATE = IN_PUBLISH_DATE
        LIMIT 1
      );
```

6.6.11 OUTER JOIN

This section describes the migration syntax of Oracle **OUTER JOIN**. The migration syntax determines how the keywords/features are migrated.

An **OUTER JOIN** returns all rows that meet the join condition. If rows of a table cannot join any rows in the other table, the statement returns these rows. In Oracle:

- Left outer join of tables A and B returns all rows from A and rows that satisfy the join condition by applying the outer join operator (+) to all columns of B in the **WHERE** conditions.
- Right outer join of tables A and B returns all rows from B and rows that satisfy the join condition by applying the outer join operator (+) to all columns of A in the **WHERE** condition.

GaussDB(DWS) does not support the **+** operator. The function of this operator is enabled using **LEFT OUTER JOIN** and **RIGHT OUTER JOIN** keywords.

Figure 6-32 Input: OUTER JOIN

```
SELECT
        empno
        ,ename
        ,job
        ,dname
        ,loc
    FROM
        emp
        ,dept
    WHERE
        emp.deptno = dept.deptno (+)
        AND salary > 50000
;
```

Figure 6-33 Output: OUTER JOIN

```
SELECT
        empno
        ,ename
        ,job
        ,dname
        ,loc
    FROM
        emp LEFT OUTER JOIN dept
            ON emp.deptno = dept.deptno
    WHERE

        salary > 50000
```

6.6.12 OUTER QUERY (+)

Join is supported by GaussDB(DWS), so parameter **supportJoinOperator** is added.

OUTER QUERY (+) can be migrated when **supportJoinOperator** is set to **false**.

Input-OUTER QUERY(+)

```
SELECT PP.PUBLISH_NO
                FROM SPMS_PARAM_PUBLISH PP
                  WHERE PP.PUBLISH_ID(+) = TB2.PUBLISH_ID;

            SELECT  I.APP_CHNAME, I.APP_SHORTNAME
                FROM SPMS_APPVERSION SA, SPMS_APP_INFO I
                 WHERE SA.APP_ID = I.APP_ID(+)
                  AND SA.DELIVERY_USER = IN_USERID
                  ORDER BY  APPVER_ID DESC ;
```

Output

```
SELECT
        PP.PUBLISH_NO
    FROM
        SPMS_PARAM_PUBLISH PP
    WHERE
        PP.PUBLISH_ID (+) = TB2.PUBLISH_ID
;
```



```
SELECT
        I.APP_CHNAME
        ,I.APP_SHORTNAME
    FROM
        SPMS_APPVERSION SA
        ,SPMS_APP_INFO I
    WHERE
        SA.APP_ID = I.APP_ID (+)
        AND SA.DELIVERY_USER = IN_USERID
    ORDER BY
        APPVER_ID DESC
;
```

6.6.13 CONNECT BY

Input-CONNECT BY

```
SELECT id FROM city_branch start with id=roleBranchId connect by prior id=parent_id;
SELECT T.BRANCH_LEVEL, t.ID
                FROM city_branch c
               WHERE      (c.branch_level = '1' OR T.BRANCH_LEVEL = '2')
                    AND (T.SIGN = '1' OR T.SIGN = '4' OR T.SIGN = '8')
                    AND T.STATUS = '1'
          START WITH c.ID = I_BRANCH_ID
          CONNECT BY c.ID = PRIOR c.parent_id
           ORDER BY c.branch_level DESC ;
```

Output

```
WITH RECURSIVE migora_cte AS (
    SELECT
            id
            ,1 AS LEVEL
        FROM
            city_branch
        WHERE
            id = roleBranchId
    UNION
    ALL SELECT
           mig_ora_cte_join_alias.id
           ,mig_ora_cte_tab_alias.LEVEL + 1 AS LEVEL
        FROM
           migora_cte mig_ora_cte_tab_alias INNER JOIN city_branch mig_ora_cte_join_alias
              ON mig_ora_cte_tab_alias.id = mig_ora_cte_join_alias.parent_id
) SELECT
        id
    FROM
        migora_cte
    ORDER BY
        LEVEL
;

WITH RECURSIVE migora_cte AS (
    SELECT
            BRANCH_LEVEL
            ,ID
            ,SIGN
            ,STATUS
            ,parent_id
            ,1 AS LEVEL
        FROM
            city_branch c
        WHERE
            c.ID = I_BRANCH_ID
    UNION
    ALL SELECT
            c.BRANCH_LEVEL
            ,c.ID
            ,c.SIGN
```



```
            ,c.STATUS
            ,c.parent_id
            ,mig_ora_cte_tab_alias.LEVEL + 1 AS LEVEL
        FROM
            migora_cte mig_ora_cte_tab_alias INNER JOIN city_branch c
                ON c.ID = mig_ora_cte_tab_alias.parent_id
) SELECT
        BRANCH_LEVEL
        ,ID
    FROM
        migora_cte c
    WHERE
        (
            c.branch_level = '1'
            OR T.BRANCH_LEVEL = '2'
        )
        AND( T.SIGN = '1' OR T.SIGN = '4' OR T.SIGN = '8' )
        AND T.STATUS = '1'
    ORDER BY
        c.branch_level DESC
;
```

Input - CONNECT BY multiple tables

The syntax shows the relationship between each child row and its parent row. It uses the **CONNECT BY** *xxx* **PRIOR** clause to define the relationship between the current row (child row) and the previous row (parent row).

```
SELECT DISTINCT a.id menuId,
             F.name menuName,
             a.status menuState,
             a.parent_id menuParentId,
             '-1' menuPrivilege,
             a.serialNo menuSerialNo
  FROM CTP_MENU a, CTP_MENU_NLS F
  START WITH a.serialno in (1, 2, 3)
CONNECT BY a.id = PRIOR a.parent_id
       AND f.locale = Language
       AND a.id = f.id
ORDER BY menuId, menuParentId;
```

Output

```
WITH RECURSIVE migora_cte AS (
       SELECT pr.service_product_id
           , t.enabled_flag
           , pr.operation_id
           , pr.enabled_flag
           , pr.product_code
           , 1 AS LEVEL
         FROM asms.cppsv_operation_sort t
           , asms.cppsv_product_class pr
        WHERE level_id = 3
          AND pr.operation_id = t.operation_id(+)
     UNION ALL
       SELECT pr.service_product_id
            , t.enabled_flag
            , pr.operation_id
            , pr.enabled_flag
            , pr.product_code
            , mig_ora_cte_tab_alias.LEVEL + 1 AS LEVEL
         FROM migora_cte mig_ora_cte_tab_alias
           , asms.cppsv_operation_sort t
           , asms.cppsv_product_class pr
        WHERE mig_ora_cte_tab_alias.service_product_id = pr.service_product_father_id
          AND pr.operation_id = t.operation_id(+) )
SELECT pr.service_product_id
  FROM migora_cte
 WHERE nvl( UPPER( enabled_flag ) ,'Y' ) = 'Y'
```



```
  AND nvl( enabled_flag ,'Y' ) = 'Y'
  AND pr.product_code = rec_product1.service_product_code
ORDER BY LEVEL;
```

6.6.14 System Functions

This section describes the migration syntax of Oracle system functions. The migration syntax determines how the keywords and features are migrated.

The system functions include:

Date functions, LOB functions, string functions, analytical functions, and regular expression functions. For details, see **Date Functions** to **Regular Expression Functions**.

6.6.14.1 Date Functions

This section describes the following date functions:

- **ADD_MONTHS**
- **DATE_TRUNC**
- **LAST_DAY**
- **MONTHS_BETWEEN**
- **SYSTIMESTAMP**

ADD_MONTHS

ADD_MONTHS is an Oracle system function and is not implicitly supported by GaussDB(DWS).

NOTE

Before using this function, perform the following operations:

1. Create and use the **MIG_ORA_EXT** schema.
2. Copy the content of the custom script and execute the script in all target databases for which migration is to be performed. For details, see **Migration Process**.

ADD_MONTHS returns a date with the month.

- Data type of the **date** parameter is DATETIME.
- Data type of the **integer** parameter is INTEGER.

The return type is DATE.

Input - ADD_MONTHS

```
SELECT
        TO_CHAR( ADD_MONTHS ( hire_date ,1 ) ,'DD-MON-YYYY' ) "Next month"
    FROM
        employees
    WHERE
        last_name = 'Baer'
;
```

Output

```
SELECT
        TO_CHAR( MIG_ORA_EXT.ADD_MONTHS ( hire_date ,1 ) ,'DD-MON-YYYY' ) "Next month"
    FROM
        employees
```



```
    WHERE
        last_name = 'Baer'
;
```

TO_DATE with Third Parameter

In TO_DATE(' 2019-05-02 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN'), the third parameter should be commented.

Input

```
CREATE TABLE PRODUCT
    ( prod_id       INTEGER
    , prod_code     VARCHAR(5)
    , prod_name     VARCHAR(100)
    , unit_price    NUMERIC(6,2) NOT NULL
    , manufacture_date  DATE DEFAULT sysdate )
 PARTITION BY RANGE (manufacture_date)
 (PARTITION "P_20190501"  VALUES LESS THAN (TO_DATE(' 2019-05-02 00:00:00', 'SYYYY-MM-DD
HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN'))
  );
```

Output

```
CREATE TABLE PRODUCT
    ( prod_id       INTEGER
    , prod_code     VARCHAR(5)
    , prod_name     VARCHAR(100)
    , unit_price    NUMERIC(6,2) NOT NULL
    , manufacture_date  DATE DEFAULT sysdate )
 PARTITION BY RANGE (manufacture_date)
 (PARTITION "P_20190501"  VALUES LESS THAN (TO_DATE(' 2019-05-02 00:00:00', 'YYYY-MM-DD
HH24:MI:SS'/* , 'NLS_CALENDAR=GREGORIAN' */))
  );
```

DATE_TRUNC

The DATE_TRUNC function returns a date with the time portion of the day truncated to the unit specified by the format model **fmt**.

Input

```
select trunc(to_char(trunc(add_months(sysdate,-12),'MM'),'YYYYMMDD')/100) into v_start_date_s from
dual;
select trunc(to_char(trunc(sysdate,'mm'),'YYYYMMDD')/100) into v_end_date_e from dual;
ID_MNTH>=TRUNC(TO_CHAR(ADD_MONTHS(to_date(to_char('||
v_curr_date||'),''YYYYMMDD''),-12),''YYYYMMDD'')/100))
AND ID_MNTH>=TRUNC(TO_CHAR(ADD_MONTHS(to_date(to_char('||
v_curr_date||'),''YYYYMMDD''),-12),''YYYYMMDD'')/100))

select
TRUNC(to_char(add_months(trunc(TO_DATE(TO_CHAR(P_DATE),'YYYYMMDD'),'MM')-1,-2),'YYYYMMDD')/
100) INTO START_MONTH from dual;
select TRUNC(TO_CHAR(trunc(TO_DATE(TO_CHAR(P_DATE),'YYYYMMDD'),'MM')-1,'YYYYMMDD')/100)
INTO END_MONTH from dual;
```

Output

```
SELECT Trunc(To_char(Date_trunc ('MONTH', mig_ora_ext.Add_months (SYSDATE, -12)) , 'YYYYMMDD') /
100)
INTO   v_start_date_s
FROM   dual;

SELECT Trunc(To_char(Date_trunc ('MONTH', SYSDATE), 'YYYYMMDD') / 100)
INTO   v_end_date_e
FROM   dual;
```



```
SELECT Trunc(To_char(mig_ora_ext.Add_months (Date_trunc ('MONTH', To_date(To_char(p_date), 'YYYYMMDD' )) - 1 , -2), 'YYYYMMDD') / 100)
INTO   start_month
FROM   dual;

SELECT Trunc(To_char(Date_trunc ('MONTH', To_date(To_char(p_date), 'YYYYMMDD')) - 1, 'YYYYMMDD') / 100)
INTO   end_month
FROM   dual;
```

LAST_DAY

The Oracle LAST_DAY function returns the last day of the month based on a date value.

```
LAST_DAY(date)
```

The return type is always DATE, regardless of the data type of the date.

LAST_DAY is an Oracle system function and is not implicitly supported by GaussDB(DWS). To support this function, DSC creates a LAST_DAY function in the **MIG_ORA_EXT** schema. The migrated statements will use the new function MIG_ORA_EXT.LAST_DAY as shown in the following example.

NOTE

Before using this function, perform the following operations:

1. Create and use the **MIG_ORA_EXT** schema.
2. Copy the content of the custom script and execute the script in all target databases for which migration is to be performed. For details, see **Migration Process**.

Input - LAST_DAY

```
SELECT
        to_date( '01/' || '07/' || to_char( sysdate ,'YYYY' ) ,'dd/mm/yyyy' ) FIRST
        ,last_day( to_date( '01/' || '07/' || to_char( sysdate ,'YYYY' ) ,'dd/mm/yyyy' ) ) last__day
  FROM
        dual;
```

Output

```
SELECT
        to_date( '01/' || '07/' || to_char( sysdate ,'YYYY' ) ,'dd/mm/yyyy' ) FIRST
        ,MIG_ORA_EXT.LAST_DAY (
            to_date( '01/' || '07/' || to_char( sysdate ,'YYYY' ) ,'dd/mm/yyyy' )
        ) last__day
  FROM
        dual;
```

MONTHS_BETWEEN

The MONTHS_BETWEEN function returns the number of months between two dates.

MONTHS_BETWEEN is an Oracle system function and is not implicitly supported by GaussDB(DWS). To support this function, use DSC to create a MONTHS_BETWEEN function in the **MIG_ORA_EXT** schema. The migrated statements will use the new function MIG_ORA_EXT.MONTHS_BETWEEN as shown in the following example.

NOTE

Before using this function, perform the following operations:

1. Create and use the **MIG_ORA_EXT** schema.
2. Copy the content of the custom script and execute the script in all the target databases to which data is to be migrated. For details, see **Migration Process**.

Input - MONTHS_BETWEEN

```
Select Months_Between(to_date('2017-06-20', 'YYYY-MM-DD'), to_date('2011-06-20', 'YYYY-MM-DD')) from dual;
```

Output

```
Select MIG_ORA_EXT.MONTHS_BETWEEN(to_date('2017-06-20', 'YYYY-MM-DD'), to_date('2011-06-20', 'YYYY-MM-DD')) from dual;
```

SYSTIMESTAMP

The SYSTIMESTAMP function returns the system date, including fractional seconds and time zones, of the system on which the database resides. The return type is **TIMESTAMP WITH TIME ZONE**.

Figure 6-34 Input - SYSTIMESTAMP

```
SELECT
        SYSTIMESTAMP
    FROM
        tab1
;
```

Figure 6-35 Output - SYSTIMESTAMP

```
SELECT
        CURRENT_TIMESTAMP
    FROM
        tab1
;
```

6.6.14.2 LOB Functions

This section describes the following LOB functions:

- **DBMS_LOB.APPEND**
- **DBMS_LOB.COMPARE**
- **DBMS_LOB.CREATETEMPORARY**
- **DBMS_LOB.INSTR**
- **DBMS_LOB.SUBSTR**

DBMS_LOB.APPEND

DBMS_LOB.APPEND function appends the content of a source LOB to a specified LOB.

Input - DBMS_LOB.APPEND

```
[sys.]dbms_lob.append(o_menuxml, to_clob('DSJKSDAJKSFDA'));
```

Output

```
o_menuxml := CONCAT(o_menuxml, CAST('DSJKSDAJKSFDA' AS CLOB));
```

Input - DBMS_LOB.APPEND

```
CREATE
    OR REPLACE PROCEDURE append_example IS clobSrc CLOB ;
    clobDest CLOB ;
BEGIN
    SELECT
            clobData INTO clobSrc
        FROM
            myTable
        WHERE
            id = 2 ;
           SELECT
                clobData INTO clobDest
            FROM
                myTable
            WHERE
                id = 1 ;
                readClob ( 1 ) ;
            DBMS_LOB.APPEND ( clobDest ,clobSrc ) ;
        readClob ( 1 ) ;
END append_example ;
/
```

Output

```
CREATE
    OR REPLACE PROCEDURE append_example IS clobSrc CLOB ;
    clobDest CLOB ;
BEGIN
    SELECT
            clobData INTO clobSrc
       FROM
            myTable
        WHERE
            id = 2 ;
            SELECT
                clobData INTO clobDest
            FROM
                myTable
            WHERE
                id = 1 ;
                readClob ( 1 ) ;
            clobDest := CONCAT( clobDest ,clobSrc ) ;
        readClob ( 1 ) ;
end ;
/
```

DBMS_LOB.COMPARE

DBMS_LOB.COMPARE is an Oracle system function and is not implicitly supported by GaussDB(DWS).

This function is used to compare the full/partial content of two LOBs. To support this feature, use DSC to create a **COMPARE** function in the **MIG_ORA_EXT** schema. The migrated statements will use the new function **MIG_ORA_EXT.MIG_CLOB_COMPARE**, and the examples of using functions in SQL statements are shown as follows.

COMPARE in SQL

Input - DBMS_LOB.COMPARE in SQL

```
SELECT a.empno ,dbms_lob.compare ( col1 ,col2 ) FROM emp a ,emp b ;
```

Output

```
SELECT a.empno ,MIG_ORA_EXT.MIG_CLOB_COMPARE ( col1 ,col2 ) FROM emp a ,emp b ;
```

Input - DBMS_LOB.COMPARE in SQL with CREATE TABLE using 5 parameters

```
CREATE TABLE abc nologging AS SELECT dbms_lob.compare ( col1 ,col2 ,3 ,5 ,4 ) FROM emp a ,emp b ;
```

Output

```
CREATE UNLOGGED TABLE abc AS ( SELECT MIG_ORA_EXT.MIG_CLOB_COMPARE ( col1 ,col2 ,3 ,5 ,4 )
FROM emp a ,emp b ) ;
```

Input - DBMS_LOB.COMPARE in SQL of a function (NVL2)

```
SELECT REPLACE( NVL2( DBMS_LOB.COMPARE ( ENAME ,Last_name ) ,'NO NULL' ,'ONE NULL' ) ,'NULL' )
FROM emp ;
```

Output

```
SELECT REPLACE( DECODE ( MIG_ORA_EXT.MIG_CLOB_COMPARE ( ENAME ,Last_name ) ,NULL ,'ONE
NULL' ,'NO NULL' ) ,'NULL' ,'' ) FROM emp ;
```

COMPARE in PL/SQL

Input - DBMS_LOB.COMPARE in PL/SQL

```
DECLARE v_clob clob;
        v_text varchar(1000);
        v_compare_res INT;
BEGIN
    v_clob := TO_CLOB('abcddedf');
    v_text := '123454';
    v_compare_res := dbms_lob.compare(v_clob, TO_CLOB(v_text));
    DBMS_OUTPUT.PUT_LINE(v_compare_res);
end;
/
```

Output

```
DECLARE v_clob clob;
        v_text varchar(1000);
        v_compare_res INT;
BEGIN
    v_clob := CAST('abcddedf' AS CLOB);
    v_text := '123454';
    v_compare_res := MIG_ORA_EXT.MIG_CLOB_COMPARE(v_clob,cast(v_text as CLOB));
    DBMS_OUTPUT.PUT_LINE(v_compare_res);
end;
/
```

DBMS_LOB.CREATETEMPORARY

The DBMS_LOB.CREATETEMPORARY function creates a temporary LOB and its corresponding index in the default temporary tablespace.
DBMS_LOB.FREETEMPORARY is used to delete the temporary LOB and its index.

Input - DBMS_LOB.CREATETEMPORARY with DBMS_LOB.FREETEMPORARY

```
DECLARE v_clob clob;
BEGIN
    DBMS_LOB.CREATETEMPORARY(v_clob, TRUE, DBMS_LOB.SESSION);
```



```
    v_clob := TO_CLOB('abcddedf');
    DBMS_OUTPUT.PUT_LINE(v_clob);
    DBMS_LOB.FREETEMPORARY(v_clob);
end;
/
```

Output

```
DECLARE v_clob clob;
BEGIN
    -- DBMS_LOB.CREATETEMPORARY(v_clob, TRUE, DBMS_LOB.SESSION);
    v_clob := CAST('abcddedf' AS CLOB);
    DBMS_OUTPUT.PUT_LINE(CAST(v_clob AS TEXT));
    -- DBMS_LOB.FREETEMPORARY(v_clob);
    NULL;
end;
/
```

DBMS_LOB.FREETEMPORARY

The DBMS_LOB.FREETEMPORARY function frees the temporary BLOB or CLOB in the default temporary tablespace. After the call to FREETEMPORARY, the LOB locator that is freed is marked as invalid.

Input - DBMS_LOB.CREATETEMPORARY and DBMS_LOB.FREETEMPORARY

```
DECLARE v_clob clob;
BEGIN
    DBMS_LOB.CREATETEMPORARY(v_clob, TRUE, DBMS_LOB.SESSION);
    v_clob := TO_CLOB('abcddedf');
    DBMS_OUTPUT.PUT_LINE(v_clob);
    DBMS_LOB.FREETEMPORARY(v_clob);
end;
/
```

Output

```
DECLARE v_clob clob ;
BEGIN
          /*DBMS_LOB.CREATETEMPORARY(v_clob, TRUE, DBMS_LOB.SESSION);*/
          v_clob := cast( 'abcddedf' as CLOB ) ;
          DBMS_OUTPUT.PUT_LINE ( v_clob ) ;
          /* DBMS_LOB.FREETEMPORARY(v_clob); */
          null ;
     end ;
     /
```

DBMS_LOB.INSTR

DBMS_LOB.INSTR function returns the matching position of the n^{th} occurrence of the pattern in the LOB, starting from the offset specified.

Input - DBMS_LOB.INSTR in SQL

```
SELECT expr1, …, DBMS_LOB.INSTR(str, septr, 1, 5)
  FROM tab1
 WHERE …;
```

Output

```
SELECT expr1, …, INSTR(str, septr, 1, 5)
  FROM tab1
 WHERE …
```

Input - DBMS_LOB.INSTR in PL/SQL


```
BEGIN
  …
      pos := DBMS_LOB.INSTR(str,septr,1, i);
  ...
END;
/
```

Output

```
BEGIN
  …
      pos := INSTR(str,septr,1, i);
  ...
END;
/
```

DBMS_LOB.SUBSTR

You can specify whether to migrate this function by configuring parameter **MigDbmsLob**.

Input - DBMS_LOB.SUBSTR when MigDbmsLob is set to true

If the value of **MigDbmsLob** is **true**, then migration happens. If the value is **false**, then migration does not happen.

Input

```
SELECT dbms_lob.substr('!2d3d4dd!',1,5);
```

Output

```
If the config param is true, it should be migrated as below:
select substr('!2d3d4dd!',5,1);

If false, it should be retained as it is:
select dbms_lob.substr('!2d3d4dd!',1,5);
```

Input

```
SELECT dbms_lob.substr('!2d3d4dd!',5);
```

Output

```
If the config param is true, it should be migrated as below:
select substr('!2d3d4dd!',1,5);

If false, it should be retained as it is:
select dbms_lob.substr('!2d3d4dd!',5);
```

6.6.14.3 String Functions

This section describes the following string functions:

- **LISTAGG**
- **STRAGG**
- **WM_CONCAT**
- **NVL2 and REPLACE**
- **QUOTE**

LISTAGG

LISTAGG is used to order data in columns within each group specified in the **ORDER BY** clause and concatenates the order results.

Figure 6-36 Input - Listagg

```
SELECT
        deptno
        ,ename
        ,LISTAGG (
            ename
            ,': '
        ) OVER( PARTITION BY deptno ,ename ORDER BY ename ) AS rn
    FROM
        emp
    ORDER BY
        deptno
        ,ename
;
```

Figure 6-37 Output - Listagg

```
SELECT
        deptno
        ,ename
        ,STRING_AGG (
            ename
            ,': '
        ) OVER( PARTITION BY deptno ,ename ORDER BY ename ) AS rn
    FROM
        emp
    ORDER BY
        deptno
        ,ename
;
```

LISTAGG can be migrated after **MigSupportForListAgg** is set to **false**.

Input- LISTAGG

```
SELECT LISTAGG(BRANCH_ID, ',') WITHIN GROUP(ORDER BY AREA_ORDER) PRODUCTRANGE
                FROM (SELECT DISTINCT VB.BRANCH_ID,
                                      VB.VER_ID,
                                      VB.AREA_ORDER
                        FROM SPMS_VERSION_BRANCH VB, SPMS_NODE_SET NS
                       WHERE VB.BRANCH_TYPE IN ('1', '3')
                         AND VB.AGENCY_BRANCH = NS.BRANCH_ID);
```

Output

```
SELECT LISTAGG (BRANCH_ID,',') WITHIN GROUP (
 ORDER BY AREA_ORDER ) PRODUCTRANGE
 FROM ( SELECT
        DISTINCT VB.BRANCH_ID
        ,VB.VER_ID
        ,VB.AREA_ORDER
         FROM
```



```
        SPMS_VERSION_BRANCH VB
       ,SPMS_NODE_SET NS
        WHERE VB.BRANCH_TYPE IN (
         '1','3')
       AND VB.AGENCY_BRANCH = NS.BRANCH_ID)
;
```

STRAGG

STRAGG is a string aggregate function used to collect values from multiple rows into a comma-separated string.

Input-STRAGG

```
SELECT DEPTNO,ENAME,STRAGG(ename) over (partition by deptno order by
        ename RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
            AS ENAME_STR FROM EMP;
```

Output

```
SELECT DEPTNO,ENAME,STRING_AGG (
   ename,',') over( partition BY deptno ORDER BY
 ename RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
 FOLLOWING ) AS ENAME_STR
   FROM EMP
;
```

WM_CONCAT

WM_CONCAT is used to aggregate data from a number of rows into a single row, giving a list of data associated with a specific value.

Figure 6-38 Input - WM_Concat

```
SELECT
        deptno
        ,WM_CONCAT (ename) over( partition BY deptno ORDER BY ename ) AS OUTPUT
        ,COUNT( ename ) over( partition BY deptno ) AS tot_count
    FROM
        emp
;
```

Figure 6-39 Output - WM_Concat

```
SELECT
        deptno
        ,STRING_AGG (
            ename
            ,','
        ) over( partition BY deptno ORDER BY ename ) AS OUTPUT
        ,COUNT( ename ) over( partition BY deptno ) AS tot_count
    FROM
        emp
;
```


NVL2 and REPLACE

NVL2(expression, value1, value2) is a function used to determine the value returned by a query based on whether a specified expression is null or not. If the expression is not null, then NVL2 returns value1. If the expression is null, then NVL2 returns value 2.

Input - NVL2

```
NVL2(Expr1, Expr2, Expr3)
```

Output

```
DECODE(Expr1, NULL, Expr3, Expr2)
```

The **REPLACE** function is used to return char with every occurrence of search_string replaced with replacement_string. If replacement_string is omitted or null, then all occurrences of search_string are removed.

The **REPLACE** function in Oracle contains two mandatory parameters and one optional parameter. The **REPLACE** function in GaussDB(DWS) contains three mandatory parameters.

Input - Nested REPLACE

```
CREATE
    OR REPLACE FUNCTION F_REPLACE_COMMA ( IS_STR IN VARCHAR2 ) RETURN VARCHAR2 IS BEGIN
            IF
                    IS_STR IS NULL
                    THEN RETURN NULL ;
                ELSE
                RETURN REPLACE( REPLACE( IS_STR ,'a' ) ,CHR ( 10 ) ) ;
        END IF ;
END F_REPLACE_COMMA ;
/
```

Output

```
CREATE
    OR REPLACE FUNCTION F_REPLACE_COMMA ( IS_STR IN VARCHAR2 ) RETURN VARCHAR2 IS BEGIN
            IF
                    IS_STR IS NULL
                    THEN RETURN NULL ;
                ELSE
                RETURN REPLACE( REPLACE( IS_STR ,'a' ,'' ) ,CHR ( 10 ) ,'' ) ;
        END IF ;
end ;
/
```

Input - More than one REPLACE

```
SELECT
        REPLACE( 'JACK and JUE' ,'J', '' ) "Changes"
        ,REPLACE( 'JACK1 and JUE' ,'J' ) "Changes1"
        ,REPLACE( 'JACK2 and JUE' ,'J' ) "Changes2"
    FROM
        DUAL
;
```

Output

```
SELECT
        REPLACE( 'JACK and JUE' ,'J' ,'' ) "Changes"
        ,REPLACE( 'JACK1 and JUE' ,'J' ,'' ) "Changes1"
        ,REPLACE( 'JACK2 and JUE' ,'J' ,'' ) "Changes2"
    FROM
```



```
        DUAL
;
```

Input - REPLACE with Three parameters

```
SELECT
        REPLACE( '123tech123' ,'123', '1')
    FROM
        dual
;
```

Output

```
SELECT
        REPLACE( '123tech123' ,'123' , '1' )
    FROM
        dual
;
```

QUOTE

QUOTE allows the user to embed single-quotes in literal strings without having to resort to double quotes. That is, you can use single quotes to specify a literal string.

For example:

```
SELECT q'[I'm using quote operator in SQL statement]' "Quote (q) Operator" FROM dual;
```

Figure 6-40 Input - Quote

```
SELECT
          q'[It's a string quote operator.]'
     FROM
          dual
;
```

Figure 6-41 Output - Quote

```
SELECT
          $q$It's a string quote operator.$q$
     FROM
          dual
;
```

6.6.14.4 Analytical Functions

Analytical functions compute an aggregate value based on a group of rows. They differ from aggregate functions in that they return multiple rows for each group. Analytical functions are commonly used to compute cumulative, moving, centered, and reporting aggregates. DSC supports analytical functions including the RATIO_TO_REPORT function.

Input - Analytical Functions

```
SELECT empno, ename,  deptno
  , COUNT(*) OVER() AS cnt
  , AVG(DISTINCT empno) OVER (PARTITION BY deptno) AS cnt_dst
  FROM emp
ORDER BY empno;
```


Output

```
WITH aggDistQuery1 AS (
    SELECT
            deptno
            ,AVG (
                DISTINCT empno
            ) aggDistAlias1
        FROM
            emp
        GROUP BY
            deptno
) SELECT
        empno
        ,ename
        ,deptno
        ,COUNT( * ) OVER( ) AS cnt
        ,(
            SELECT
                    aggDistAlias1
                FROM
                    aggDistQuery1
                WHERE
                    deptno = MigTblAlias.deptno
        ) AS cnt_dst
    FROM
        emp MigTblAlias
    ORDER BY
       empno
;
```

RATIO_TO_REPORT

RATIO_TO_REPORT is an analytic function which returns the proportion of a value to a group of values.

Input - RATIO_TO_REPORT

```
SELECT last_name, salary
         , RATIO_TO_REPORT(salary) OVER () AS rr
  FROM employees
WHERE job_id = 'PU_CLERK';
```

Output

```
SELECT last_name, salary
    , salary / NULLIF( SUM (salary) OVER( ), 0 ) AS rr
  FROM employees
WHERE job_id = 'PU_CLERK';
```

Input - RATIO_TO_REPORT with AGGREGATE column in SELECT

```
SELECT
        Ename
        ,Deptno
        ,Empno
        ,SUM (salary)
        ,RATIO_TO_REPORT (
            COUNT( DISTINCT Salary )
        ) OVER( PARTITION BY Deptno ) RATIO
    FROM
        emp1
    ORDER BY
        Ename
        ,Deptno
        ,Empno
;
```

Output


```
SELECT
        Ename
        ,Deptno
        ,Empno
        ,SUM (salary)
        ,COUNT( DISTINCT Salary ) / NULLIF( SUM ( COUNT( DISTINCT Salary ) ) OVER( PARTITION BY
Deptno ) ,0 ) RATIO
    FROM
        emp1
    ORDER BY
        Ename
        ,Deptno
        ,Empno
;
```

Input - RATIO_TO_REPORT with the AGGREGATE column using extending grouping feature but OUNT (Salary) in the RATIO TO REPORT column is not present in SELECT

Use the **extendedGroupByClause** configuration parameter to configure migration of the extended GROUP BY clause.

```
SELECT
        Ename
        ,Deptno
        ,Empno
        ,SUM (salary)
        ,RATIO_TO_REPORT (
            COUNT( Salary )
        ) OVER( PARTITION BY Deptno ) RATIO
    FROM
        emp1
    GROUP BY
        GROUPING SETS (
            Ename
            ,Deptno
            ,Empno
        )
    ORDER BY
        Ename
        ,Deptno
        ,Empno
;
```

Output

```
SELECT
        Ename
        ,Deptno
        ,Empno
        ,ColumnAlias1
        ,aggColumnalias1 / NULLIF( SUM ( aggColumnalias1 ) OVER( PARTITION BY Deptno ) ,0 ) RATIO
    FROM
        (
            SELECT
                    SUM (salary) AS ColumnAlias1
                    ,COUNT( Salary ) aggColumnalias1
                    ,NULL AS Deptno
                    ,NULL AS Empno
                    ,Ename
                FROM
                    emp1
                GROUP BY
                    Ename
            UNION
            ALL SELECT
                    SUM (salary) AS ColumnAlias1
                    ,COUNT( Salary ) aggColumnalias1
                    ,Deptno
```



```
                ,NULL AS Empno
                ,NULL AS Ename
            FROM
                emp1
            GROUP BY
                Deptno
        UNION
        ALL SELECT
                SUM (salary) AS ColumnAlias1
                ,COUNT( Salary ) aggColumnalias1
                ,NULL AS Deptno
                ,Empno
                ,NULL AS Ename
            FROM
                emp1
            GROUP BY
                Empno
    )
ORDER BY
    Ename
    ,Deptno
    ,Empno
;
```

6.6.14.5 Regular Expression Functions

Regular expressions specify patterns to match strings using standardized syntax conventions. In Oracle, regular expressions are implemented using a set of SQL functions that allow you to search and use string data.

DSC can migrate **REGEXP_INSTR**, **REGEXP_SUBSTR**, and **REGEXP_REPLACE** regular expressions. Details are as follows:

- Regexp (REGEXP_INSTR and REGEXP_SUBSTR) that includes the **sub_expr** parameter are not supported. If the input script includes **sub_expr**, the DSC will log an error for it.
- Regexp (REGEXP_INSTR, REGEXP_SUBSTR, and REGXP_REPLACE) uses the **match_param** parameter to set the default matching behavior. The DSC supports values i (case-insensitive) and c (case-sensitive) for this parameter. Other values for **match_param** are not supported.
- Regexp (REGEXP_INSTR) uses the **return_option** parameter to set what is returned for regexp. The DSC supports the value 0 (zero) for this parameter. Other values for return_option are not supported.

REGEXP_INSTR

REGEXP_INSTR extends the functionality of the INSTR function by supporting the regular expression pattern for the search string. REGEXP_INSTR with 2 to 6 parameters are supported for migration.

The **sub_expr** parameter (parameter #7) is available in Oracle but is not supported for migration. If the input script includes **sub_expr**, the DSC will log an error for it.

For **return_option**, the value 0 (zero) is supported. Other values for return_option are not supported.

For **match_param**, values i (case-insensitive) and c (case-sensitive) are supported. Other values for **match_param** are not supported.

```
REGEXP_INSTR(
    string,
```



```
    pattern,
    [start_position,]
    [nth_appearance,]
    [return_option,]
    [match_param,]
    [sub_expr]
)
```

Bulk Operations

Input - REGEXP_INSTR

```
SELECT
        REGEXP_INSTR( 'TechOnTheNet is a great resource' ,'t' )
    FROM
        dual
;
```

Output

```
SELECT
        MIG_ORA_EXT.REGEXP_INSTR (
            'TechOnTheNet is a great resource'
            ,'t'
        )
    FROM
        dual
;
```

Input - REGEXP_INSTR with 7 parameters (Invalid)

```
SELECT
        Empno
        ,ename
        ,REGEXP_INSTR( ename ,'a|e|i|o|u' ,1 ,1 ,0 ,'i' ,7 ) AS Dname
    FROM
        emp19
;
```

Output

The input expression has 7 parameters. Since the tool supports REGEXP_INSTR with 2 to 6 arguments, an error will be logged, starting "*Seven(7) arguments for REGEXP_INSTR function is not supported.*"

```
SELECT
        Empno
        ,ename
        ,REGEXP_INSTR( ename ,'a|e|i|o|u' ,1 ,1 ,0 ,'i' ,7 ) AS Dname
    FROM
        emp19
;
```

BLogic Operations

Input - REGEXP_INSTR

```
CREATE OR REPLACE FUNCTION myfct
RETURN VARCHAR2
IS
res VARCHAR2(200) ;
BEGIN
   res := 100 ;
   INSERT INTO emp19 RW(RW.empno,RW.ename,dname) SELECT res, RWN.ename key
, regexp_instr(ename ,'[ae]',4,2,0, 'i')   as Dname FROM   emp19 RWN ;

   RETURN res ;
END ;
/
```


Output

```
CREATE
    OR REPLACE FUNCTION myfct RETURN VARCHAR2 IS res VARCHAR2 ( 200 ) ;
    BEGIN
        res := 100 ;
        INSERT INTO emp19 ( empno ,ename ,dname ) SELECT
            res ,RWN.ename "KEY" ,MIG_ORA_EXT.REGEXP_INSTR ( ename ,'[ae]' ,4 ,2 ,0 ,'i' ) as Dname
        FROM
            emp19 RWN ;
            RETURN res ; END ;
/
```

REGEXP_SUBSTR

REGEXP_SUBSTR extends the functionality of the SUBSTR function by supporting regular expression pattern for the search string. REGEXP_SUBSTR with 2 to 5 parameters are supported for migration.

The **sub_expr** parameter (parameter #6) is available in Oracle but is not supported for migration. If the input script includes **sub_expr**, the DSC will log an error for it.

For **match_param**, values i (case-insensitive) and c (case-sensitive) are supported. Other values for **match_param** are not supported.

```
REGEXP_SUBSTR(
   string,
     pattern,
     [start_position,]
     [nth_appearance,]
     [match_param,]
     [sub_expr]
 )
```

Bulk Operations

Input - REGEXP_SUBSTR

```
SELECT
        Ename
        ,REGEXP_SUBSTR( 'Programming' ,'(\w).*?\1' ,1 ,1 ,'i' )
    FROM
        emp16
;
```

Output

```
SELECT
        Ename
        ,MIG_ORA_EXT.REGEXP_SUBSTR (
            'Programming'
            ,'(\w).*?\1'
            ,1
            ,1
            ,'i'
        )
    FROM
        emp16
;
```

Input - REGEXP_SUBSTR

```
SELECT
        REGEXP_SUBSTR( '1234567890' ,'(123)(4(56)(78))' ,1 ,1 ,'i'  ) "REGEXP_SUBSTR"
    FROM
```



```
        DUAL
;
```

Output

```
SELECT
        MIG_ORA_EXT.REGEXP_SUBSTR (
            '1234567890'
            ,'(123)(4(56)(78))'
            ,1
            ,1
            ,'i'
        ) "REGEXP_SUBSTR"
    FROM
        DUAL
;
```

Input - REGEXP_SUBSTR with 6 parameters (Invalid)

```
SELECT
        REGEXP_SUBSTR( '1234567890' ,'(123)(4(56)(78))' ,1 ,1 ,'i' ,1 ) "REGEXP_SUBSTR"
    FROM
        DUAL
;
```

Output

The input expression has 6 arguments. Since the tool supports REGEXP_SUBSTR with 2 to 5 parameters an error will be logged, starting "*Error message: Six(6) arguments for REGEXP_SUBSTR function is not supported.*"

```
SELECT
        REGEXP_SUBSTR( '1234567890' ,'(123)(4(56)(78))' ,1 ,1 ,'i' ,1 ) "REGEXP_SUBSTR"
    FROM
        DUAL
;
```

BLogic Operations

Input - REGEXP_SUBSTR

```
CREATE OR REPLACE FUNCTION myfct
RETURN VARCHAR2
IS
res VARCHAR2(200) ;
BEGIN
   res := 100 ;
   INSERT INTO emp19 RW(RW.empno,RW.ename,dname) SELECT res, RWN.ename key
,REGEXP_ SUBSTR ('TechOnTheNet', 'a|e|i|o|u', 1, 1, 'i') as Dname FROM   emp19 RWN ;

   RETURN res ;
END ;
/
```

Output

```
CREATE
    OR REPLACE FUNCTION myfct RETURN VARCHAR2 IS res VARCHAR2 ( 200 ) ;
    BEGIN
        res := 100 ;
        INSERT INTO emp19 ( empno ,ename ,dname ) SELECT
            res ,RWN.ename "KEY" ,MIG_ORA_EXT.REGEXP_ SUBSTR ( 'TechOnTheNet' ,'a|e|i|o|u' ,1 ,1 ,'i' ) as
Dname
        FROM
            emp19 RWN ;
            RETURN res ;
END ;
/
```


REGEXP_REPLACE

REGEXP_REPLACE extends the functionality of the REPLACE function by supporting regular expression pattern for the search string. REGEXP_REPLACE with 2 to 6 parameters are supported for migration.

For **match_param**, values i (case-insensitive) and c (case-sensitive) are supported. Other values for **match_param** are not supported.

```
REGEXP_REPLACE(
     string,
     pattern,
     [replacement_string,]
     [start_position,]
     [nth_appearance,]
     [match_param]
)
```

Bulk Operations

Input - REGEXP_REPLACE

```
SELECT
        testcol
        ,regexp_replace( testcol ,'([[:digit:]]{3})\.([[:digit:]]{3})\.([[:digit:]]{4})' ,'(\1) \2-\3' ) RESULT
    FROM
        test
    WHERE
        LENGTH( testcol ) = 12
;
```

Output

```
SELECT
        testcol
        ,MIG_ORA_EXT.REGEXP_REPLACE (
            testcol
            ,'([[:digit:]]{3})\.([[:digit:]]{3})\. ([[:digit:]]{4})'
            ,'(\1) \2-\3'
        ) RESULT
    FROM
        test
    WHERE
        LENGTH( testcol ) = 12
;
```

Input - REGEXP_REPLACE

```
SELECT
        UPPER( regexp_replace ( 'foobarbequebazilbarfbonk barbeque' ,'(b[^b]+)(b[^b]+)' ) )
    FROM
        DUAL
;
```

Output

```
SELECT
        UPPER( MIG_ORA_EXT.REGEXP_REPLACE ( 'foobarbequebazilbarfbonk barbeque' ,'(b[^b]+)(b[^b]
+)' ) )
    FROM
        DUAL
;
```

Input - REGEXP_REPLACE with 7 parameters (Invalid)

```
SELECT
        REGEXP_REPLACE( 'TechOnTheNet' ,'a|e|i|o|u' ,'Z' ,1 ,1 ,'i' ,'(\1) \2-\3' ) AS First_Occurrence
    FROM
```



```
        emp
;
```

Output

The input expression has 7 parameters. Since the tool supports REGEXP_REPLACE with 2 to 6 parameters, an error will be logged, starting "*Too many arguments for REGEXP_REPLACE function [Max:6 argument(s) is/are allowed].*"

```
SELECT
        REGEXP_REPLACE( 'TechOnTheNet' ,'a|e|i|o|u' ,'Z' ,1 ,1 ,'i' ,'(\1) \2-\3' ) AS First_Occurrence
    FROM
        emp
;
```

BLogic Operations

Input - REGEXP_REPLACE

```
CREATE OR REPLACE FUNCTION myfct
RETURN VARCHAR2
IS
res VARCHAR2(200) ;
BEGIN
   res := 100 ;
   INSERT INTO emp19 RW(RW.empno,RW.ename,dname) SELECT res, RWN.ename key
,REGEXP_REPLACE ('TechOnTheNet', 'a|e|i|o|u', 'Z', 1, 1, 'i') as Dname FROM   emp19 RWN ;

   RETURN res ;
END ;
/
```

Output

```
CREATE
    OR REPLACE FUNCTION myfct RETURN VARCHAR2 IS res VARCHAR2 ( 200 ) ;
    BEGIN
        res := 100 ;
        INSERT INTO emp19 ( empno ,ename ,dname ) SELECT
            res ,RWN.ename "KEY" ,MIG_ORA_EXT.REGEXP_REPLACE ( 'TechOnTheNet' ,'a|e|i|o|u' ,'Z' ,1 ,1 ,'i' )
as Dname
        FROM
            emp19 RWN ;
            RETURN res ;
END ;
/
```

LISTAGG/regexp_replace/regexp_instr

Configure the following parameters before migrating LISTAGG/regexp_replace/regexp_instr:

- MigSupportForListAgg=false
- MigSupportForRegexReplace=false

Input- REMOVE LISTAGG/regexp_replace/regexp_instr

```
SELECT LISTAGG(T.OS_SOFTASSETS_ID,',') WITHIN GROUP(ORDER BY T.SOFTASSETS_ID)
      INTO V_OS_SOFTASSETS_IDS
      FROM SPMS_SYSSOFT_PROP_APPR T
     WHERE T.APPR_ID = I_APPR_ID
       AND T.SYSSOFT_PROP = '001';

V_ONLY_FILE_NAME := REGEXP_REPLACE( I_FILENAME ,'.*/' ,'' ) ;

THEN v_auth_type := 102;
```



```
        ELSIF v_status IN ('0100', '0200')
            AND REGEXP_INSTR (v_role_str, ',(411|414),') > 0
```

Output

```
"SELECT LISTAGG(T.OS_SOFTASSETS_ID,',') WITHIN GROUP(ORDER BY T.SOFTASSETS_ID)
      INTO V_OS_SOFTASSETS_IDS
      FROM SPMS_SYSSOFT_PROP_APPR T
     WHERE T.APPR_ID = I_APPR_ID
       AND T.SYSSOFT_PROP = '001';

V_ONLY_FILE_NAME := REGEXP_REPLACE (I_FILENAME, '.*/', '');

THEN  v_auth_type := 102;
        ELSIF v_status IN ('0100', '0200')
                AND REGEXP_INSTR (v_role_str, ',(411|414),') > 0"
```

6.6.15 PL/SQL

This section describes the migration syntax of Oracle PL/SQL. The migration syntax determines how the keywords and features are migrated.

PL/SQL combines the procedural features of SQL and programming languages.

SQL Commands

Currently, GaussDB(DWS) does not support **set define off/on** and **spool off**. After being converted by the DSC tool, related commands are commented out in the target database.

Oracle Syntax	Syntax After Migration

set define off
spool ORACLE.log
create table product
(
  product_id   VARCHAR2(20),
  product_name   VARCHAR2(50)
);
spool off
	/*set define off;*/
	/*spool ORACLE.log*/
	CREATE TABLE product
	(
	  product_id VARCHAR2(20),
	  product_name VARCHAR2(50)
	);
	/*spool off*/

For details, see the following topics:

EDITIONABLE

Variable Assignment

END

EXCEPTION Handling

Subtransaction Handling

STRING

LONG

RESULT_CACHE

Relational Operators with Spaces

Substitution Variables

PARALLEL_ENABLE

TRUNCATE TABLE

ALTER SESSION

AUTONOMOUS

Procedure Call

EDITIONABLE

The **EDITIONABLE** keyword is not supported in GaussDB(DWS). So it needs to be removed from the destination database.

Input – EDITIONABLE

```
CREATE OR REPLACE EDITIONABLE PACKAGE "PACK1"."PACKAGE_SEND_MESSAGE"
AS
  TYPE filelist IS REF CURSOR;
  PROCEDURE get_message_info (in_userid         IN     VARCHAR2,
                      in_branchid       IN     VARCHAR2,
                      in_appverid       IN     VARCHAR2,
                      in_app_list_flag  IN     VARCHAR2,
                      in_filetype       IN     VARCHAR2,
                      in_filestate      IN     VARCHAR2,
                      o_retcode            OUT VARCHAR2,
                      o_errormsg           OUT VARCHAR2,
                      o_seq                OUT VARCHAR2,
                      o_totalnum           OUT NUMBER,
                      o_filelist           OUT filelist);
```

Output

```
/*~~PACKAGE_SEND_MESSAGE~~*/
CREATE
    SCHEMA PACKAGE_SEND_MESSAGE
;
```

Variable Assignment

Figure 6-42 Input - PL/SQL

```
BEGIN
 ...
 v_rowcount : = SQL%ROWCOUNT;
 ..
END;
```

Figure 6-43 Output - PL/SQL

```
BEGIN
 ...
 v_rowcount := SQL%ROWCOUNT;
 ..
END;
```


END

END with label is not supported in GaussDB(DWS), so, the label name is removed during migration.

Input - END with a procedure name

```
CREATE OR REPLACE PROCEDURE sp_ins_emp
  …
  …
  …
END sp_ins_emp;
```

Output

```
CREATE OR REPLACE PROCEDURE sp_ins_emp
  …
  …
  …
END;
```

Input - END with a function name

```
CREATE FUNCTION fn_get_bal
  …
  …
  …
END get_bal;
/
```

Output

```
CREATE FUNCTION fn_get_bal
  …
  …
  …
END;
/
```

EXCEPTION Handling

GaussDB(DWS) does not support **EXCEPTION** handling. To migrate scripts, set the **exceptionHandler** parameter to **True**.

For DSC, this parameter must be set to the default value **False**.

Figure 6-44 Input - EXCEPTION Handling

```
1  CREATE
2      OR REPLACE FUNCTION get_salary ( n_emp_id NUMBER ) RETURN NUMBER IS n_salary NUMBER ( 8 ,2 ) ;
3      BEGIN
4           SELECT
5                     salary INTO n_salary
6                FROM
7                     employees
8                WHERE
9                     id = n_emp_id ;
10                    RETURN n_salary ;
11               EXCEPTION WHEN NO_DATA_FOUND
12          THEN RETURN NULL ;
13          WHEN TOO_MANY_ROWS
14     THEN RETURN NULL ;
15 END get_salary ;
16 /
```


Figure 6-45 Output - EXCEPTION Handling

```
CREATE
    OR REPLACE FUNCTION get_salary ( n_emp_id NUMBER ) RETURN NUMBER IS n_salary NUMBER ( 8 ,2 ) ;
    BEGIN
         SELECT
                  salary INTO n_salary
              FROM
                   employees
              WHERE
                   id = n_emp_id ;
                   RETURN n_salary ;
              /* EXCEPTION WHEN NO_DATA_FOUND THEN RETURN NULL ; WHEN TOO_MANY_ROWS THEN RETURN NULL ;  */
    end ;


    /
```

Subtransaction Handling

Subtransaction (that is commit and rollback statements in PL/SQL) is not supported. This parameter must be set to the default **True**.

Figure 6-46 Input - Subtransaction Handling

```
CREATE
     OR REPLACE FUNCTION SUB_TRANSACTION ( x NUMBER ,y VARCHAR2 ) RETURN NUMBER IS id_val NUMBER ( 8 ,2 ) ;
     BEGIN
          INSERT INTO employees ( id ,first_name )
     VALUES ( x ,y ) ;
     UPDATE
          employees
     SET
          id = x
     WHERE
          first_name = y ;

          commit ;
     select
          id into id_val
     from
          employees
     where
          first_name = 'James' ;
          DELETE
          FROM
               employees
          WHERE
               first_name = y ;
               RETURN id_val ;

          Rollback ;
END SUB_TRANSACTION ;
/
```


Figure 6-47 Output - Subtransaction Handling

```
CREATE
    OR REPLACE FUNCTION SUB_TRANSACTION ( x NUMBER ,y VARCHAR2 ) RETURN NUMBER IS id_val NUMBER ( 8 ,2 ) ;
    BEGIN
         INSERT INTO employees ( id ,first_name ) select
                   x ,y ;
                   UPDATE
              employees
         SET
              id = x
         WHERE
              first_name = y ;
              /* commit; */
         null ;
         select
              id into id_val
         from
              employees
         where
              first_name = 'James' ;
              DELETE
              FROM
                   employees
              WHERE
                   first_name = y ;
                   RETURN id_val ;
              /* Rollback; */
         null ;
end ;
/
```

STRING

The Oracle PL/SQL string type is not supported by GaussDB(DWS). This data type is handled by using VARCHAR.

Figure 6-48 Input - STRING

```
--STRING
 CREATE
    OR REPLACE FUNCTION text_length ( a CLOB ) RETURN String DETERMINISTIC IS BEGIN
              RETURN DBMS_LOB.GETLENGTH ( a ) ;
    END text_length ;
    /
```

Figure 6-49 Output - STRING

```
/* STRING */
 CREATE
     OR REPLACE FUNCTION text_length ( a CLOB ) RETURN VARCHAR DETERMINISTIC IS BEGIN
               RETURN DBMS_LOB.GETLENGTH ( a ) ;
     end ;
     /
```

LONG

LONG is migrated as TEXT.

Input - LONG

```
CREATE OR REPLACE FUNCTION fn_proj_det
                    ( i_proj_cd INT )
 RETURN LONG
 IS
   v_proj_det LONG;
 BEGIN
    SELECT proj_det
      INTO v_proj_det
      FROM project
     WHERE proj_cd = i_proj_cd;

   RETURN v_proj_det;
```



```
END;
/
```

Output

```
CREATE OR REPLACE FUNCTION fn_proj_det
                    ( i_proj_cd INT )
 RETURN TEXT
 IS
    v_proj_det TEXT;
 BEGIN
     SELECT proj_det
       INTO v_proj_det
       FROM project
      WHERE proj_cd = i_proj_cd;

    RETURN v_proj_det;
 END;
 /
```

RESULT_CACHE

When a function with result cache is called, Oracle executes the function, adds the result to the result cache, and then returns the function.

When the function call is repeated, Oracle fetches the results from the cache rather than re-executing the function.

Under certain scenarios, this caching behavior can result in significant performance gains.

The target database does not support this keyword, which will be removed from the target file.

Figure 6-50 Input - RESULT_CACHE

```
CREATE OR REPLACE FUNCTION fn_get_emp_by_eno
            ( val_in IN NUMBER )
RETURN NUMBER
RESULT_CACHE
IS
  l_returnvalue NUMBER;
BEGIN
  SELECT deptno
    INTO l_returnvalue
  FROM emp t
 WHERE  t.empno = val_in;

  RETURN l_returnvalue;
END  fn_get_emp_by_eno;
/
```


Figure 6-51 Output - RESULT_CACHE

```
CREATE OR REPLACE FUNCTION fn_get_emp_by_eno
             ( val_in IN NUMBER )
RETURN NUMBER

IS
   l_returnvalue NUMBER ;
BEGIN
        SELECT deptno
            INTO l_returnvalue
          FROM emp t
       WHERE t.empno = val_in ;

       RETURN l_returnvalue ;
END;
/
```

Relational Operators with Spaces

The relational operators (<=, >=, !=) with spaces are not supported by GaussDB(DWS). DSC removes spaces between the operators.

Figure 6-52 Input - Relational operator

```
--RELATIONAL OPERATOR
CREATE
    OR REPLACE FUNCTION REL_OPTR ( n_emp_id NUMBER ) RETURN NUMBER IS n_salary NUMBER ( 8 ,2 ) ;
    BEGIN
         SELECT
                 salary INTO n_salary
            FROM
                 employees
            WHERE
                 id > = n_emp_id ;
                 RETURN n_salary ;
    END REL_OPTR ;
    /
```

Figure 6-53 Output - Relational operator

```
/* RELATIONAL OPERATOR */
CREATE
    OR REPLACE FUNCTION REL_OPTR ( n_emp_id NUMBER ) RETURN NUMBER IS n_salary NUMBER ( 8 ,2 ) ;
    BEGIN
         SELECT
                 salary INTO n_salary
            FROM
                 employees
            WHERE
                 id >= n_emp_id ;
                 RETURN n_salary ;
    end ;
    /
```

Substitution Variables

Substitution variables are a feature of Oracle SQL*Plus tool. When a substitution variable is used in a statement, SQL*Plus requests an input value and rewrites the statement to include it. The rewritten statement is passed to the Oracle database. When the Oracle script input contains any substitution variables, the DSC displays the following message. Messages are recorded in the console and log files.


```
**************************************************************************
USER ATTENTION!!!  Variable: &bbid should be substituted in the file : "/home/testmigration/V100R002C60/
MigrationTool/Input/proc_frss_jczbsc.SQL"  Variable: &wdbs should be substituted in the file : "/home/
testmigration/V100R002C60/MigrationTool/Input/proc_frss_jczbsc.SQL"  Variable: &batch_no should be
substituted in the file : "/home/testmigration/V100R002C60/MigrationTool/Input/proc_frss_jczbsc.SQL"
**************************************************************************
```

PARALLEL_ENABLE

In Oracle, PARALLEL_ENABLE is used to enable parallel loading among partitions.

Input - PARALLEL_ENABLE

```
CREATE OR REPLACE FUNCTION F_REPLACE_COMMA (IS_STR IN VARCHAR2)
RETURN VARCHAR2
parallel_enable
IS
BEGIN
   IF IS_STR IS NULL THEN
      RETURN NULL;
   ELSE
      RETURN REPLACE(REPLACE(IS_STR, CHR(13) || CHR(10), ''), ',', ', ');
   END IF;
END F_REPLACE_COMMA;
/
```

Output

```
CREATE OR REPLACE FUNCTION F_REPLACE_COMMA (IS_STR IN VARCHAR2)
RETURN VARCHAR2
IS
BEGIN
   IF IS_STR IS NULL THEN
      RETURN NULL;
   ELSE
      RETURN REPLACE(REPLACE(IS_STR, CHR(13) || CHR(10), ''), ',', ', ');
   END IF;
END;
/
```

PARALLEL Clause

PARALLEL should be commented.

Input

```
CREATE TABLE PRODUCT
    ( prod_id       INTEGER      NOT NULL PRIMARY KEY
    , prod_code     VARCHAR(5)
    , prod_name     VARCHAR(100)
    , unit_price    NUMERIC(6,2) NOT NULL )
   PARALLEL 8;
```

Output

```
CREATE TABLE PRODUCT
    ( prod_id       INTEGER      NOT NULL PRIMARY KEY
    , prod_code     VARCHAR(5)
    , prod_name     VARCHAR(100)
    , unit_price    NUMERIC(6,2) NOT NULL )
     /* PARALLEL 8 */;
```

TRUNCATE TABLE

The **TRUNCATE TABLE** statement in Oracle is used to remove all records from a table. It performs the same function as a DELETE statement without a WHERE

clause. After truncating, the table will exist but it will be empty. DSC supports migration of TRUNCATE TABLE statements with static table names only. Migration of TRUNCATE TABLE statements with dynamic table names are not supported by the tool.

NOTE

The tool does not support migration of TRUNCATE TABLE statements with dynamic table names.

Example: l_table :='truncate table ' || **itable_name**

In this example, **itable_name** indicates a dynamic table name and is not supported by the DSC. The unsupported statements will be copied verbatim to the migrated scripts.

Input - TRUNCATE TABLE with Execute Immediate

```
CREATE OR REPLACE PROCEDURE schema1.proc1
AS
BEGIN
   EXECUTE IMMEDIATE 'TRUNCATE TABLE QUERY_TABLE';
End proc1;
/
```

Output

```
CREATE
    OR REPLACE PROCEDURE schema1.proc1 AS BEGIN
        EXECUTE IMMEDIATE 'TRUNCATE TABLE schema1.QUERY_TABLE' ;
    end ;
/
```

Input - TRUNCATE TABLE inside procedure

NOTE

Migration tool does not add schema names for dynamic PL/SQL statements.

```
CREATE
    OR REPLACE PROCEDURE schemName.sp_dd_table ( itable_name VARCHAR2 ) IS l_table VARCHAR2
( 255 ) ;
    BEGIN
        l_table :='truncate table ' || itable_name ;
        ---- dbms_utility.exec_ddl_statement(l_table);
dbms_output.put_line ( itable_name || ' ' || 'Truncated' ) ;
END sp_dd_table ;
/
```

Output

```
CREATE
    OR REPLACE PROCEDURE schemName.sp_dd_table ( itable_name VARCHAR2 ) IS l_table VARCHAR2
( 255 ) ;
    BEGIN
        l_table :='truncate table ' || itable_name ;
/*
dbms_utility.exec_ddl_statement(l_table); */
dbms_output.put_line ( itable_name || ' ' || 'Truncated' ) ;
end ;
/
```

ALTER SESSION

The **ALTER SESSION** statement in Oracle is used to set or modify the parameters and behavior of the database connection. The statement stays in effect until you disconnect from the database. The DSC supports the migration of ALTER SESSION as follows:

- ALTER SESSION with ADVISE, ENABLE, DISABLE, CLOSE and FORCE clauses are migrated as commented scripts.
- ALTER SESSION with SET CLAUSE parameter (Example: NLS_DATE_FORMAT, NLS_DATE_LANGUAGE, and so on) are copied verbatim.

NOTE

The tool does not support migration of ALTER SESSION statements that have a variable for the command clause.

Example: EXECUTE IMMEDIATE ' alter session ' || ***command_val*** || 'parallel ' || type_value.

In this example, **command_val** is a variable and this is not supported by the DSC. The unsupported statements will be copied verbatim in the migrated scripts.

Input - ALTER SESSION

```
ALTER SESSION ENABLE  PARALLEL DDL;
ALTER SESSION  ADVISE COMMIT;
ALTER SESSION CLOSE DATABASE LINK local;
ALTER SESSION  SET NLS_DATE_FORMAT = 'YYYY MM DD HH24:MI:SS';
ALTER SESSION SET current_schema = 'isfc';
```

Output

```
/*ALTER SESSION ENABLE PARALLEL DDL;*/
/*ALTER SESSION ADVISE COMMIT;*/
/*ALTER SESSION CLOSE DATABASE LINK local;*/
ALTER SESSION SET NLS_DATE_FORMAT = 'YYYY MM DD HH24:MI:SS';
ALTER SESSION SET current_schema = 'isfc';
```

Input - ALTER SESSION

```
CREATE OR REPLACE
    PROCEDURE PUBLIC .TEST_CALL is
   command_val varchar2 ( 1000 ) ;
    type_value number ;
   BEGIN
             command_val := 'enable parallel ddl' ;
             dbms_output.put_line ( mike ) ;
-- execute immediate 'ALTER SESSION DISABLE GUARD' ;
            execute immediate 'ALTER SESSION ADVISE ROLLBACK' ;
EXECUTE IMMEDIATE ' alter session ' || command_val || 'parallel ' || type_value ;
END TEST_CALL;
/
```

Output

```
CREATE OR REPLACE
    PROCEDURE PUBLIC.TEST_CALL is
    command_val varchar2 ( 1000 ) ;
    type_value number ;
    BEGIN
              command_val := 'enable parallel ddl' ;
dbms_output.put_line ( mike ) ;
/*  execute immediate 'ALTER SESSION DISABLE GUARD' ; */
            execute immediate '/*ALTER SESSION ADVISE ROLLBACK*/' ;
EXECUTE IMMEDIATE 'alter session ' || command_val || 'parallel ' || type_value ;
 END ;
 /
```

AUTONOMOUS

Input - AUTONOMOUS

```
CREATE OR REPLACE EDITIONABLE PACKAGE BODY "Pack1"."DEMO_PROC" is
   PROCEDURE log(proc_name IN VARCHAR2, info IN VARCHAR2) IS
   PRAGMA AUTONOMOUS_TRANSACTION;
```

Output

```
CREATE OR REPLACE PROCEDURE DEMO_PROC.log ( proc_name IN VARCHAR2 ,info IN VARCHAR2 ) IS
/*PRAGMA AUTONOMOUS_TRANSACTION;*/
```

Procedure Call

Procedure with no parameter needs to put () after procedure name while calling the same procedure.

For example, pkg_etl.clear_temp_tables()

Input

```
CREATE OR REPLACE PACKAGE BODY IC_STAGE.pkg_etl
AS
 PROCEDURE clear_temp_tables
 IS
 BEGIN
  NULL;
 END clear_temp_tables;
END pkg_etl;
/
CREATE OR REPLACE PACKAGE BODY IC_STAGE.PKG_REVN_ARPU
AS
   PROCEDURE AGGR_X_AGG00_REVN_DEALER (p_date    PLS_INTEGER,
                                       p_days    PLS_INTEGER)
   AS
      v_start_date   PLS_INTEGER;
      v_curr_date    PLS_INTEGER;
   BEGIN
      v_start_date := TO_CHAR (TO_DATE (p_date, 'yyyymmdd') - (p_days - 1), 'yyyymmdd');
      v_curr_date := p_date;

      WHILE (v_curr_date >= v_start_date)
      LOOP
         pkg_etl.clear_temp_tables;
         pkg_dw.bind_variable ('v_curr_date', v_curr_date);

   v_curr_date := TO_CHAR (TO_DATE (v_curr_date, 'yyyymmdd') - 1, 'yyyymmdd');
      END LOOP;

   END;
END PKG_REVN_ARPU;
/
```

Output

```
CREATE OR REPLACE PROCEDURE IC_STAGE.pkg_etl#clear_temp_tables PACKAGE IS
BEGIN
    NULL ;
END ;
/

CREATE OR REPLACE PROCEDURE IC_STAGE.PKG_REVN_ARPU#AGGR_X_AGG00_REVN_DEALER
 ( p_date INTEGER
    , p_days INTEGER )
PACKAGE
AS
 v_start_date  INTEGER;
    v_curr_date  INTEGER;
BEGIN
     v_start_date := TO_CHAR( TO_DATE( p_date ,'yyyymmdd' ) - ( p_days - 1 ), 'yyyymmdd' ) ;
     v_curr_date := p_date ;
```



```
    WHILE ( v_curr_date >= v_start_date )
 LOOP
        pkg_etl#clear_temp_tables ( ) ;
        pkg_dw.bind_variable ( 'v_curr_date' ,v_curr_date ) ;
        v_curr_date := TO_CHAR( TO_DATE( v_curr_date ,'yyyymmdd' ) - 1,'yyyymmdd' ) ;
    END LOOP ;
END ;
/
```

Function Name Having No Parameter Is Called

Function name which does not have any parameter, called by function name with parameter is not supported in EXCEPTION statement. For example, in **SAD.SAD_CALC_ITEM_PKG_TEST_OB#error_msg ( )**, **error_msg** is defined without parameter, as shown below:

```
CREATE
OR REPLACE FUNCTION SAD.SAD_CALC_ITEM_PKG_TEST_OB#func_name
RETURN VARCHAR2 IS MIG_PV_VAL_DUMMY_G_FUNC_NAME VARCHAR2 ( 30 ) :=
MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE ( current_schema ( )
---
BEGIN
---
RETURN l_func_name ;
END ;
```

SCRIPTS: SAD_CALC_ITEM_PKG_TEST_OB.sql, SAD_CALC_ITEM_PRI_TEST_OB.sql

INPUT :

```
CREATE OR REPLACE PACKAGE BODY "SAD"."SAD_CALC_ITEM_PKG_TEST_OB" IS
PROCEDURE back_sad_cost_line_t(pi_contract_number      IN VARCHAR2,
pi_quotation_id         IN NUMBER,
pi_product_code         IN VARCHAR2,
pi_process_batch_number IN NUMBER,
po_error_msg            OUT VARCHAR2) IS
BEGIN
---
LOOP
INSERT INTO sad_cost_line_bak
(processing_batch_number,
contract_number,
product_code,
quotation_id,
item_code,
refresh_date,
split_date,
error_msg,
created_by,
creation_date,
last_updated_by,
last_update_date)
VALUES
(pi_process_batch_number,
cur_1.contract_number,
cur_1.product_code,
cur_1.quotation_id,
cur_1.item_code,
cur_1.refresh_date,
cur_1.split_date,
cur_1.error_msg,
cur_1.created_by,
cur_1.creation_date,
cur_1.last_updated_by,
cur_1.last_update_date);
END LOOP;
---
WHEN OTHERS THEN
```



```
po_error_msg := 'Others Exception raise in ' || func_name || ',' || SQLERRM;
END back_sad_cost_line_t;
END SAD_CALC_ITEM_PKG_TEST_OB;
```

OUTPUT :

```
CREATE
OR REPLACE PROCEDURE SAD.SAD_CALC_ITEM_PKG_TEST_OB#back_sad_cost_line_t ( pi_contract_number
IN VARCHAR2
,pi_quotation_id IN NUMBER
,pi_product_code IN VARCHAR2
,pi_process_batch_number IN NUMBER
,po_error_msg OUT VARCHAR2 ) IS MIG_PV_VAL_DUMMY_G_FUNC_NAME VARCHAR2 ( 30 ) :=
MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE ( current_schema ( )
,'SAD_CALC_ITEM_PKG_TEST_OB'
,'g_func_name' ) ::VARCHAR2 ( 30 ) ;
ex_data_error
EXCEPTION ;
ex_prog_error
EXCEPTION ;
BEGIN
---
LOOP
INSERT INTO sad_cost_line_bak (
processing_batch_number
,contract_number
,product_code
,quotation_id
,item_code
,refresh_date
,split_date
,SAD.SAD_CALC_ITEM_PKG_TEST_OB#error_msg ( )
,created_by
,creation_date
,last_updated_by
,last_update_date
)
VALUES
( pi_process_batch_number ,cur_1.contract_number ,cur_1.product_code ,cur_1.quotation_id ,cur_1.item_code
 ,cur_1.refresh_date ,cur_1.split_date ,cur_1.error_msg ,cur_1.created_by ,cur_1.creation_date ,cur_1.last_updat
ed_by ,cur_1.last_update_date ) ;
END LOOP ;
---
WHEN OTHERS THEN
po_error_msg := 'Others Exception raise in ' || SAD.SAD_CALC_ITEM_PKG_TEST_OB#func_name ( ) || ',' ||
SQLERRM ;
END ;
```

Input

```
CREATE OR REPLACE PACKAGE BODY SAD.bas_dml_lookup_pkg IS
  g_pkg_name CONSTANT VARCHAR2(30) := 'bas_dml_ic_price_rule_pkg' ;
  g_func_name VARCHAR2(100);

  FUNCTION func_name
  RETURN VARCHAR2
  IS
    l_func_name VARCHAR2(100) ;
  BEGIN
     l_func_name := g_pkg_name || '.' || g_func_name ;
     RETURN l_func_name ;

   END ;

  PROCEDURE data_change_logs ( pi_table_name        IN VARCHAR2
                             , pi_table_key_columns IN VARCHAR2
                             , po_error_msg         OUT VARCHAR2
       )
  IS
  BEGIN
```



```
  g_func_name := 'insert_fnd_data_change_logs_t';

  INSERT INTO fnd_data_change_logs_t
    ( logid, table_name, table_key_columns )
  VALUES
    ( fnd_data_change_logs_t_s.NEXTVAL
    , pi_table_name, pi_table_key_columns );
 EXCEPTION
  WHEN OTHERS THEN
    po_error_msg := 'Others Exception raise in ' || func_name || ',' || SQLERRM;
 END data_change_logs;

END bas_dml_lookup_pkg;
/
```

Output

```
CREATE OR REPLACE FUNCTION SAD.bas_dml_lookup_pkg#func_name
RETURN VARCHAR2
IS
    MIG_PV_VAL_DUMMY_G_PKG_NAME VARCHAR2(30) := MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE
( 'SAD', 'BAS_DML_LOOKUP_PKG', 'G_PKG_NAME' )::VARCHAR2(30) ;
    MIG_PV_VAL_DUMMY_G_FUNC_NAME VARCHAR2(100) := MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE
( 'SAD', 'BAS_DML_LOOKUP_PKG', 'G_FUNC_NAME' )::VARCHAR2(100) ;
    l_func_name VARCHAR2(100) ;
BEGIN
    l_func_name := MIG_PV_VAL_DUMMY_G_PKG_NAME || '.' || MIG_PV_VAL_DUMMY_G_FUNC_NAME ;
    RETURN l_func_name ;

END ;
/
CREATE OR REPLACE PROCEDURE SAD.bas_dml_lookup_pkg#data_change_logs ( pi_table_name IN
VARCHAR2
                , pi_table_key_columns IN VARCHAR2
                , po_error_msg OUT VARCHAR2 )
IS
 MIG_PV_VAL_DUMMY_G_FUNC_NAME VARCHAR2(30) := MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE
( 'SAD' ,'BAS_DML_LOOKUP_PKG' ,'G_FUNC_NAME' )::VARCHAR2(30) ;
BEGIN
    MIG_PV_VAL_DUMMY_G_FUNC_NAME := 'insert_fnd_data_change_logs_t' ;

    INSERT INTO fnd_data_change_logs_t (
        logid,table_name,table_key_columns )
    VALUES ( NEXTVAL ( 'fnd_data_change_logs_t_s' )
          , pi_table_name, pi_table_key_columns ) ;

    MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE ( 'SAD', 'BAS_DML_LOOKUP_PKG', 'G_FUNC_NAME',
MIG_PV_VAL_DUMMY_G_FUNC_NAME ) ;

    EXCEPTION
      WHEN OTHERS THEN
        po_error_msg := 'Others Exception raise in ' || SAD.bas_dml_lookup_pkg#func_name( ) || ',' ||
SQLERRM ;
        MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE ( 'SAD', 'BAS_DML_LOOKUP_PKG', 'G_FUNC_NAME',
MIG_PV_VAL_DUMMY_G_FUNC_NAME ) ;

END ;
/
```

6.6.16 PL/SQL Collections (Using User-Defined Types)

This section describes the migration syntax of Oracle PL/SQL Collections. The migration syntax decides how the keywords/features are migrated.

A user-defined type (UDT) is a data type that is derived from a supported data type.

UDT determines how data behaves and is structured in applications, using both built-in and user-defined data types. UDT makes it easier to work with PL/SQL collections.

UDT Table

The table type is created to track the structure of the UDT. No data will be stored in the table.

Input - CREATE TABLE TYPE

```
CREATE <OR REPLACE> TYPE <schema.>inst_no_type IS TABLE OF VARCHAR2 (32767);
```

Output

```
CREATE TABLE<schema.>mig_inst_no_type
      ( typ_col VARCHAR2 (32767) );
```

UDT VArray

Input - CREATE VArray

```
CREATE TYPE phone_list_typ_demo AS VARRAY(n) OF VARCHAR2(25);
```

Output

```
CREATE TABLE mig_pone_list_typ_demo
       ( typ_col VARCHAR2 (25) );
```

Declare UDT

Input - Declare UDT

```
DECLARE
      v_SQL_txt_array
       inst_no_type <:=
       inst_no_type()>;
BEGIN
      …
```

Output

```
DECLARE
/*       v_SQL_txt_array inst_no_type <:= inst_no_type()>; */
BEGIN
        EXECUTE IMMEDIATE 'DROP TABLE IF EXISTS
         v_SQL_txt_array;
            CREATE LOCAL TEMPORARY TABLE
             v_SQL_txt_array
                ON COMMIT PRESERVE ROWS
                AS SELECT *, CAST(NULL AS INT) AS
                 typ_idx_col
                   FROM mig_inst_no_type
                  WHERE FALSE';
     …
```

UDT Count

Input - UDT - COUNT in FOR LOOP

```
BEGIN
   ...
  FOR i IN 1..v_jobnum_list.COUNT
  LOOP
    SELECT COUNT(*) INTO v_abc
```



```
      FROM ...
     WHERE ...
       AND nvl(t.batch_num,
        c_batchnum_null_num) =
        v_jobnum_list(i);
   ...
END LOOP;
...
```

Output

```
BEGIN
  ...
  FOR i IN 1..(SELECT COUNT(*) from v_jobnum_list)
  LOOP
    SELECT COUNT(*) INTO v_abc
      FROM ...
     WHERE ...
       AND nvl(t.batch_num, c_batchnum_null_num) =
       (SELECT typ_col FROM v_jobnum_list
            WHERE typ_idx_col = i);
   ...
END LOOP;
...
```

UDT Record

A Record type is used to create records and can be defined in the declarative part of any PL/SQL block, subprogram, or package.

Input - RECORD Type

```
Create
    or Replace Procedure test_proc AS TYPE t_log IS RECORD ( col1 int ,col2 emp.ename % type ) ;
    fr_wh_SQL t_log ;
BEGIN
    fr_wh_SQL.col1 := 101 ;
    fr_wh_SQL.col2 := 'abcd' ;
DBMS_OUTPUT.PUT_LINE ( fr_wh_SQL.col1 || ',' || fr_wh_SQL.col2 ) ;
END test_proc;
/
```

Output

```
Create
    or Replace Procedure test_proc AS /*TYPE t_log IS RECORD ( col1 int,col2 emp.ename%type );*/
    fr_wh_SQL RECORD ;
    MIG_t_log_col1 int ;
MIG_t_log_col2 emp.ename % type ;
BEGIN
select
        MIG_t_log_col1 as col1 ,MIG_t_log_col2 as col2 INTO FR_WH_SQL ;
        fr_wh_SQL.col1 := 101 ;
    fr_wh_SQL.col2 := 'abcd' ;
DBMS_OUTPUT.PUT_LINE ( fr_wh_SQL.col1 || ',' || fr_wh_SQL.col2 ) ;
END ;
/
```

Enhancement of User-defined types

The tool supports the enhancement of PL/SQL type of TABLE used in Oracle for specific data types and for any table column.

Input - PL/SQL type of TABLE of a specific data-type

```
DECLARE
  type       fr_wh_SQL_info_type is table of VARCHAR(10);
 fr_wh_SQL  fr_wh_SQL_info_type    [:= fr_wh_SQL_info_type()];
```



```
BEGIN
    …
```

Output

```
DECLARE
/*      type   fr_wh_SQL_info_type    is table of varchar(10); */
/*      fr_wh_SQL    fr_wh_SQL_info_type    [:= fr_wh_SQL_info_type()]; */
BEGIN
    EXECUTE IMMEDIATE 'DROP TABLE IF EXISTS mig_fr_wh_SQL_info_type;
        CREATE LOCAL TEMPORARY TABLE mig_fr_wh_SQL_info_type
            ( typ_col VARCHAR (10) )
          ON COMMIT PRESERVE ROWS' ;

     EXECUTE IMMEDIATE 'DROP TABLE IF EXISTS fr_wh_SQL;
            CREATE LOCAL TEMPORARY TABLE fr_wh_SQL
                ON COMMIT PRESERVE ROWS AS
                AS SELECT  *, CAST(NULL AS INT) AS typ_idx_col
                    FROM mig_fr_wh_SQL_info_type
                   WHERE FALSE';
          …
```

Input - PL/SQL type of TABLE of any table's column

```
DECLARE
    type         fr_wh_SQL_info_type    is table of fr_wh_SQL_info.col1%type;
    fr_wh_SQL   fr_wh_SQL_info_type    [:= fr_wh_SQL_info_type()];
BEGIN
     …
```

Output

```
DECLARE
/*      type          fr_wh_SQL_info_type    is table of fr_wh_SQL_info.col1%type; */
/*      fr_wh_SQL   fr_wh_SQL_info_type    [:= fr_wh_SQL_info_type()]; */
BEGIN
    EXECUTE IMMEDIATE 'DROP TABLE IF EXISTS mig_fr_wh_SQL_info_type;
        CREATE LOCAL TEMPORARY TABLE mig_fr_wh_SQL_info_type
            ON COMMIT PRESERVE ROWS
            AS SELECT col1 AS typ_col
                FROM fr_wh_SQL_info
               WHERE FALSE' ;

     EXECUTE IMMEDIATE 'DROP TABLE IF EXISTS fr_wh_SQL;
        CREATE LOCAL TEMPORARY TABLE fr_wh_SQL
            ON COMMIT PRESERVE ROWS AS
            AS SELECT  *, CAST(NULL AS INT) AS typ_idx_col
                FROM mig_fr_wh_SQL_info_type
               WHERE FALSE';
…
```

EXTEND

GaussDB(DWS) supports keyword **EXTEND**.

Input - Extend

```
FUNCTION FUNC_EXTEND ( in_str  IN   VARCHAR2)
    RETURN ARRYTYPE
  AS
    v_count2    INTEGER;
    v_strlist   arrytype;
    v_node      VARCHAR2 (2000);
  BEGIN
    v_count2 := 0;
    v_strlist := arrytype ();
   FOR v_i IN 1 .. LENGTH (in_str)
    LOOP
      IF v_node IS NULL
        THEN
```



```
            v_node := '';
        END IF;

        IF (v_count2 = 0) OR (v_count2 IS NULL)
        THEN
          EXIT;
        ELSE
          v_strlist.EXTEND ();
          v_strlist (v_i) := v_node;
          v_node := '';
        END IF;
    END LOOP;

    RETURN v_strlist;
END;
/
```

Output

```
FUNCTION FUNC_EXTEND ( in_str IN VARCHAR2 )
RETURN ARRYTYPE AS v_count2 INTEGER ;
v_strlist arrytype ;
v_node VARCHAR2 ( 2000 ) ;
BEGIN
        v_count2 := 0 ;
        v_strlist := arrytype ( ) ;
    FOR v_i IN 1.. LENGTH( in_str ) LOOP
        IF
            v_node IS NULL
            THEN
                v_node := '' ;
            END IF ;
            IF
                ( v_count2 = 0 )
                OR( v_count2 IS NULL )
                THEN
                    EXIT ;
                ELSE
                    v_strlist.EXTEND ( 1 ) ;
                    v_strlist ( v_i ) := v_node ;
                    v_node := '' ;
                END IF ;
            END LOOP ;
        RETURN v_strlist ;
    END ;
    /
```

RECORD

The Record type declared in the package specification is actually used in the corresponding package body.

After **plsqlCollection** is set to varray, UDT will be migrated as VARRY.

Input – RECORD

```
CREATE OR REPLACE FUNCTION func1 (i1 INT)
RETURN INT
As
TYPE r_rthpagat_list IS RECORD (--Record information about cross-border RMB business parameters
(rthpagat)
rthpagat_REQUESTID RMTS_REMITTANCE_PARAM.REQUESTID%TYPE ,rthpagat_PARAMTNAME
RMTS_REMITTANCE_PARAM.PARAMTNAME%TYPE ,rthpagat_PARAMNUM
RMTS_REMITTANCE_PARAM.PARAMNUM%TYPE ,rthpagat_PARAMSTAT
RMTS_REMITTANCE_PARAM.PARAMSTAT%TYPE ,rthpagat_REQTELLERNO
RMTS_REMITTANCE_PARAM.REQTELLERNO%TYPE ,rthpagat_REQUESTTIME
RMTS_REMITTANCE_PARAM.REQUESTTIME%TYPE ,rthpagat_HOSTERRNO
RMTS_REMITTANCE_PARAM.HOSTERRNO%TYPE ,rthpagat_HOSTERRMSG
```



```
RMTS_REMITTANCE_PARAM.HOSTERRMSG%TYPE ,rthpagat_GATBANK
RMTS_REMITTANCE_PARAM.VALUE1%TYPE ,rthpagat_GATEEBANK
RMTS_REMITTANCE_PARAM.VALUE2%TYPE ,rthpagat_TELLER
RMTS_REMITTANCE_PARAM.VALUE3%TYPE ,rthpagat_DATE
RMTS_REMITTANCE_PARAM.VALUE4%TYPE ,rthpagat_BM_GATBANK
RMTS_REMITTANCE_PARAM.VALUE5%TYPE ,rthpagat_BM_GATEEBANK
RMTS_REMITTANCE_PARAM.VALUE6%TYPE ,rthpagat_BM_LMTEL
RMTS_REMITTANCE_PARAM.VALUE7%TYPE ,rthpagat_BM_LMDAT
RMTS_REMITTANCE_PARAM.VALUE8%TYPE ) ;

 v1  r_rthpagat_list;
BEGIN

END;
/
```

Output

```
CREATE
TYPE rmts_remitparammgmt_rthpagat.r_rthpagat_list AS (/* O_ERRMSG error description */
Rthpagat_REQUESTID
    rthpagat_REQUESTID RMTS_REMITTANCE_PARAM.REQUESTID%TYPE ,rthpagat_PARAMTNAME
RMTS_REMITTANCE_PARAM.PARAMTNAME%TYPE ,rthpagat_PARAMNUM
RMTS_REMITTANCE_PARAM.PARAMNUM%TYPE ,rthpagat_PARAMSTAT
RMTS_REMITTANCE_PARAM.PARAMSTAT%TYPE ,rthpagat_REQTELLERNO
RMTS_REMITTANCE_PARAM.REQTELLERNO%TYPE ,rthpagat_REQUESTTIME
RMTS_REMITTANCE_PARAM.REQUESTTIME%TYPE ,rthpagat_HOSTERRNO
RMTS_REMITTANCE_PARAM.HOSTERRNO%TYPE ,rthpagat_HOSTERRMSG
RMTS_REMITTANCE_PARAM.HOSTERRMSG%TYPE ,rthpagat_GATBANK
RMTS_REMITTANCE_PARAM.VALUE1%TYPE ,rthpagat_GATEEBANK
RMTS_REMITTANCE_PARAM.VALUE2%TYPE ,rthpagat_TELLER
RMTS_REMITTANCE_PARAM.VALUE3%TYPE ,rthpagat_DATE
RMTS_REMITTANCE_PARAM.VALUE4%TYPE ,rthpagat_BM_GATBANK
RMTS_REMITTANCE_PARAM.VALUE5%TYPE ,rthpagat_BM_GATEEBANK
RMTS_REMITTANCE_PARAM.VALUE6%TYPE ,rthpagat_BM_LMTEL
RMTS_REMITTANCE_PARAM.VALUE7%TYPE ,rthpagat_BM_LMDAT
RMTS_REMITTANCE_PARAM.VALUE8%TYPE ) ;

CREATE OR REPLACE FUNCTION func1 (i1 INT)
RETURN INT
AS
 v1  r_rthpagat_list;
BEGIN

END;
/
```

Naming Convention of Type Name

User-defined types allow for the definition of data types that model the structure and behavior of the data in an application.

Input

```
CREATE
    TYPE t_line AS ( product_line VARCHAR2 ( 30 )
                            ,product_amount NUMBER ) ;
;
```

Output

```
CREATE
    TYPE sad_dml_product_pkg.t_line AS ( product_line VARCHAR2 ( 30 )
                                                    ,product_amount NUMBER ) ;
```

Input

```
CREATE
    TYPE t_line AS ( product_line VARCHAR2 ( 30 )
                          ,product_amount NUMBER ) ;
```

Output

```
CREATE
    TYPE SAD.sad_dml_product_pkg#t_line AS ( product_line VARCHAR2 ( 30 )
                                                          ,product_amount NUMBER ) ;
```

NOTE

- For the first output(pkg.t),if **pkgSchemaNaming** is set to **true** in the configuration, PL RECORD migration should have package name as a schema name along with a type name.
- For the second output (pkg#t), assume that TYPE belongs to sad_dml_product_pkg package.

If **pkgSchemaNaming** is set to **false** in the configuration, PL RECORD migration should have schema name as a schema name along with a package name + a type name separated by # as a type name.

SUBTYPE

PL/SQL's **SUBTYPE** statement enables the creation of aliases for user-defined subtypes or predefined data types, also known as abstract data types.

Input

```
CREATE OR REPLACE PACKAGE "SAD"."BAS_SUBTYPE_PKG" IS
SUBTYPE CURRENCY IS BAS_PRICE_LIST_T.CURRENCY%TYPE;
END bas_subtype_pkg;
/
CREATE OR REPLACE PACKAGE BODY "SAD"."BAS_SUBTYPE_PKG" IS
BEGIN
  NULL;
END bas_subtype_pkg;
/
--**********************************************************************
CREATE OR REPLACE PACKAGE BODY SAD.bas_lookup_misc_pkg IS
  FUNCTION get_currency(pi_price_type IN NUMBER) RETURN VARCHAR2 IS
    v_currency bas_subtype_pkg.currency;
  BEGIN
    g_func_name := 'get_currency';
    FOR rec_currency IN (SELECT currency FROM sad_price_type_v WHERE price_type_code = pi_price_type)
    LOOP
      v_currency := rec_currency.currency;
    END LOOP;
    RETURN v_currency;
  END get_currency;
 END SAD.bas_lookup_misc_pkg;
 /
```

Output

```
 CREATE OR REPLACE FUNCTION SAD.bas_lookup_misc_pk#get_currency(pi_price_type IN NUMBER)
RETURN VARCHAR2 IS
    v_currency BAS_PRICE_LIST_T.CURRENCY%TYPE;
  BEGIN
    g_func_name := 'get_currency';
    FOR rec_currency IN (SELECT currency FROM sad_price_type_v WHERE price_type_code = pi_price_type)
    LOOP
      v_currency := rec_currency.currency;
    END LOOP;
    RETURN v_currency;
  END ;
 /
```


NOTE

Since GaussDB(DWS) does not support **SUBTYPE**, you will have to substitute it with the actual type that was used when creating the **SUBTYPE** variable.

6.6.17 PL/SQL Packages

This section describes the migration syntax of Oracle PL/SQL **Packages** and **REF CURSOR**. The migration syntax decides how the keywords/features are migrated.

This section includes the following content:

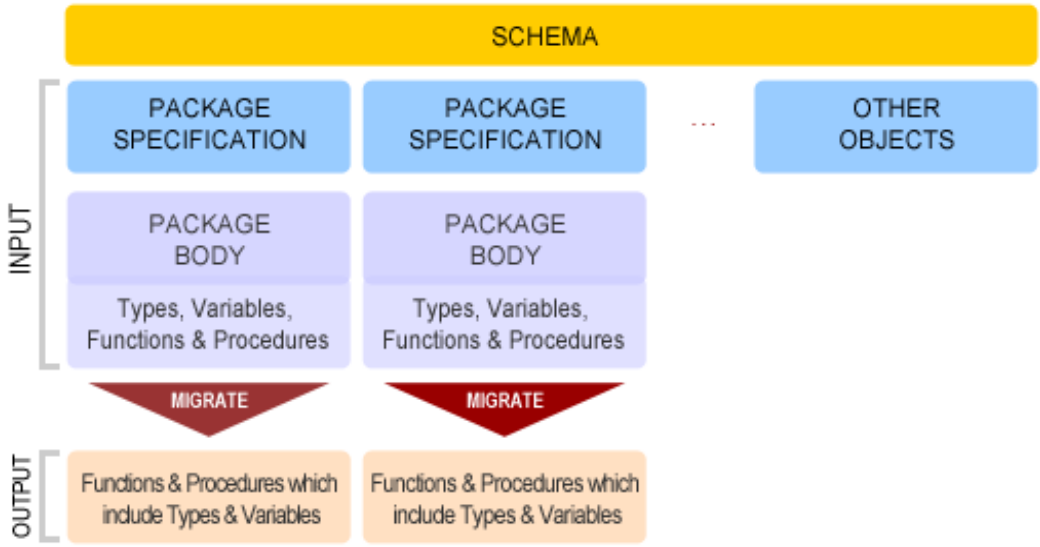
Packages, Package Variables, Package Split, REF CURSOR, VARRAY, Creating a Schema for Package, Grant Execute, Package Name List and Data Type. For details, see **Packages** to **Data Type**.

6.6.17.1 Packages

Packages are schema objects that group logically related PL/SQL types, variables, functions and procedures. In Oracle, each package consists of two parts: package specification and package body. The package specification may contain variables and REF CURSOR in variables. The package REF CURSORs are identified and migrated in the referred places. The functions and procedures in the package body are migrated into individual functions and procedures. The types and variables in the package body are migrated to each of the functions and procedures.

If the schema names of the package specification and package body do not match, then the tool will log a schema name mismatch error to the **DSCError.log** file.

Figure 6-54 Migration of PL/SQL packages



Input - PL/SQL Packages (Package specifications and body)

```
CREATE OR REPLACE PACKAGE BODY pkg_get_empdet
IS
 PROCEDURE get_ename(eno in number,ename out varchar2)
 IS
 BEGIN
```



```
 SELECT ename || ',' || last_name
  INTO ename
    FROM emp
    WHERE empno = eno;

 END  get_ename;

 FUNCTION get_sal(eno in number) return number
 IS
 lsalary number;
 BEGIN
  SELECT salary
   INTO lsalary
      FROM emp
      WHERE empno = eno;
   RETURN lsalary;

  END get_sal;
 END  pkg_get_empdet;
/
```

Output (The output contains separate functions and procedures for each of the functions and procedures in the package body of the input)

```
CREATE
    OR REPLACE PROCEDURE
pkg_get_empdet.get_ename ( eno in number ,ename out varchar2 ) IS
    BEGIN

SELECT

ename || ',' || last_name INTO ename
         FROM

emp
         WHERE

empno = eno ;
         END ;
         /

CREATE
    or REPLACE FUNCTION
pkg_get_empdet.get_sal ( eno in number )
   return number IS lsalary number ;
BEGIN

SELECT

salary INTO lsalary

FROM

emp

WHERE

empno = eno ;

RETURN lsalary ;
         END ;
         /
```

Input - PL/SQL Packages

```
CREATE OR REPLACE VIEW vw_emp_name AS
         Select pkg_get_empdet.get_sal(emp.empno) as empsal from emp;
```

Output


```
CREATE
OR REPLACE VIEW vw_emp_name AS (SELECT

pkg_get_empdet.get_sal (emp.empno) AS empsal
    FROM
        emp)
;

output:
set
package_name_list = 'func' ;
CREATE
OR REPLACE FUNCTION func1 ( i1 INT )
RETURN INT As TYPE r_rthpagat_list IS RECORD ( /* Record
information about cross-border RMB */
business parameters ( rthpagat ) rthpagat_REQUESTID
RMTS_REMITTANCE_PARAM.REQUESTID%TYPE ,rthpagat_PARAMTNAME
RMTS_REMITTANCE_PARAM.PARAMTNAME%TYPE ,rthpagat_PARAMNUM
RMTS_REMITTANCE_PARAM.PARAMNUM%TYPE ,rthpagat_PARAMSTAT
RMTS_REMITTANCE_PARAM.PARAMSTAT%TYPE ,rthpagat_REQTELLERNO
RMTS_REMITTANCE_PARAM.REQTELLERNO%TYPE
,rthpagat_REQUESTTIME RMTS_REMITTANCE_PARAM.REQUESTTIME%TYPE
,rthpagat_HOSTERRNO RMTS_REMITTANCE_PARAM.HOSTERRNO%TYPE ,rthpagat_HOSTERRMSG
RMTS_REMITTANCE_PARAM.HOSTERRMSG%TYPE ,rthpagat_GATBANK
RMTS_REMITTANCE_PARAM.VALUE1%TYPE ,rthpagat_GATEEBANK
RMTS_REMITTANCE_PARAM.VALUE2%TYPE ,rthpagat_TELLER
RMTS_REMITTANCE_PARAM.VALUE3%TYPE ,rthpagat_DATE
RMTS_REMITTANCE_PARAM.VALUE4%TYPE ,rthpagat_BM_GATBANK
RMTS_REMITTANCE_PARAM.VALUE5%TYPE ,rthpagat_BM_GATEEBANK
RMTS_REMITTANCE_PARAM.VALUE6%TYPE ,rthpagat_BM_LMTEL
RMTS_REMITTANCE_PARAM.VALUE7%TYPE ,rthpagat_BM_LMDAT
RMTS_REMITTANCE_PARAM.VALUE8%TYPE ) ;
v1 r_rthpagat_list ;
BEGIN
        END ;
        /
        reset
package_name_list ;
```

Input -Function/Procedure With No Parameter

In case a procedure or function does not have any parameter or argument, put () after procedure or function name while calling the same procedure or function.

```
CREATE OR REPLACE PACKAGE BODY SAD.bas_lookup_misc_pkg IS

 g_pkg_name CONSTANT VARCHAR2(30) := 'bas_lookup_misc_pkg';
 g_func_name VARCHAR2(30);


 FUNCTION func_name
 RETURN VARCHAR2
  IS
   l_func_name VARCHAR2(100);
 BEGIN
   l_func_name := g_pkg_name || '.' || g_func_name;
   RETURN l_func_name;
 END func_name;
 ---------------------------------------------------------------------------
 PROCEDURE insert_fnd_data_change_logs(pi_table_name              IN VARCHAR2,
                                       pi_table_key_columns       IN VARCHAR2,
                                       pi_table_key_values        IN VARCHAR2,
                                       pi_column_name             IN VARCHAR2,
                                       pi_column_change_from_value IN VARCHAR2,
                                       pi_column_change_to_value   IN VARCHAR2,
                                       pi_op_code                 IN NUMBER,
                                       pi_description             IN VARCHAR2,
                                       po_error_msg               OUT VARCHAR2)
 IS
```



```
 BEGIN
   g_func_name := 'insert_fnd_data_change_logs_t';

 EXCEPTION
   WHEN OTHERS THEN
     po_error_msg := 'Others Exception raise in ' || func_name || ',' || SQLERRM;

 END insert_fnd_data_change_logs;
END SAD.bas_lookup_misc_pkg;
/
```

Output

```
CREATE
    OR REPLACE FUNCTION SAD.bas_lookup_misc_pkg#func_name
    RETURN VARCHAR2
 PACKAGE
 IS
 l_func_name VARCHAR2 ( 100 ) ;
    MIG_PV_VAL_DUMMY_G_PKG_NAME VARCHAR2 ( 30 ) := MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE
( 'SAD' ,'bas_lookup_misc_pkg' ,'g_pkg_name' ) ::VARCHAR2 ( 30 ) ;
    MIG_PV_VAL_DUMMY_G_FUNC_NAME VARCHAR2 ( 30 ) := MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE
( 'SAD' ,'bas_lookup_misc_pkg' ,'g_func_name' ) ::VARCHAR2 ( 30 ) ;

BEGIN
    l_func_name := MIG_PV_VAL_DUMMY_G_PKG_NAME || '.' || MIG_PV_VAL_DUMMY_G_FUNC_NAME ;

 MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE
( 'SAD','bas_lookup_misc_pkg','g_pkg_name',MIG_PV_VAL_DUMMY_G_PKG_NAME ) ;
 MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE
( 'SAD','bas_lookup_misc_pkg','g_func_name',MIG_PV_VAL_DUMMY_G_FUNC_NAME ) ;

    RETURN l_func_name ;


END ;

---------------------------------------------------------------------------------------------------
CREATE
    OR REPLACE PROCEDURE SAD.bas_lookup_misc_pkg#insert_fnd_data_change_logs ( pi_table_name IN
VARCHAR2
    ,pi_table_key_columns IN VARCHAR2
    ,pi_table_key_values IN VARCHAR2
    ,pi_column_name IN VARCHAR2
    ,pi_column_change_from_value IN VARCHAR2
    ,pi_column_change_to_value IN VARCHAR2
    ,pi_op_code IN NUMBER
    ,pi_description IN VARCHAR2
    ,po_error_msg OUT VARCHAR2 )
 PACKAGE
 IS

    MIG_PV_VAL_DUMMY_G_FUNC_NAME VARCHAR2 ( 30 ) := MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE
( 'SAD' ,'bas_lookup_misc_pkg' ,'g_func_name' ) ::VARCHAR2 ( 30 ) ;

BEGIN
    MIG_PV_VAL_DUMMY_G_FUNC_NAME := 'insert_fnd_data_change_logs_t' ;

 MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE
( 'SAD','bas_lookup_misc_pkg','g_pkg_name',MIG_PV_VAL_DUMMY_G_PKG_NAME ) ;
 MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE
( 'SAD','bas_lookup_misc_pkg','g_func_name',MIG_PV_VAL_DUMMY_G_FUNC_NAME ) ;


        EXCEPTION
            WHEN OTHERS THEN
            po_error_msg := 'Others Exception raise in ' || SAD.bas_lookup_misc_pkg#func_name() || ',' ||
SQLERRM ;
```



```
END ;
/
```

Input - Package Body with no procedure and functions

In case package body does not have any logic,for example, procedures and functions, DSC needs to remove all code from the same package. The output is basically blank.

```
CREATE OR REPLACE PACKAGE BODY SAD.bas_subtype_pkg IS
BEGIN
  NULL;
END bas_subtype_pkg;
/
```

Input - SUBTYPE

With the SUBTYPE statement, PL/SQL allows you to define your own subtypes or aliases of predefined datatypes, sometimes referred to as abstract datatypes.

```
CREATE OR REPLACE PACKAGE "SAD"."BAS_SUBTYPE_PKG" IS
SUBTYPE CURRENCY IS BAS_PRICE_LIST_T.CURRENCY%TYPE;
END bas_subtype_pkg;
/
CREATE OR REPLACE PACKAGE BODY "SAD"."BAS_SUBTYPE_PKG" IS
BEGIN
  NULL;
END bas_subtype_pkg;
/
--**********************************************************************
CREATE OR REPLACE PACKAGE BODY SAD.bas_lookup_misc_pkg IS
  FUNCTION get_currency(pi_price_type IN NUMBER) RETURN VARCHAR2 IS
    v_currency bas_subtype_pkg.currency;
  BEGIN
    g_func_name := 'get_currency';
    FOR rec_currency IN (SELECT currency FROM sad_price_type_v WHERE price_type_code = pi_price_type)
    LOOP
      v_currency := rec_currency.currency;
    END LOOP;
    RETURN v_currency;
  END get_currency;
 END SAD.bas_lookup_misc_pkg;
 /
```

Output

```
"SAD"."BAS_SUBTYPE_PKG" package will be blank after migration.
--************************************************************


 CREATE OR REPLACE FUNCTION SAD.bas_lookup_misc_pk#get_currency(pi_price_type IN NUMBER)
RETURN VARCHAR2 IS
    v_currency BAS_PRICE_LIST_T.CURRENCY%TYPE;
  BEGIN
    g_func_name := 'get_currency';
    FOR rec_currency IN (SELECT currency FROM sad_price_type_v WHERE price_type_code = pi_price_type)
    LOOP
      v_currency := rec_currency.currency;
    END LOOP;
    RETURN v_currency;
  END ;
 /
```

NOTE

As the SUBTYPE not supported in GaussDB, the SUBTYPE variable used needs to be replaced with the actual type.

Input - sys.dbms_job

The DBMS_JOB package schedules and manages jobs in the job queue.

```
CREATE OR REPLACE PACKAGE BODY "SAD"."EIP_HTM_INTEGRATION_PKG"
IS
  PROCEDURE Greate_import_instruction_job
  IS
    v_jobid NUMBER;
  BEGIN
      IF
bas_lookup_misc_pkg.Exits_run_job('eip_htm_integration_pkg.import_instruction_job') = 'N' THEN
  sys.dbms_job.Submit(job => v_jobid,
                                          what => 'begin
                                                     eip_htm_integration_pkg.import_instruction_job;
                                                   end;',
                                          next_date => SYSDATE);

  COMMIT;
END IF;
---
END greate_import_instruction_job;
END eip_htm_integration_pkg;
```

Output

```
CREATE OR REPLACE PROCEDURE
sad.Eip_htm_integration_pkg#greate_import_instruction_job
IS
  v_jobid NUMBER;
BEGIN
    IF Bas_lookup_misc_pkg#exits_run_job (
          'eip_htm_integration_pkg.import_instruction_job') = 'N' THEN
     dbms_job.Submit(job => v_jobid,
                                          what => 'begin
                                                     eip_htm_integration_pkg.import_instruction_job;
                                                   end;',
                                          next_date => SYSDATE);
      /*  COMMIT;  */
      NULL;
    END IF;
---
END;
```

NOTE

Remove the SYS schema while calling the package.

Input - Procedure/Function variable

The variable declaration of GaussDB(DWS) does not support the NULL constraint. Therefore, you need to comment out the NULL keyword.

```
CREATE OR REPLACE PACKAGE BODY SAD.sad_lookup_contract_pkg IS
 FUNCTION CONTRACT_DISTRIBUTE_STATUS_S2(PI_CONTRACT_NUMBER IN VARCHAR2)
    RETURN VARCHAR2 IS
    L_CONTRACT_DISTRIBUTE_STATUS VARCHAR2(10)   NULL;

  BEGIN
      IF CUR_CONTRACT.CONTRACT_STATUS = 0 THEN
       L_CONTRACT_DISTRIBUTE_STATUS := 'Cancel';
     ELSE
       L_CONTRACT_DISTRIBUTE_STATUS := 'Active';
     END IF;

    RETURN L_CONTRACT_DISTRIBUTE_STATUS;

  EXCEPTION
    WHEN OTHERS THEN
      L_CONTRACT_DISTRIBUTE_STATUS := NULL;
```



```
  END CONTRACT_DISTRIBUTE_STATUS_S2;
END sad_lookup_contract_pkg;
/
```

Output

```
CREATE OR REPLACE FUNCTION sad_lookup_contract_pkg.Contract_distribute_status_s2
( pi_contract_number IN VARCHAR2 )
  RETURN VARCHAR2
IS
  l_contract_distribute_statusvarchar2 ( 10 )
  /* NULL */
  ;
BEGIN
  IF cur_contract.contract_status = 0 THEN
    l_contract_distribute_status := 'Cancel' ;
  ELSE
    l_contract_distribute_status := 'Active' ;
  END IF ;
  RETURN l_contract_distribute_status ;
EXCEPTION
WHEN OTHERS THEN
  l_contract_distribute_status := NULL ;
END ;/
```

Input - Configuration parameter addPackageNameList = true

Hint to access objects from specific schema by system.

```
CREATE OR REPLACE PACKAGE BODY IC_STAGE.PKG_REVN_ARPU
AS
-----------
-----------
END PKG_REVN_ARPU;
/
```

Output

```
SET  package_name_list = 'PKG_REVN_ARPU' ;
--------------
--------------
reset package_name_list ;
```

Input - Configuration parameter addPackageNameList = false

Hint to access objects from specific schema by system.

```
CREATE OR REPLACE PACKAGE BODY IC_STAGE.PKG_REVN_ARPU
AS
-----------
-----------
END PKG_REVN_ARPU;
/
```

Output

```
SET SEARCH_PATH=PKG_REVN_ARPU,PUBLIC;
```

Input -PACKAGE

Hint that procedure and functions belong to a package.

```
CREATE OR REPLACE PACKAGE BODY SAD.sad_lookup_contract_pkg
IS
 FUNCTION CONTRACT_DISTRIBUTE_STATUS_S2(PI_CONTRACT_NUMBER IN VARCHAR2)
    RETURN VARCHAR2 IS
    L_CONTRACT_DISTRIBUTE_STATUS VARCHAR2(10) ;

  BEGIN
      IF CUR_CONTRACT.CONTRACT_STATUS = 0 THEN
```



```
      L_CONTRACT_DISTRIBUTE_STATUS := 'Cancel';
    ELSE
      L_CONTRACT_DISTRIBUTE_STATUS := 'Active';
    END IF;

  RETURN L_CONTRACT_DISTRIBUTE_STATUS;

 EXCEPTION
   WHEN OTHERS THEN
     L_CONTRACT_DISTRIBUTE_STATUS := NULL;

 END CONTRACT_DISTRIBUTE_STATUS_S2;
END sad_lookup_contract_pkg;
/
```

Output

```
CREATE OR REPLACE FUNCTION sad_lookup_contract_pkg.Contract_distribute_status_s2
( pi_contract_number IN VARCHAR2 )
  RETURN VARCHAR2
PACKAGE
IS
  l_contract_distribute_statusvarchar2 ( 10 ) ;
BEGIN
  IF cur_contract.contract_status = 0 THEN
    l_contract_distribute_status := 'Cancel' ;
  ELSE
    l_contract_distribute_status := 'Active' ;
  END IF ;
  RETURN l_contract_distribute_status ;
EXCEPTION
WHEN OTHERS THEN
  l_contract_distribute_status := NULL ;
END ;
/
```

NOTE

You need to put the PACKAGE keyword while creating any procedure and function in front of the IS/AS statement.

Input -Nested Procedure

Creating a procedure inside a procedure is known as a nested procedure. The nested procedure is private and belongs to the parent procedure.

```
CREATE OR REPLACE PROCEDURE refresh_sw_product_amount(pi_stage_id IN NUMBER)
  IS

    v_product_amount          sad_sw_product_amount_t.product_amount%TYPE;
 FUNCTION get_sw_no
 RETURN VARCHAR2
 IS
     v_xh       NUMBER;
   BEGIN
     BEGIN
       SELECT nvl(to_number(substrb(MAX(sw_no), 3, 4)), 0)
         INTO v_xh
         FROM sad.sad_sw_product_amount_t
        WHERE pi_stage_id = pi_stage_id;
     EXCEPTION WHEN OTHERS THEN
       v_xh := 0;
     END;

     RETURN 'SW' || lpad(to_char(v_xh + 1), 4, '0') || 'Y';
   END get_sw_no;

   BEGIN
```



```
    FOR rec_pu IN (SELECT t.*, sh.header_id
                     FROM asms.ht_stages t, asms.ht, sad.sad_distribution_headers_t sh
                    WHERE t.hth = ht.hth
                      AND sh.contract_number = t.hth
                      AND sh.stage_id = t.stage_id
                      AND ht.sw_track_flag = 'Y'
                      AND to_char(t.category_id) IN
                          (SELECT code
                             FROM asms.asms_lookup_values
                            WHERE type_code = 'CATEGORY_ID_EQUIPMENT'
                              AND enabled_flag = 'Y')
                      AND nvl(t.status, '-1') <> '0'
                      AND t.stage_id = pi_stage_id)

    LOOP
      SELECT nvl(SUM(nvl(product_amount, 0)), 0)
        INTO v_product_amount
        FROM sad.sad_products_t sp
       WHERE sp.header_id = rec_pu.header_id
         AND sp.sw_flag = 'Y';


    END LOOP;

END refresh_sw_product_amount;
```

Output

```
CREATE OR REPLACE FUNCTION get_sw_no(pi_stage_id IN NUMBER)
 RETURN VARCHAR2 IS
    v_xh       NUMBER;
  BEGIN
    BEGIN
      SELECT nvl(to_number(substrb(MAX(sw_no), 3, 4)), 0)
        INTO v_xh
        FROM sad.sad_sw_product_amount_t
       WHERE pi_stage_id = pi_stage_id;
    EXCEPTION WHEN OTHERS THEN
      v_xh := 0;
    END;

    RETURN 'SW' || lpad(to_char(v_xh + 1), 4, '0') || 'Y';
END ;
/

 --*****************************************************************************
 CREATE OR REPLACE PROCEDURE refresh_sw_product_amount(pi_stage_id IN NUMBER)
 IS

   v_product_amount          sad_sw_product_amount_t.product_amount%TYPE;

   BEGIN

    FOR rec_pu IN (SELECT t.*, sh.header_id
                     FROM asms.ht_stages t, asms.ht, sad.sad_distribution_headers_t sh
                    WHERE t.hth = ht.hth
                      AND sh.contract_number = t.hth
                      AND sh.stage_id = t.stage_id
                      AND ht.sw_track_flag = 'Y'
                      AND to_char(t.category_id) IN
                          (SELECT code
                             FROM asms.asms_lookup_values
                            WHERE type_code = 'CATEGORY_ID_EQUIPMENT'
                              AND enabled_flag = 'Y')
                      AND nvl(t.status, '-1') <> '0'
                      AND t.stage_id = pi_stage_id)

    LOOP
      SELECT nvl(SUM(nvl(product_amount, 0)), 0)
        INTO v_product_amount
```

```
        FROM sad.sad_products_t sp
       WHERE sp.header_id = rec_pu.header_id
         AND sp.sw_flag = 'Y';


    END LOOP;

END;
/
```

NOTE

When nested procedures/functions are implemented, the package variables in all procedures/functions must be processed.

After migrating sub-procedures/functions, migrate the parent procedure/function.

if pkgSchemaNaming = false

if **pkgSchemaNaming** is set to **false**, PL RECORD migration should not have package name in the type name as its schema.

Input

```
CREATE OR REPLACE PACKAGE BODY SAD.sad_dml_product_pkg IS

 PROCEDURE save_sad_product_line_amount(pi_stage_id           IN NUMBER,
           pi_product_line_code   IN VARCHAR2,
           po_error_msg           OUT VARCHAR2) IS

  TYPE t_line IS RECORD(
 product_line   VARCHAR2(30),
 product_amount NUMBER);
  TYPE tab_line IS TABLE OF t_line INDEX BY BINARY_INTEGER;
  rec_line             tab_line;
  v_product_line_arr   VARCHAR2(5000);
  v_product_line       VARCHAR2(30) ;
  v_count              INTEGER;
  v_start              INTEGER;
  v_pos                INTEGER;

BEGIN
  v_count      := 0;
  v_start      := 1;

   v_product_line_arr := pi_product_line_code;
 LOOP
  v_pos := instr(v_product_line_arr, ',', v_start);
  IF v_pos <= 0
  THEN
 EXIT;
  END IF;
  v_product_line := substr(v_product_line_arr, v_start, v_pos - 1);
  v_count := v_count + 1;
  rec_line(v_count).product_line := v_product_line;
  rec_line(v_count).product_amount := 0;
  v_product_line_arr := substr(v_product_line_arr, v_pos + 1, length(v_product_line_arr));

 END LOOP;

  FOR v_count IN 1 .. rec_line.count
  LOOP
 UPDATE sad_product_line_amount_t spl
    SET spl.product_line_amount = rec_line(v_count).product_amount
  WHERE spl.stage_id = pi_stage_id
    AND spl.product_line_code = rec_line(v_count).product_line;
 IF SQL%NOTFOUND
 THEN
   INSERT INTO sad_product_line_amount_t
       (stage_id, product_line_code, product_line_amount)
```



```
   VALUES (pi_stage_id, rec_line(v_count).product_line, rec_line(v_count).product_amount);
 END IF;
  END LOOP;

 EXCEPTION
   WHEN OTHERS THEN
 po_error_msg := 'Others Exception raise in ' || func_name || ',' || SQLERRM;
 END save_sad_product_line_amount;

END sad_dml_product_pkg;
/
```

Output

```
CREATE  TYPE SAD.sad_dml_product_pkg#t_line AS
 ( product_line VARCHAR2 ( 30 )
   , product_amount NUMBER ) ;

CREATE OR REPLACE PROCEDURE SAD.sad_dml_product_pkg#save_sad_product_line_amount
 ( pi_stage_id IN NUMBER
   , pi_product_line_code IN VARCHAR2
   , po_error_msg OUT VARCHAR2 )
PACKAGE
IS
 TYPE tab_line IS VARRAY ( 10240 ) OF SAD.sad_dml_product_pkg#t_line ;
    rec_line tab_line ;
    v_product_line_arr VARCHAR2 ( 5000 ) ;
    v_product_line VARCHAR2 ( 30 ) ;
    v_count INTEGER ;
    v_start INTEGER ;
    v_pos INTEGER ;
BEGIN
    v_count := 0 ;
    v_start := 1 ;
    v_product_line_arr := pi_product_line_code ;

    LOOP
       v_pos := instr( v_product_line_arr ,',' ,v_start ) ;

  IF v_pos <= 0 THEN
     EXIT ;
  END IF ;

  v_product_line := SUBSTR( v_product_line_arr ,v_start ,v_pos - 1 ) ;
  v_count := v_count + 1 ;
  rec_line ( v_count ).product_line := v_product_line ;
  rec_line ( v_count ).product_amount := 0 ;
  v_product_line_arr := SUBSTR( v_product_line_arr ,v_pos + 1 ,length( v_product_line_arr ) ) ;

    END LOOP ;

    FOR v_count IN 1.. rec_line.count
 LOOP
       UPDATE sad_product_line_amount_t spl
         SET spl.product_line_amount = rec_line ( v_count ).product_amount
        WHERE spl.stage_id = pi_stage_id
         AND spl.product_line_code = rec_line ( v_count ).product_line ;

  IF SQL%NOTFOUND THEN
  INSERT INTO sad_product_line_amount_t
     ( stage_id, product_line_code, product_line_amount )
  VALUES ( pi_stage_id, rec_line ( v_count ).product_line
     , rec_line ( v_count ).product_amount ) ;

  END IF ;

    END LOOP ;

EXCEPTION
   WHEN OTHERS THEN
```



```
    po_error_msg := 'Others Exception raise in ' || func_name || ',' || SQLERRM ;

END ;
/
```

6.6.17.2 Package Variables

Package variables are available in Oracle packages that allow variables to retain all the functions and procedures in the package. DSC uses customized functions to help GaussDB(DWS) support package variables.

NOTE

Prerequisites

- Create and use the **MIG_ORA_EXT** schema.
- Copy the contents of the custom script and execute the script in all target databases for which migration is to be performed. For details, see **Migration Process**.

If there is a space between a schema name and a package name, or either the package specification or body has quotes, the output may not be the same as expected.

Input - Package variables

```
CREATE
    OR REPLACE PACKAGE scott.pkg_adm_util IS un_stand_value long := '`' ;
    defaultdate date := sysdate ;
g_pkgname CONSTANT VARCHAR2 ( 255 ) DEFAULT 'pkg_adm_util' ;
procedure p1 ;
END pkg_adm_util ;
/

CREATE
    OR REPLACE PACKAGE BODY scott.pkg_adm_util AS defaulttime timestamp := systimestamp ;
    PROCEDURE P1 AS BEGIN
        scott.pkg_adm_util.un_stand_value := 'A' ;
        pkg_adm_util.un_stand_value := 'B' ;
    un_stand_value := 'C' ;
DBMS_OUTPUT.PUT_LINE ( pkg_adm_util.defaultdate ) ;
DBMS_OUTPUT.PUT_LINE ( defaulttime ) ;
DBMS_OUTPUT.PUT_LINE ( scott.pkg_adm_util.un_stand_value ) ;
DBMS_OUTPUT.PUT_LINE ( pkg_adm_util.un_stand_value ) ;
DBMS_OUTPUT.PUT_LINE ( un_stand_value ) ;
END ;
END ;
/
```

Output

```
SCHEMA pkg_adm_util
;
BEGIN
INSERT INTO MIG_ORA_EXT.MIG_PKG_VARIABLES
( SCHEMA_NAME ,PACKAGE_NAME ,SPEC_OR_BODY ,VARIABLE_NAME
,VARIABLE_TYPE ,CONSTANT_
I ,DEFAULT_VALUE ,EXPRESSION_I )
VALUES
( UPPER( 'scott' ) ,UPPER( 'pkg_adm_util' ) ,'S' ,UPPER(
'un_stand_value' ) ,UPPE
R( 'TEXT' ) ,false ,'`' ,false ) ;
INSERT INTO MIG_ORA_EXT.MIG_PKG_VARIABLES
( SCHEMA_NAME ,PACKAGE_NAME ,SPEC_OR_BODY ,VARIABLE_NAME
,VARIABLE_TYPE ,CONSTANT_
I ,DEFAULT_VALUE ,EXPRESSION_I )
VALUES
( UPPER( 'scott' ) ,UPPER( 'pkg_adm_util' ) ,'S' ,UPPER(
'defaultdate' ) ,UPPER( '
date' ) ,false ,$q$sysdate$q$ ,true ) ;
```



```
INSERT INTO MIG_ORA_EXT.MIG_PKG_VARIABLES
( SCHEMA_NAME ,PACKAGE_NAME ,SPEC_OR_BODY ,VARIABLE_NAME
,VARIABLE_TYPE ,CONSTANT_
I ,DEFAULT_VALUE ,EXPRESSION_I )
VALUES
( UPPER( 'scott' ) ,UPPER( 'pkg_adm_util' ) ,'S' ,UPPER(
'g_pkgname' ) ,UPPER( 'VA
RCHAR2 ( 255 )' ) ,true ,'pkg_adm_util' ,false ) ;
END ;
/
BEGIN
INSERT INTO MIG_ORA_EXT.MIG_PKG_VARIABLES
( SCHEMA_NAME ,PACKAGE_NAME ,SPEC_OR_BODY ,VARIABLE_NAME
,VARIABLE_TYPE ,CONSTANT_
I ,DEFAULT_VALUE ,EXPRESSION_I )
VALUES
( UPPER( 'scott' ) ,UPPER( 'pkg_adm_util' ) ,'B' ,UPPER(
'defaulttime' ) ,UPPER( '
timestamp' ) ,false ,$q$CURRENT_TIMESTAMP$q$ ,true ) ;
END ;
/
CREATE
OR REPLACE PROCEDURE pkg_adm_util.P1 AS
BEGIN
MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE ( current_schema ( )
,'pkg_adm_util' ,'un_stand_value' ,( 'A' ) ::TEXT ) ;
MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE ( current_schema ( )
,'pkg_adm_util' ,'un_stand_value' ,( 'B' ) ::TEXT ) ;
MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE ( current_schema ( )
,'pkg_adm_util' ,'un_stand_value' ,( 'C' ) ::TEXT ) ;

DBMS_OUTPUT.PUT_LINE ( MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE
( 'scott' ,'pkg_adm_util' ,'defaultdate' ) :: date ) ;
DBMS_OUTPUT.PUT_LINE ( MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE(
'scott' ,'pkg_adm_util' ,'defaulttime' ) :: timestamp ) ;
DBMS_OUTPUT.PUT_LINE ( MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE(
'scott' ,'pkg_adm_util' ,'un_stand_value' ) :: TEXT ) ;
DBMS_OUTPUT.PUT_LINE ( MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE(
'scott' ,'pkg_adm_util' ,'un_stand_value' ) :: TEXT ) ;
DBMS_OUTPUT.PUT_LINE ( MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE(
'scott' ,'pkg_adm_util' ,'un_stand_value' ) :: TEXT ) ;
END ;
/
```

NOTE

If **pkgSchemaNaming** is **true**.

- Oracle supports package variables for multiple schemas. If different schemas have the same package and variable names, such as:
  - schema1.mypackage.myvariable
  - schema2.mypackage.myvariable

  After migration, the schema names will not be used to differentiate the two package variables. Because schema names are ignored, the last data type declaration or operation for [*any_schema*]**.mypackage.myvariable** will overwrite the type and value for **schema1.mypackage.myvariable** and **schema2.mypackage.myvariable**.

Input-Package variable with default value declared in one package by using CONSTANT keyword and used in another package

The global variable declared in the package specification is accessed in the same or another package.

```
PACKAGE "SAD"."BAS_SUBTYPE_PKG" : (Declaring global variable)
-------------------------------------------------
g_header_waiting_split_status CONSTANT VARCHAR2(20) := 'Waiting_Distribute';
```



```
PACKAGE SAD.sad_lookup_stage_pkg: (Used global variable)
-------------------------------------------------
PROCEDURE calc_product_price(pi_contract_no   IN VARCHAR2 DEFAULT NULL,
                             pi_stage_id      IN NUMBER DEFAULT NULL,
                             pi_calc_category IN VARCHAR2 DEFAULT 'all',
                             pi_op_code       IN NUMBER,
                             po_error_msg     OUT VARCHAR2)
 IS

 CURSOR cur_contract IS
      SELECT DISTINCT sdh.contract_number, sdh.stage_id
        FROM sad_distribution_headers_t sdh
       WHERE sdh.status = bas_subtype_pkg.g_header_waiting_split_status
         AND sdh.contract_number = nvl(pi_contract_no, sdh.contract_number)
         AND sdh.stage_id = nvl(pi_stage_id, sdh.stage_id);

 v_ras_flag VARCHAR2 ( 1 ) ;
BEGIN
..
...
END calc_product_price;
/
```

Output

```
PROCEDURE calc_product_price(pi_contract_no   IN VARCHAR2 DEFAULT NULL,
                             pi_stage_id      IN NUMBER DEFAULT NULL,
                             pi_calc_category IN VARCHAR2 DEFAULT 'all',
                             pi_op_code       IN NUMBER,
                             po_error_msg     OUT VARCHAR2)
 IS

 MIG_PV_VAL_DUMMY_G_HEADER_WAITING_SPLIT_STATUS VARCHAR2 ( 20 ) :=
MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE
( 'SAD' ,'bas_subtype_pkg' ,'g_header_waiting_split_status' ) ::VARCHAR2 ( 20 ) ;

 CURSOR cur_contract IS
      SELECT DISTINCT sdh.contract_number, sdh.stage_id
        FROM sad_distribution_headers_t sdh
       WHERE sdh.status = MIG_PV_VAL_DUMMY_G_HEADER_WAITING_SPLIT_STATUS
         AND sdh.contract_number = nvl(pi_contract_no, sdh.contract_number)
         AND sdh.stage_id = nvl(pi_stage_id, sdh.stage_id);

 v_ras_flag VARCHAR2 ( 1 ) ;

BEGIN
..
...
END;
/
```

NOTE

Package variables need to be declared before CURSOR declaration.

Input-Variable of type EXCEPTION

A package variable is a kind of global variable, which can be used in the entire package after being declared once.

```
CREATE OR REPLACE PACKAGE BODY SAD.sad_lookup_stage_pkg IS

  ex_prog_error EXCEPTION;

PROCEDURE assert_null ( pi_value IN VARCHAR2 )
IS
BEGIN
    IF pi_value IS NOT NULL THEN
           RAISE ex_prog_error ;
```



```
   END IF ;

END assert_null;

END SAD.sad_lookup_stage_pkg
/
```

Output

```
CREATE
    OR REPLACE PROCEDURE SAD.sad_lookup_stage_pkg#assert_null
 ( pi_value IN VARCHAR2 )
PACKAGE
IS
  ex_prog_error EXCEPTION;
BEGIN
   IF pi_value IS NOT NULL THEN
         RAISE ex_prog_error ;

   END IF ;

END ;
/
```

NOTE

As GaussDB does not have the software package functions, the package variable needs to be declared in the procedure or function.

Input - If the configuration parameter pkgSchemaNaming is set to false

A package variable is a kind of global variable, which can be used in the entire package after being declared once.

```
CREATE OR REPLACE PACKAGE BODY SAD.bas_lookup_misc_pkg IS

  g_pkg_name CONSTANT VARCHAR2(30) := 'bas_lookup_misc_pkg';
  g_func_name VARCHAR2(30);

  FUNCTION func_name RETURN VARCHAR2 IS
    l_func_name VARCHAR2(100);
  BEGIN
    l_func_name := g_pkg_name || '.' || g_func_name;
    RETURN l_func_name;
  END;
END SAD.bas_lookup_misc_pkg;
/
```

Output

```
INSERT INTO MIG_ORA_EXT.MIG_PKG_VARIABLES (
         PACKAGE_NAME
         ,SPEC_OR_BODY
         ,VARIABLE_NAME
         ,VARIABLE_TYPE
         ,CONSTANT_I
         ,DEFAULT_VALUE
         ,RUNTIME_EXEC_I
    )
    VALUES ( UPPER( 'bas_lookup_misc_pkg' )
    ,'B'
    ,UPPER( 'g_func_name' )
    ,UPPER( 'VARCHAR2(30)' )
    ,FALSE
    ,NULL
    ,FALSE ) ;

END ;
/
```



```
--*********************************************************************
CREATE
    OR REPLACE FUNCTION SAD.bas_lookup_misc_pkg#func_name
    RETURN VARCHAR2
  PACKAGE
  IS
  l_func_name VARCHAR2 ( 100 ) ;
    MIG_PV_VAL_DUMMY_G_PKG_NAME VARCHAR2 ( 30 ) := MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE
( 'SAD' ,'bas_lookup_misc_pkg' ,'g_pkg_name' ) ::VARCHAR2 ( 30 ) ;
    MIG_PV_VAL_DUMMY_G_FUNC_NAME VARCHAR2 ( 30 ) := MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE
( 'SAD' ,'bas_lookup_misc_pkg' ,'g_func_name' ) ::VARCHAR2 ( 30 ) ;

BEGIN
   l_func_name := MIG_PV_VAL_DUMMY_G_PKG_NAME || '.' || MIG_PV_VAL_DUMMY_G_FUNC_NAME ;

 MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE
( 'SAD','bas_lookup_misc_pkg','g_pkg_name',MIG_PV_VAL_DUMMY_G_PKG_NAME ) ;
 MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE
( 'SAD','bas_lookup_misc_pkg','g_func_name',MIG_PV_VAL_DUMMY_G_FUNC_NAME ) ;

   RETURN l_func_name ;


END ;
/
```

NOTE

If the configuration parameter **pkgSchemaNaming** is set to **false**, package variable migration is not happening properly in some places (for example, GET to fetch default value and SET to assign final value are not added). This setting is not recommended by the kernel team. Please check with Kernel team.

Input-Package variable declared with data type as table column %TYPE

If a data type is declared as table column %TYPE for a variable, the data type which is defined on table creation level is considered to be the corresponding column.

```
CREATE OR REPLACE PACKAGE BODY SAD.bas_lookup_misc_pkg IS

 v_emp_name emp.ename%TYPE;

PROCEDURE save_emp_dtls ( v_empno IN VARCHAR2 )
IS
BEGIN

   IF v_emp_name IS NULL THEN
     v_emp_name := 'test';
   END IF ;

END save_emp_dtls;

END bas_lookup_misc_pkg
/
```

Output

```
BEGIN

    INSERT INTO MIG_ORA_EXT.MIG_PKG_VARIABLES (
        PACKAGE_NAME
        ,SPEC_OR_BODY
        ,VARIABLE_NAME
        ,VARIABLE_TYPE
        ,CONSTANT_I
        ,DEFAULT_VALUE
        ,RUNTIME_EXEC_I
    )
```



```
    VALUES ( UPPER( 'bas_lookup_misc_pkg' )
    ,'B'
    ,UPPER( 'v_emp_name' )
    ,UPPER( 'VARCHAR2(30)' )
    ,FALSE
    ,NULL
    ,FALSE ) ;

END ;
/
--*********************************************************
CREATE
     OR REPLACE PROCEDURE SAD.bas_lookup_misc_pkg#save_emp_dtls ( v_empno IN VARCHAR2 )
PACKAGE
IS
  MIG_PV_VAL_DUMMY_EMP_NAME VARCHAR2 ( 30 ) := MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE
( 'SAD' ,'bas_lookup_misc_pkg' ,'v_emp_name' ) ::VARCHAR2 ( 30 ) ;
BEGIN
    IF MIG_PV_VAL_DUMMY_EMP_NAME IS NULL THEN
       MIG_PV_VAL_DUMMY_EMP_NAME := 'test';
    END IF ;

END ;
/
```

NOTE

While migrating a package variable with a data type as table column %TYPE, take the actual data type from a table and use it while declaring a variable, rather than using %TYPE.

Input - If the configuration parameter "pkgSchemaNaming" is set to false

If the PACKAGE name is specified along with the SCHEMA name, use the SCHEMA name on GET() to fetch the default value and SET() to assign the final value .

```
CREATE OR REPLACE PACKAGE BODY SAD.bas_lookup_misc_pkg IS

  g_pkg_name CONSTANT VARCHAR2(30) := 'bas_lookup_misc_pkg';
  g_func_name VARCHAR2(30);

  FUNCTION func_name RETURN VARCHAR2 IS
    l_func_name VARCHAR2(100);
  BEGIN
    l_func_name := g_pkg_name || '.' || g_func_name;
    RETURN l_func_name;
  END;
END SAD.bas_lookup_misc_pkg;
/
```

Output

```
BEGIN
    INSERT INTO MIG_ORA_EXT.MIG_PKG_VARIABLES (
        PACKAGE_NAME
        ,SPEC_OR_BODY
        ,VARIABLE_NAME
        ,VARIABLE_TYPE
        ,CONSTANT_I
        ,DEFAULT_VALUE
        ,RUNTIME_EXEC_I
    )
    VALUES ( UPPER( 'bas_lookup_misc_pkg' )
    ,'B'
    ,UPPER( 'g_pkg_name' )
    ,UPPER( 'VARCHAR2(30)' )
    ,TRUE
    ,'bas_lookup_misc_pkg'
    ,FALSE ) ;
```



```
    INSERT INTO MIG_ORA_EXT.MIG_PKG_VARIABLES (
        PACKAGE_NAME
        ,SPEC_OR_BODY
        ,VARIABLE_NAME
        ,VARIABLE_TYPE
        ,CONSTANT_I
        ,DEFAULT_VALUE
        ,RUNTIME_EXEC_I
    )
    VALUES ( UPPER( 'bas_lookup_misc_pkg' )
    ,'B'
    ,UPPER( 'g_func_name' )
    ,UPPER( 'VARCHAR2(30)' )
    ,FALSE
    ,NULL
    ,FALSE ) ;

END ;
/
--********************************************************************
CREATE
    OR REPLACE FUNCTION SAD.bas_lookup_misc_pkg#func_name
    RETURN VARCHAR2
  PACKAGE
  IS
  l_func_name VARCHAR2 ( 100 ) ;
    MIG_PV_VAL_DUMMY_G_PKG_NAME VARCHAR2 ( 30 ) := MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE
( 'SAD' ,'bas_lookup_misc_pkg' ,'g_pkg_name' ) ::VARCHAR2 ( 30 ) ;
    MIG_PV_VAL_DUMMY_G_FUNC_NAME VARCHAR2 ( 30 ) := MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE
( 'SAD' ,'bas_lookup_misc_pkg' ,'g_func_name' ) ::VARCHAR2 ( 30 ) ;

BEGIN
   l_func_name := MIG_PV_VAL_DUMMY_G_PKG_NAME || '.' || MIG_PV_VAL_DUMMY_G_FUNC_NAME ;

 MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE
( 'SAD','bas_lookup_misc_pkg','g_pkg_name',MIG_PV_VAL_DUMMY_G_PKG_NAME ) ;
 MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE
( 'SAD','bas_lookup_misc_pkg','g_func_name',MIG_PV_VAL_DUMMY_G_FUNC_NAME ) ;

   RETURN l_func_name ;


END ;
/
```

Input - If the configuration parameter pkgSchemaNaming is set to false

If the configuration parameter **pkgSchemaNaming** is set to **false**.

```
CREATE OR REPLACE PACKAGE BODY bas_lookup_misc_pkg IS

  g_pkg_name CONSTANT VARCHAR2(30) := 'bas_lookup_misc_pkg';
  g_func_name VARCHAR2(30);

  FUNCTION func_name RETURN VARCHAR2 IS
    l_func_name VARCHAR2(100);
  BEGIN
    l_func_name := g_pkg_name || '.' || g_func_name;
    RETURN l_func_name;
  END;
END SAD.bas_lookup_misc_pkg;
/
```

Output

```
BEGIN
    INSERT INTO MIG_ORA_EXT.MIG_PKG_VARIABLES (
        PACKAGE_NAME
        ,SPEC_OR_BODY
```

```
        ,VARIABLE_NAME
        ,VARIABLE_TYPE
        ,CONSTANT_I
        ,DEFAULT_VALUE
        ,RUNTIME_EXEC_I
    )
    VALUES ( UPPER( 'bas_lookup_misc_pkg' )
    ,'B'
    ,UPPER( 'g_pkg_name' )
    ,UPPER( 'VARCHAR2(30)' )
    ,TRUE
    ,'bas_lookup_misc_pkg'
    ,FALSE ) ;

    INSERT INTO MIG_ORA_EXT.MIG_PKG_VARIABLES (
        PACKAGE_NAME
        ,SPEC_OR_BODY
        ,VARIABLE_NAME
        ,VARIABLE_TYPE
        ,CONSTANT_I
        ,DEFAULT_VALUE
        ,RUNTIME_EXEC_I
    )
    VALUES ( UPPER( 'bas_lookup_misc_pkg' )
    ,'B'
    ,UPPER( 'g_func_name' )
    ,UPPER( 'VARCHAR2(30)' )
    ,FALSE
    ,NULL
    ,FALSE ) ;

END ;
/
--**********************************************************************
CREATE
    OR REPLACE FUNCTION bas_lookup_misc_pkg#func_name
    RETURN VARCHAR2
  PACKAGE
  IS
  l_func_name VARCHAR2 ( 100 ) ;
    MIG_PV_VAL_DUMMY_G_PKG_NAME VARCHAR2 ( 30 ) := MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE
( CURRENT_SCHEMA() ,'bas_lookup_misc_pkg' ,'g_pkg_name' ) ::VARCHAR2 ( 30 ) ;
    MIG_PV_VAL_DUMMY_G_FUNC_NAME VARCHAR2 ( 30 ) := MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE
( CURRENT_SCHEMA() ,'bas_lookup_misc_pkg' ,'g_func_name' ) ::VARCHAR2 ( 30 ) ;

BEGIN
   l_func_name := MIG_PV_VAL_DUMMY_G_PKG_NAME || '.' || MIG_PV_VAL_DUMMY_G_FUNC_NAME ;

 MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE
( CURRENT_SCHEMA(),'bas_lookup_misc_pkg','g_pkg_name',MIG_PV_VAL_DUMMY_G_PKG_NAME ) ;
 MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE
( CURRENT_SCHEMA(),'bas_lookup_misc_pkg','g_func_name',MIG_PV_VAL_DUMMY_G_FUNC_NAME ) ;

   RETURN l_func_name ;


END ;
/
```

Input: if pkgSchemaNaming is set to false, package variable

The global variable is not correctly converted during package conversion, and an error is reported during compilation. If the configuration parameter **pkgSchemaNaming** is set to **false**, package variable migration is not happening properly in some places. This setting is not recommended by Kernel team. Please check with Kernel team.

```
CREATE OR REPLACE PACKAGE BODY SAD.bas_dml_lookup_pkg IS
  g_pkg_name CONSTANT VARCHAR2(30) := 'bas_dml_ic_price_rule_pkg' ;
```



```
 g_func_name VARCHAR2 (100);

 FUNCTION func_name
 RETURN VARCHAR2
 IS
   l_func_name VARCHAR2(100) ;
 BEGIN
    l_func_name := g_pkg_name || '.' || g_func_name ;
    RETURN l_func_name ;

  END ;

END bas_dml_lookup_pkg ;
/
```

Output

```
BEGIN
    INSERT INTO MIG_ORA_EXT.MIG_PKG_VARIABLES (
          USER_NAME, PACKAGE_NAME, SPEC_OR_BODY
        , VARIABLE_NAME, VARIABLE_TYPE
        , CONSTANT_I, DEFAULT_VALUE, RUNTIME_EXEC_I
    )
    VALUES ( 'SAD', UPPER( 'bas_dml_lookup_pkg' ), 'B'
          , UPPER( 'g_pkg_name' ), UPPER( 'VARCHAR2 ( 30 )' )
          , TRUE, 'bas_dml_ic_price_rule_pkg', FALSE ) ;

    INSERT INTO MIG_ORA_EXT.MIG_PKG_VARIABLES (
          USER_NAME, PACKAGE_NAME, SPEC_OR_BODY
        , VARIABLE_NAME, VARIABLE_TYPE
        , CONSTANT_I, DEFAULT_VALUE, RUNTIME_EXEC_I
    )
    VALUES ( 'SAD', UPPER( 'bas_dml_lookup_pkg' ), 'B'
          , UPPER( 'g_func_name' ), UPPER( 'VARCHAR2(100)' )
          , FALSE, NULL, FALSE ) ;

END ;
/

CREATE OR REPLACE FUNCTION SAD.bas_dml_lookup_pkg#func_name
RETURN VARCHAR2
IS
    MIG_PV_VAL_DUMMY_G_PKG_NAME VARCHAR2(30) := MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE
( 'SAD', 'BAS_DML_LOOKUP_PKG', 'G_PKG_NAME' )::VARCHAR2(30) ;
    MIG_PV_VAL_DUMMY_G_FUNC_NAME VARCHAR2(100) := MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE
( 'SAD', 'BAS_DML_LOOKUP_PKG', 'G_FUNC_NAME' )::VARCHAR2(100) ;
    l_func_name VARCHAR2(100) ;
BEGIN
    l_func_name := MIG_PV_VAL_DUMMY_G_PKG_NAME || '.' || MIG_PV_VAL_DUMMY_G_FUNC_NAME ;
    RETURN l_func_name ;

END ;
/
```

Input: table field type definition in the (%type) table

During package conversion, the schema definition is not added to the table field type definition in the (%type) table. An error is reported during compilation.

```
CREATE TABLE CTP_BRANCH
    ( ID            VARCHAR2(10)
  , NAME          VARCHAR2(100)
  , DESCRIPTION   VARCHAR2(500)
  );

CREATE OR REPLACE PACKAGE BODY SAD.bas_dml_lookup_pkg IS
  g_pkg_name CONSTANT VARCHAR2(30) := 'bas_dml_ic_price_rule_pkg' ;
  g_func_name CTP_BRANCH.NAME%TYPE;

  FUNCTION func_name
```



```
 RETURN VARCHAR2
 IS
   l_func_name VARCHAR2(100) ;
 BEGIN
    l_func_name := g_pkg_name || '.' || g_func_name ;
    RETURN l_func_name ;

  END ;

END bas_dml_lookup_pkg ;
/
```

Output

```
BEGIN
    INSERT INTO MIG_ORA_EXT.MIG_PKG_VARIABLES (
          USER_NAME, PACKAGE_NAME, SPEC_OR_BODY
        , VARIABLE_NAME, VARIABLE_TYPE
        , CONSTANT_I, DEFAULT_VALUE, RUNTIME_EXEC_I
    )
    VALUES ( 'SAD', UPPER( 'bas_dml_lookup_pkg' ), 'B'
          , UPPER( 'g_pkg_name' ), UPPER( 'VARCHAR2 ( 30 )' )
          , TRUE, 'bas_dml_ic_price_rule_pkg', FALSE ) ;

    INSERT INTO MIG_ORA_EXT.MIG_PKG_VARIABLES (
          USER_NAME, PACKAGE_NAME, SPEC_OR_BODY
        , VARIABLE_NAME, VARIABLE_TYPE
        , CONSTANT_I, DEFAULT_VALUE, RUNTIME_EXEC_I
    )
    VALUES ( 'SAD', UPPER( 'bas_dml_lookup_pkg' ), 'B'
          , UPPER( 'g_func_name' ), UPPER( 'VARCHAR2(100)' )
          , FALSE, NULL, FALSE ) ;

END ;
/
CREATE OR REPLACE FUNCTION SAD.bas_dml_lookup_pkg#func_name
RETURN VARCHAR2
IS
    MIG_PV_VAL_DUMMY_G_PKG_NAME VARCHAR2(30) := MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE
( 'SAD', 'BAS_DML_LOOKUP_PKG', 'G_PKG_NAME' )::VARCHAR2(30) ;
    MIG_PV_VAL_DUMMY_G_FUNC_NAME VARCHAR2(100) := MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE
( 'SAD', 'BAS_DML_LOOKUP_PKG', 'G_FUNC_NAME' )::VARCHAR2(100) ;
    l_func_name VARCHAR2(100) ;
BEGIN
    l_func_name := MIG_PV_VAL_DUMMY_G_PKG_NAME || '.' || MIG_PV_VAL_DUMMY_G_FUNC_NAME ;
    RETURN l_func_name ;

END ;
/
```

EXCEPTION

Package variables can be declare as **EXCEPTION**. GaussDB(DWS) does not support this function.

Input

```
CREATE OR REPLACE PACKAGE BODY product_pkg IS

 ex_prog_error EXCEPTION;

 PROCEDURE assert_null(pi_value IN VARCHAR2) IS
 BEGIN
   IF pi_value IS NOT NULL
   THEN
     RAISE ex_prog_error;
   END IF;
 EXCEPTION
   WHEN ex_prog_error THEN
```



```
    RAISE ex_prog_error;

  END assert_null;
END product_pkg;
/
```

Output

```
CREATE OR REPLACE PROCEDURE product_pkg.Assert_null (pi_value IN VARCHAR2)
IS
  ex_prog_error EXCEPTION;
BEGIN
    IF pi_value IS NOT NULL THEN
      RAISE ex_prog_error;
    END IF;
EXCEPTION
  WHEN ex_prog_error THEN
           RAISE ex_prog_error;
END;

/
```

Default value

function is specified as a default value for a package variable.

Input

```
BEGIN
    INSERT INTO MIG_ORA_EXT.MIG_PKG_VARIABLES (
        PACKAGE_NAME
        ,SPEC_OR_BODY
        ,VARIABLE_NAME
        ,VARIABLE_TYPE
        ,CONSTANT_I
        ,DEFAULT_VALUE
        ,RUNTIME_EXEC_I
    )
    VALUES ( UPPER( 'PKG_REVN_ARPU' )
    ,'B'
    ,UPPER( 'imodel' )
    ,UPPER( 'log_table.ds_exec%TYPE' )
    ,FALSE
    ,pkg_etl.proc_set_chain ( 'DAILY ARPU' )
    ,FALSE ) ;

END ;
/
gSQL:PKG_REVN_ARPU_04.SQL:23: ERROR:  function pkg_etl.proc_set_chain(unknown) does not exist
LINE 15:      ,pkg_etl.proc_set_chain ( 'DAILY ARPU' )
              ^
HINT:  No function matches the given name and argument types. You might need to add explicit type casts.



CREATE OR REPLACE PACKAGE BODY IC_STAGE.PKG_REVN_ARPU
AS
 imodel   log_table.ds_exec%TYPE := pkg_etl.proc_set_chain ('DAILY ARPU');
PROCEDURE AGGR_X_AGG00_REVN_DEALER (p_date    PLS_INTEGER,
                                    p_days    PLS_INTEGER)
  AS
    v_start_date   PLS_INTEGER;
    v_curr_date    PLS_INTEGER;
  v_imodel   VARCHAR2(100);
  BEGIN
    pkg_etl.proc_start (p_date, 'AGGR_X_AGG00_REVN_DEALER ');

    v_start_date :=
      TO_CHAR (TO_DATE (p_date, 'yyyymmdd') - (p_days - 1), 'yyyymmdd');
    v_curr_date := p_date;
```



```
  v_imodel := imodel;

  END;
END PKG_REVN_ARPU;
/
```

Output

```
SET
    package_name_list = 'PKG_REVN_ARPU' ;

BEGIN
    INSERT INTO MIG_ORA_EXT.MIG_PKG_VARIABLES (
        PACKAGE_NAME
        ,SPEC_OR_BODY
        ,VARIABLE_NAME
        ,VARIABLE_TYPE
        ,CONSTANT_I
        ,DEFAULT_VALUE
        ,RUNTIME_EXEC_I
    )
    VALUES ( UPPER( 'PKG_REVN_ARPU' )
    ,'B'
    ,UPPER( 'imodel' )
    ,UPPER( 'log_table.ds_exec%TYPE' )
    ,FALSE
    ,$q$pkg_etl.proc_set_chain ('DAILY ARPU')$q$
    ,TRUE ) ;

END ;
/
CREATE
    OR REPLACE PROCEDURE PKG_REVN_ARPU.AGGR_X_AGG00_REVN_DEALER ( p_date INTEGER
    ,p_days INTEGER )
  AS
  MIG_PV_VAL_DUMMY_IMODEL log_table.ds_exec%TYPE := MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE
( CURRENT_USER,'PKG_REVN_ARPU','imodel' ) ::log_table.ds_exec%TYPE ;
    v_start_date INTEGER ;
    v_curr_date INTEGER ;
    v_imodel VARCHAR2 ( 100 ) ;

BEGIN
    pkg_etl.proc_start ( p_date ,'AGGR_X_AGG00_REVN_DEALER ' ) ;
    v_start_date := TO_CHAR( TO_DATE( p_date ,'yyyymmdd' ) - ( p_days - 1 ),'yyyymmdd' ) ;
    v_curr_date := p_date ;
    v_imodel := MIG_PV_VAL_DUMMY_IMODEL ;
    MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE
( CURRENT_USER,'PKG_REVN_ARPU','imodel',MIG_PV_VAL_DUMMY_IMODEL ) ;

END ;
/
reset package_name_list ;
```

PLS_INTEGER

A PLS_INTEGER datatype is not converted into INTEGER for package variables but it is working fine for other local variables. Therefore, it should be converted to INTEGER, such as, varaible1 PLS_INTEGER ==> varaible1 INTEGER

SCRIPTS: SAD_CALC_BPART_PRICE_PKG.sql, SAD_CALC_ITEM_PKG_TEST_OB.sql, SAD_CALC_ITEM_PRICE_TEST_OB.sql, SAD_CALC_ITEM_PRI_TEST_OB.sql, SAD_CALC_ITEM_TEST_OB.sql

Input

```
CREATE OR REPLACE PACKAGE BODY "SAD"."SAD_CALC_BPART_PRICE_PKG" IS
g_max_number_of_entities PLS_INTEGER := 100;
FUNCTION split_warning(pi_contract_number IN VARCHAR2,
```



```
pi_stage_id        IN NUMBER,
pi_quotation_id    IN NUMBER,
pi_cfg_instance_id IN NUMBER) RETURN VARCHAR2 IS
BEGIN
---
l_item_list := items_no_cost(pi_contract_number        => pi_contract_number,
pi_stage_id               => pi_stage_id,
pi_quotation_id           => pi_quotation_id,
pi_cfg_instance_id        => pi_cfg_instance_id,
pi_max_number_of_entities => g_max_number_of_entities,
pi_sep_char               => g_item_sep_char,
po_error_msg              => po_error_msg);
---
END split_warning;
END SAD_CALC_BPART_PRICE_PKG;
```

Output

```
BEGIN
---
INSERT INTO MIG_ORA_EXT.MIG_PKG_VARIABLES (
PACKAGE_NAME
,SPEC_OR_BODY
,VARIABLE_NAME
,VARIABLE_TYPE
,CONSTANT_I
,DEFAULT_VALUE
,RUNTIME_EXEC_I
)
VALUES ( UPPER( 'SAD_CALC_BPART_PRICE_PKG' )
,'B'
,UPPER( 'g_max_number_of_entities' )
,UPPER( 'PLS_INTEGER' )
,FALSE
,100
,FALSE ) ;
---
END;
/
CREATE
OR REPLACE FUNCTION SAD.SAD_CALC_BPART_PRICE_PKG#split_warning ( pi_contract_number IN
VARCHAR2
,pi_stage_id IN NUMBER
,pi_quotation_id IN NUMBER
,pi_cfg_instance_id IN NUMBER )
RETURN VARCHAR2 IS
---
MIG_PV_VAL_DUMMY_G_MAX_NUMBER_OF_ENTITIES PLS_INTEGER :=
MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE ( current_schema ( )
,'SAD_CALC_BPART_PRICE_PKG'
,'g_max_number_of_entities' ) ::PLS_INTEGER ;
---
l_item_list := SAD.SAD_CALC_BPART_PRICE_PKG#items_no_cost ( pi_contract_number =>
pi_contract_number ,
pi_stage_id => pi_stage_id ,
pi_quotation_id => pi_quotation_id ,
pi_cfg_instance_id => pi_cfg_instance_id ,
pi_max_number_of_entities => MIG_PV_VAL_DUMMY_G_MAX_NUMBER_OF_ENTITIES ,
pi_sep_char => MIG_PV_VAL_DUMMY_G_ITEM_SEP_CHAR ,
po_error_msg => po_error_msg ) ;
---
END;
```

Input

```
PLS_INTEGER datatype not converted into INTEGER for package variables but it's working fine for other
local variables therefore for package variables also PLS_INTEGER should be converted to INTEGER datatype
i.e varaible1 PLS_INTEGER ==> varaible1 INTEGER

SCRIPTS : SAD_CALC_BPART_PRICE_PKG.SQL, SAD_CALC_ITEM_PKG_TEST_OB.SQL,
```



```
SAD_CALC_ITEM_PRICE_TEST_OB.SQL, SAD_CALC_ITEM_PRI_TEST_OB.SQL, SAD_CALC_ITEM_TEST_OB.SQL

INPUT :

CREATE OR REPLACE PACKAGE BODY "SAD"."SAD_CALC_BPART_PRICE_PKG" IS

 g_max_number_of_entities PLS_INTEGER := 100;

 FUNCTION split_warning(pi_contract_number IN VARCHAR2,
                        pi_stage_id        IN NUMBER,
                        pi_quotation_id    IN NUMBER,
                        pi_cfg_instance_id IN NUMBER) RETURN VARCHAR2 IS

 BEGIN
 ---

 l_item_list := items_no_cost(pi_contract_number        => pi_contract_number,
                              pi_stage_id              => pi_stage_id,
                              pi_quotation_id          => pi_quotation_id,
                              pi_cfg_instance_id       => pi_cfg_instance_id,
                              pi_max_number_of_entities => g_max_number_of_entities,
                              pi_sep_char              => g_item_sep_char,
                              po_error_msg             => po_error_msg);

 ---

 END split_warning;

END SAD_CALC_BPART_PRICE_PKG;


OUTPUT :

BEGIN

---
INSERT INTO MIG_ORA_EXT.MIG_PKG_VARIABLES (
        PACKAGE_NAME
        ,SPEC_OR_BODY
        ,VARIABLE_NAME
        ,VARIABLE_TYPE
        ,CONSTANT_I
        ,DEFAULT_VALUE
        ,RUNTIME_EXEC_I
    )
    VALUES ( UPPER( 'SAD_CALC_BPART_PRICE_PKG' )
    ,'B'
    ,UPPER( 'g_max_number_of_entities' )
    ,UPPER( 'PLS_INTEGER' )
    ,FALSE
    ,100
    ,FALSE ) ;
---

END;
/

CREATE
    OR REPLACE FUNCTION SAD.SAD_CALC_BPART_PRICE_PKG#split_warning ( pi_contract_number IN
VARCHAR2
    ,pi_stage_id IN NUMBER
    ,pi_quotation_id IN NUMBER
    ,pi_cfg_instance_id IN NUMBER )
    RETURN VARCHAR2 IS

 ---

    MIG_PV_VAL_DUMMY_G_MAX_NUMBER_OF_ENTITIES PLS_INTEGER :=
MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE ( current_schema ( )
```



```
    ,'SAD_CALC_BPART_PRICE_PKG'
    ,'g_max_number_of_entities' ) ::PLS_INTEGER ;

 ---

 l_item_list := SAD.SAD_CALC_BPART_PRICE_PKG#items_no_cost ( pi_contract_number =>
pi_contract_number ,
              pi_stage_id => pi_stage_id ,
              pi_quotation_id => pi_quotation_id ,
              pi_cfg_instance_id => pi_cfg_instance_id ,
              pi_max_number_of_entities => MIG_PV_VAL_DUMMY_G_MAX_NUMBER_OF_ENTITIES ,
              pi_sep_char => MIG_PV_VAL_DUMMY_G_ITEM_SEP_CHAR ,
              po_error_msg => po_error_msg ) ;
 ---

END;
```

Output

```
BEGIN
    INSERT INTO MIG_ORA_EXT.MIG_PKG_VARIABLES
  (  PACKAGE_NAME, SPEC_OR_BODY, VARIABLE_NAME
       , VARIABLE_TYPE, CONSTANT_I, DEFAULT_VALUE
       , RUNTIME_EXEC_I )
    VALUES ( UPPER('SAD_CALC_BPART_PRICE_PKG')
     , 'B', UPPER( 'g_max_number_of_entities' )
     , UPPER( 'INTEGER' ),FALSE,100
     , FALSE ) ;
END ;
/

CREATE OR REPLACE FUNCTION SAD.SAD_CALC_BPART_PRICE_PKG#split_warning
 ( pi_contract_number IN VARCHAR2
   , pi_stage_id   IN NUMBER )
RETURN VARCHAR2
PACKAGE
IS
 MIG_PV_VAL_DUMMY_G_MAX_NUMBER_OF_ENTITIES INTEGER :=
MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE('SAD', 'SAD_CALC_BPART_PRICE_PKG',
'g_max_number_of_entities') ::INTEGER ;
   po_error_msg sad_products_t.exception_description%TYPE ;

BEGIN
    l_item_list := items_no_cost ( pi_contract_number => pi_contract_number ,pi_stage_id => pi_stage_id
       , pi_max_number_of_entities => MIG_PV_VAL_DUMMY_G_MAX_NUMBER_OF_ENTITIES
       , po_error_msg => po_error_msg ) ;
    MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE
('SAD' ,'SAD_CALC_BPART_PRICE_PKG' ,'g_max_number_of_entities' ,MIG_PV_VAL_DUMMY_G_MAX_NUMBER
_OF_ENTITIES);

    RETURN po_error_msg ;

EXCEPTION
   WHEN OTHERS THEN
      po_error_msg := 'Program Others abnormal, Fail to obtain the warning information.' || SQLERRM ;
      MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE
( 'SAD' ,'SAD_CALC_BPART_PRICE_PKG' ,'g_max_number_of_entities' ,MIG_PV_VAL_DUMMY_G_MAX_NUMBE
R_OF_ENTITIES ) ;

      RETURN po_error_msg ;

END ;
/
```

Cursor With Package Variable

The cursor declared in SAD.sad_calc_product_price_pkg#calc_product_price contains package variables and needs to be handled.

Input


```
CREATE OR REPLACE PACKAGE SAD.bas_subtype_pkg IS
  g_header_waiting_split_status CONSTANT VARCHAR2(20) := 'Waiting_Distribute';
  SUBTYPE error_msg IS sad_products_t.exception_description%TYPE;
END bas_subtype_pkg;
/

CREATE OR REPLACE PACKAGE BODY SAD.sad_calc_product_price_pkg IS
  PROCEDURE calc_product_price(pi_contract_no   IN VARCHAR2 DEFAULT NULL,
                               pi_stage_id      IN NUMBER DEFAULT NULL,
                               po_error_msg     OUT VARCHAR2) IS
    CURSOR cur_contract IS
      SELECT DISTINCT sdh.contract_number, sdh.stage_id
        FROM sad_distribution_headers_t sdh
       WHERE sdh.status = bas_subtype_pkg.g_header_waiting_split_status
         AND sdh.contract_number = nvl(pi_contract_no, sdh.contract_number)
         AND sdh.stage_id = nvl(pi_stage_id, sdh.stage_id);

    lv_error_msg bas_subtype_pkg.error_msg;
  BEGIN
    FOR rec_contract IN cur_contract
    LOOP

        validate_process_status(rec_contract.contract_number,
                                rec_contract.stage_id,
                                lv_error_msg);
    END LOOP;

 po_error_msg := lv_error_msg;
  END calc_product_price;

END sad_calc_product_price_pkg;
/
```

Output

```
BEGIN
     INSERT INTO MIG_ORA_EXT.MIG_PKG_VARIABLES
    ( PACKAGE_NAME,SPEC_OR_BODY,VARIABLE_NAME
    , VARIABLE_TYPE,CONSTANT_I,DEFAULT_VALUE
    , RUNTIME_EXEC_I )
     VALUES ( UPPER('bas_subtype_pkg'), 'S', UPPER('g_header_waiting_split_status')
    , UPPER( 'VARCHAR2(20)' ), TRUE, 'Waiting_Distribute'
    , FALSE ) ;
END ;
/

CREATE OR REPLACE PROCEDURE SAD.sad_calc_product_price_pkg#calc_product_price
 ( pi_contract_no IN VARCHAR2 DEFAULT NULL
    , pi_stage_id IN NUMBER DEFAULT NULL
    , po_error_msg OUT VARCHAR2 )
PACKAGE
IS
 MIG_PV_VAL_DUMMY_G_HEADER_WAITING_SPLIT_STATUS VARCHAR2 ( 20 ) :=
MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE ( 'SAD' ,'bas_subtype_pkg'
    ,'g_header_waiting_split_status' ) ::VARCHAR2 ( 20 ) ;

 CURSOR cur_contract IS
 SELECT DISTINCT sdh.contract_number, sdh.stage_id
      FROM sad_distribution_headers_t sdh
     WHERE sdh.status = MIG_PV_VAL_DUMMY_G_HEADER_WAITING_SPLIT_STATUS
       AND sdh.contract_number = nvl( pi_contract_no ,sdh.contract_number )
       AND sdh.stage_id = nvl( pi_stage_id ,sdh.stage_id ) ;

    lv_error_msg sad_products_t.exception_description%TYPE ;
BEGIN
     FOR rec_contract IN cur_contract
  LOOP
         validate_process_status ( rec_contract.contract_number ,rec_contract.stage_id ,lv_error_msg ) ;

     END LOOP ;
```



```
    po_error_msg := lv_error_msg ;
    MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE
( 'SAD' ,'bas_subtype_pkg' ,'g_header_waiting_split_status' ,MIG_PV_VAL_DUMMY_G_HEADER_WAITING_SPLI
T_STATUS ) ;

END ;
/
```

SET VARIABLE function after the RETURN

SET VARIABLE function should be called before the RETURN statements in the procedure and function.

Input

```
CREATE OR REPLACE PACKAGE BODY SAD.bas_dml_lookup_pkg IS
 g_pkg_name CONSTANT VARCHAR2(30) := 'bas_dml_lookup_pkg' ;
 g_func_name VARCHAR2(100);

 FUNCTION func_name
 RETURN VARCHAR2
 IS
   l_func_name VARCHAR2(100) ;
 BEGIN
 g_func_name := 'func_name';
    l_func_name := g_pkg_name || '.' || g_func_name ;
    RETURN l_func_name ;

  END;

 PROCEDURE data_change_logs ( pi_table_name        IN VARCHAR2
                            , pi_table_key_columns IN VARCHAR2
                            , po_error_msg         OUT VARCHAR2
      )
 IS
 BEGIN
   g_func_name := 'data_change_logs';

IF pi_table_name IS NULL
THEN
 RETURN;
END IF;

   INSERT INTO fnd_data_change_logs_t
     ( logid, table_name, table_key_columns )
   VALUES
     ( fnd_data_change_logs_t_s.NEXTVAL
     , pi_table_name, pi_table_key_columns );
 EXCEPTION
   WHEN OTHERS THEN
     po_error_msg := 'Others Exception raise in ' || func_name || ',' || SQLERRM;
 END data_change_logs;

END bas_dml_lookup_pkg;
/
```

Output

```
BEGIN
    INSERT INTO MIG_ORA_EXT.MIG_PKG_VARIABLES
 ( PACKAGE_NAME,SPEC_OR_BODY,VARIABLE_NAME
 , VARIABLE_TYPE,CONSTANT_I,DEFAULT_VALUE
 , RUNTIME_EXEC_I )
    VALUES ( UPPER('bas_dml_lookup_pkg'), 'B', UPPER('g_pkg_name')
   , UPPER( 'VARCHAR2(30)' ), TRUE, 'bas_dml_lookup_pkg'
   , FALSE ) ;

    INSERT INTO MIG_ORA_EXT.MIG_PKG_VARIABLES
 ( PACKAGE_NAME,SPEC_OR_BODY,VARIABLE_NAME
```



```
  , VARIABLE_TYPE,CONSTANT_I,DEFAULT_VALUE
  , RUNTIME_EXEC_I )
  VALUES ( UPPER('bas_dml_lookup_pkg'), 'B', UPPER('g_func_name')
   , UPPER( 'VARCHAR2(100)' ), FALSE, NULL, FALSE ) ;

END ;
/
CREATE OR REPLACE FUNCTION SAD.bas_dml_lookup_pkg#func_name
RETURN VARCHAR2
PACKAGE
IS
 MIG_PV_VAL_DUMMY_G_PKG_NAME VARCHAR2 ( 30 ) := MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE
( 'SAD' ,'bas_dml_lookup_pkg' ,'g_pkg_name' ) ::VARCHAR2 ( 30 ) ;
    MIG_PV_VAL_DUMMY_G_FUNC_NAME VARCHAR2 ( 100 ) :=
MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE ( 'SAD' ,'bas_dml_lookup_pkg' ,'g_func_name' ) ::VARCHAR2
( 100 ) ;
    l_func_name VARCHAR2 ( 100 ) ;

BEGIN
    MIG_PV_VAL_DUMMY_G_FUNC_NAME := 'func_name' ;
    l_func_name := MIG_PV_VAL_DUMMY_G_PKG_NAME || '.' || MIG_PV_VAL_DUMMY_G_FUNC_NAME ;
    MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE
( 'SAD' ,'bas_dml_lookup_pkg' ,'g_func_name' ,MIG_PV_VAL_DUMMY_G_FUNC_NAME ) ;
    MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE
( 'SAD' ,'bas_dml_lookup_pkg' ,'g_pkg_name' ,MIG_PV_VAL_DUMMY_G_PKG_NAME ) ;

    RETURN l_func_name ;
END ;
/

CREATE OR REPLACE PROCEDURE SAD.bas_dml_lookup_pkg#data_change_logs
 ( pi_table_name IN VARCHAR2
   , pi_table_key_columns IN VARCHAR2
   , po_error_msg OUT VARCHAR2 )
PACKAGE
IS
 MIG_PV_VAL_DUMMY_G_FUNC_NAME VARCHAR2 ( 100 ) := MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE
( 'SAD' ,'bas_dml_lookup_pkg' ,'g_func_name' ) ::VARCHAR2 ( 100 ) ;
BEGIN
    MIG_PV_VAL_DUMMY_G_FUNC_NAME := 'data_change_logs' ;

    IF pi_table_name IS NULL THEN
       MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE
( 'SAD' ,'bas_dml_lookup_pkg' ,'g_func_name' ,MIG_PV_VAL_DUMMY_G_FUNC_NAME ) ;
  RETURN ;
    END IF ;

    INSERT INTO fnd_data_change_logs_t ( logid, table_name, table_key_columns )
    VALUES ( NEXTVAL ( 'fnd_data_change_logs_t_s' ), pi_table_name, pi_table_key_columns ) ;

    MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE
( 'SAD' ,'bas_dml_lookup_pkg' ,'g_func_name' ,MIG_PV_VAL_DUMMY_G_FUNC_NAME ) ;

EXCEPTION
   WHEN OTHERS THEN
      po_error_msg := 'Others Exception raise in ' || SAD.bas_dml_lookup_pkg#func_name ( ) || ',' ||
SQLERRM ;
  MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE
( 'SAD' ,'bas_dml_lookup_pkg' ,'g_func_name' ,MIG_PV_VAL_DUMMY_G_FUNC_NAME ) ;
END ;
/
```

Empty package

Empty package bodies do not need to be migrated.

Input

```
CREATE OR REPLACE PACKAGE BODY SAD.bas_subtype_pkg IS
BEGIN
```



```
  NULL;
END bas_subtype_pkg;
/
```

Output will be an empty file.

6.6.17.3 Splitting Packages

The package specification is migrated as a schema named after the package and the procedures and functions in the package body is migrated as **Packagename.procedurename** and **Packagename.funtionname**.

Migration can be performed after **pkgSchemaNaming** is set to **true**.

Input – PACKAGE1.FUNC1

```
CREATE OR REPLACE PACKAGE BODY pack AS
  FUNCTION get_fullname(n_emp_id NUMBER) RETURN VARCHAR2 IS
     v_fullname VARCHAR2(46);
  BEGIN
    SELECT first_name || ',' ||  last_name
    INTO v_fullname
    FROM employees
    WHERE employee_id = n_emp_id;
    RETURN v_fullname;
  END get_fullname;

 PROCEDURE get_salary(n_emp_id NUMBER) RETURN NUMBER IS
    n_salary NUMBER(8,2);
  BEGIN
    SELECT salary
    INTO n_salary
    FROM employees
    WHERE employee_id = n_emp_id;
    END get_salary;
END pack;
/
```

Output

```
CREATE
OR REPLACE FUNCTION pack.get_fullname ( n_emp_id NUMBER )
RETURN VARCHAR2 IS v_fullname VARCHAR2 ( 46 ) ;
BEGIN
        SELECT
                first_name || ',' || last_name INTO v_fullname
            FROM
                employees
            WHERE
                employee_id = n_emp_id ;
            RETURN v_fullname ;
        END ;
/
CREATE
    OR REPLACE FUNCTION pack.get_salary ( n_emp_id NUMBER )
    RETURN NUMBER IS n_salary NUMBER ( 8 ,2 ) ;
BEGIN
        SELECT
                salary INTO n_salary
            FROM
                employees
            WHERE
                employee_id = n_emp_id ;
            RETURN n_salary ;
        END ;
        /
```

If pkgSchemaNaming is set to false, packages can be split.

When **bas_lookup_misc_pkg** is calling **insert_fnd_data_change_logs**, **insert_fnd_data_change_logs** will be not migrated.

Input

```
CREATE OR REPLACE PACKAGE BODY SAD.bas_dml_lookup_pkg IS
  g_pkg_name CONSTANT VARCHAR2(30) := 'bas_dml_ic_price_rule_pkg' ;
  g_func_name VARCHAR2(100);

  FUNCTION func_name
  RETURN VARCHAR2
  IS
    l_func_name VARCHAR2(100) ;
  BEGIN
     l_func_name := g_pkg_name || '.' || g_func_name ;
     RETURN l_func_name ;

   END ;

  PROCEDURE data_change_logs ( pi_table_name        IN VARCHAR2
                             , pi_table_key_columns IN VARCHAR2
                             , po_error_msg         OUT VARCHAR2
       )
  IS
  BEGIN
    g_func_name := 'insert_fnd_data_change_logs_t';

    INSERT INTO fnd_data_change_logs_t
      ( logid, table_name, table_key_columns )
    VALUES
      ( fnd_data_change_logs_t_s.NEXTVAL
      , pi_table_name, pi_table_key_columns );
  EXCEPTION
    WHEN OTHERS THEN
      po_error_msg := 'Others Exception raise in ' || func_name || ',' || SQLERRM;
  END data_change_logs;

END bas_dml_lookup_pkg;
/
```

Output

```
CREATE OR REPLACE FUNCTION SAD.bas_dml_lookup_pkg#func_name
RETURN VARCHAR2
IS
    MIG_PV_VAL_DUMMY_G_PKG_NAME VARCHAR2(30) := MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE
( 'SAD', 'BAS_DML_LOOKUP_PKG', 'G_PKG_NAME' )::VARCHAR2(30) ;
    MIG_PV_VAL_DUMMY_G_FUNC_NAME VARCHAR2(100) := MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE
( 'SAD', 'BAS_DML_LOOKUP_PKG', 'G_FUNC_NAME' )::VARCHAR2(100) ;
    l_func_name VARCHAR2(100) ;
BEGIN
    l_func_name := MIG_PV_VAL_DUMMY_G_PKG_NAME || '.' || MIG_PV_VAL_DUMMY_G_FUNC_NAME ;
    RETURN l_func_name ;

END ;
/
CREATE OR REPLACE PROCEDURE SAD.bas_dml_lookup_pkg#data_change_logs ( pi_table_name IN
VARCHAR2
               , pi_table_key_columns IN VARCHAR2
               , po_error_msg OUT VARCHAR2 )
IS
 MIG_PV_VAL_DUMMY_G_FUNC_NAME VARCHAR2(30) := MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE
( 'SAD' ,'BAS_DML_LOOKUP_PKG' ,'G_FUNC_NAME' )::VARCHAR2(30) ;
BEGIN
    MIG_PV_VAL_DUMMY_G_FUNC_NAME := 'insert_fnd_data_change_logs_t' ;

    INSERT INTO fnd_data_change_logs_t (
        logid,table_name,table_key_columns )
    VALUES ( NEXTVAL ( 'fnd_data_change_logs_t_s' )
```



```
          , pi_table_name, pi_table_key_columns ) ;

    MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE ( 'SAD', 'BAS_DML_LOOKUP_PKG', 'G_FUNC_NAME',
MIG_PV_VAL_DUMMY_G_FUNC_NAME ) ;

    EXCEPTION
      WHEN OTHERS THEN
        po_error_msg := 'Others Exception raise in ' || SAD.bas_dml_lookup_pkg#func_name( ) || ',' ||
SQLERRM ;
        MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE ( 'SAD', 'BAS_DML_LOOKUP_PKG', 'G_FUNC_NAME',
MIG_PV_VAL_DUMMY_G_FUNC_NAME ) ;

END ;
/
```

PACKAGE Keyword

The kernel needs to add the package tag to the functions and stored procedures converted from the package.

Input

```
CREATE OR REPLACE PACKAGE BODY SAD.bas_dml_lookup_pkg IS

 FUNCTION func_name
 RETURN VARCHAR2
 IS
  l_func_name VARCHAR2(100) ;
 BEGIN
    l_func_name := 'bas_dml_lookup_pkg' || '.' || 'func_name' ;
    RETURN l_func_name ;

  END ;

END bas_dml_lookup_pkg ;
/
```

Output

```
CREATE OR REPLACE FUNCTION func_name
RETURN VARCHAR2
PACKAGE
IS
 l_func_name VARCHAR2(100) ;
BEGIN
  l_func_name := 'bas_dml_lookup_pkg' || '.' || 'func_name' ;
  RETURN l_func_name ;

END ;
/
```

6.6.17.4 REF CURSOR

REF Cursor is a data type that can store the database cursor values and is used to return query results. DSC supports migration of REF CURSOR. The example below shows how the DSC migrates **lref_strong_emptyp** (local REF CURSOR) and ref_strong_emptyp (package-level REF CURSOR).

Input - REF CURSOR in PL/SQL Package (Package Specification and Body)

```
# Package specification
CREATE OR REPLACE PACKAGE pkg_refcur
IS
   TYPE ref_variable IS REF CURSOR;
   TYPE ref_strong_emptyp IS REF CURSOR RETURN emp_o%ROWTYPE;
   PROCEDURE p_get_employees ( v_id in INTEGER ,po_results OUT ref_strong_emptyp );
```



```
END pkg_refcur ;
/

# Package body
CREATE OR REPLACE PACKAGE BODY pkg_refcur
IS
   TYPE lref_strong_emptyp IS REF CURSOR RETURN emp_o%ROWTYPE ;
   var_num NUMBER ;

   PROCEDURE p_get_employees ( v_id IN INTEGER, po_results OUT ref_strong_emptyp )
   is
      vemp_rc lref_strong_emptyp ;
   Begin
      OPEN po_results for
      SELECT * FROM emp_o e
       WHERE e.id = v_id;

   EXCEPTION
      WHEN OTHERS THEN
         RAISE;
   END p_get_employees;
END pkg_refcur;
/
```

Output

```
BEGIN
        INSERT INTO MIG_ORA_EXT.MIG_PKG_VARIABLES
( SCHEMA_NAME ,PACKAGE_NAME ,SPEC_OR_BODY ,VARIABLE_NAME ,VARIABLE_TYPE ,CONSTANT_I ,DEFA
ULT_VALUE ,EXPRESSION_I )
    VALUES ( UPPER( current_schema
( ) ) ,UPPER( 'pkg_refcur' ) ,'B' ,UPPER( 'var_num' ) ,UPPER( 'NUMBER' ) ,false ,NULL ,false ) ;
END ;
/

CREATE
    OR REPLACE PROCEDURE pkg_refcur#p_get_employees ( v_id IN INTEGER ,po_results OUT
SYS_REFCURSOR ) is vemp_rc SYS_REFCURSOR ;
    Begin
        OPEN po_results for SELECT
                *
            FROM
                emp_o e
            WHERE
                e.id = v_id ;
                EXCEPTION WHEN OTHERS
            THEN RAISE ;
    END ;
    /
```

6.6.17.5 Creating a Schema for Package

The package declaration is migrated as a schema named after the package. The migration can be performed after **pkgSchemaNaming** is set to **false**.

Input – Create schema for Package

```
CREATE OR REPLACE EDITIONABLE PACKAGE "PACK_DEMO"."PACKAGE_GET_NOVA_INFO" AS

    TYPE  novalistcur  is REF CURSOR;
   PROCEDURE getNovaInfo (
      i_appEnShortName     IN       VARCHAR2,
      o_flag            OUT    VARCHAR2,
      o_errormsg    OUT    VARCHAR2,
      o_novalist      OUT    novalistcur
   );
```

Output


```
/*~~PACKAGE_GET_NOVA_INFO~~*/
CREATE
    SCHEMA PACKAGE_GET_NOVA_INFO
;
```

6.6.18 VARRAY

REF CURSOR is defined as a return parameter.

It can be migrated after **plsqlCollection** is set to **varray**.

Input - VARRAY

```
CREATE
OR REPLACE TYPE TYPE_RMTS_ARRAYTYPE IS TABLE
OF VARCHAR2 (30000);

CREATE OR REPLACE PACKAGE BODY SCMS_STRING_UTILS
As
FUNCTION END_WITH (SRCSTRING VARCHAR2, --Source character string

ENDCHAR VARCHAR2, --End character string

IGNORECASE BOOLEAN --Ignore Case

)
RETURN BOOLEAN IS SRCLEN NUMBER (20) := LENGTH(SRCSTRING);
ENDLEN NUMBER (20) := LENGTH(ENDCHAR);
V_TOKEN_ARRAY TYPE_RMTS_ARRAYTYPE := TYPE_RMTS_ARRAYTYPE ();
V_TOKEN_ARRAY1 TYPE_RMTS_ARRAYTYPE := TYPE_RMTS_ARRAYTYPE ();
I NUMBER (20) := 1;
TMP_CHAR VARCHAR(1);
TMP_CHAR1 VARCHAR(1);
BEGIN
...
 END;
END;
/
```

Output

```
CREATE
OR REPLACE FUNCTION SCMS_STRING_UTILS.END_WITH (SRCSTRING VARCHAR2 /* source character
string */
, ENDCHAR VARCHAR2 /* End character string */
, IGNORECASE BOOLEAN /* Ignore case */
)
RETURN BOOLEAN IS SRCLEN NUMBER (20) := LENGTH(SRCSTRING);
ENDLEN NUMBER (20) := LENGTH(ENDCHAR);
TYPE TYPE_RMTS_ARRAYTYPE IS VARRAY (1024) OF VARCHAR2 (30000);
V_TOKEN_ARRAY TYPE_RMTS_ARRAYTYPE /*:= TYPE_RMTS_ARRAYTYPE()*/
;
V_TOKEN_ARRAY1 TYPE_RMTS_ARRAYTYPE /*:= TYPE_RMTS_ARRAYTYPE()*/
;
I NUMBER (20) := 1;
TMP_CHAR VARCHAR(1);
TMP_CHAR1 VARCHAR(1);
BEGIN
END;
```

6.6.19 Granting Execution Permissions

This feature is used to give privileges to users for specific packages. All the procedures and functions defined in the specific packages will be granted the execution permission.

Input


```
GRANT EXECUTE ON SAD.BAS_LOOKUP_MISC_PKG TO EIP_SAD;
```

Output

```
GRANT EXECUTE ON procedure_name TO EIP_SAD;
GRANT EXECUTE ON function1_name TO EIP_SAD;
```

NOTE

Both procedure _name and function1_name must belong to SAD.BAS_LOOKUP_MISC_PKG.

The execution permission

The last authorization of the package is not converted.

--GRANT

Input

```
Below should be created as 1spec/t603.SQL
CREATE OR REPLACE PACKAGE SAD.bas_dml_lookup_pkg IS
 FUNCTION func_name RETURN VARCHAR2;
 PROCEDURE data_change_logs ( pi_table_name        IN VARCHAR2
                            , pi_table_key_columns IN VARCHAR2
                            , po_error_msg         OUT VARCHAR2
        );
END bas_dml_lookup_pkg;
/
GRANT EXECUTE ON SAD.bas_dml_lookup_pkg TO eip_sad;
============================
Below should be created as 2body/t603.SQL
CREATE OR REPLACE PACKAGE BODY SAD.bas_dml_lookup_pkg IS
  g_pkg_name CONSTANT VARCHAR2(30) := 'bas_dml_ic_price_rule_pkg' ;
  g_func_name VARCHAR2(100);

  FUNCTION func_name
  RETURN VARCHAR2
  IS
    l_func_name VARCHAR2(100) ;
  BEGIN
     l_func_name := g_pkg_name || '.' || g_func_name ;
     RETURN l_func_name ;

   END func_name;

  PROCEDURE data_change_logs ( pi_table_name        IN VARCHAR2
                             , pi_table_key_columns IN VARCHAR2
                             , po_error_msg         OUT VARCHAR2
       )
  IS
  BEGIN
    ...
  END data_change_logs;

END bas_dml_lookup_pkg;
/
```

Output

```
BEGIN
     INSERT INTO MIG_ORA_EXT.MIG_PKG_VARIABLES
    ( PACKAGE_NAME,SPEC_OR_BODY,VARIABLE_NAME
    , VARIABLE_TYPE,CONSTANT_I,DEFAULT_VALUE
    , RUNTIME_EXEC_I )
     VALUES ( UPPER('bas_dml_lookup_pkg'), 'B', UPPER('g_pkg_name')
      , UPPER( 'VARCHAR2(30)' ),TRUE,'bas_dml_ic_price_rule_pkg'
      , FALSE ) ;

     INSERT INTO MIG_ORA_EXT.MIG_PKG_VARIABLES
```



```
   ( PACKAGE_NAME,SPEC_OR_BODY,VARIABLE_NAME
   , VARIABLE_TYPE,CONSTANT_I,DEFAULT_VALUE
   , RUNTIME_EXEC_I )
    VALUES ( UPPER('bas_dml_lookup_pkg'), 'B', UPPER( 'g_func_name' )
  , UPPER( 'VARCHAR2(100)' ),FALSE,NULL
  , FALSE ) ;

END ;
/

CREATE OR REPLACE FUNCTION SAD.bas_dml_lookup_pkg#bas_dml_lookup_pkg#func_name
RETURN VARCHAR2
PACKAGE
IS
    MIG_PV_VAL_DUMMY_G_PKG_NAME VARCHAR2(30) :=
MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE( 'SAD' ,'bas_dml_lookup_pkg' ,'g_pkg_name' )::VARCHAR2(30);
    MIG_PV_VAL_DUMMY_G_FUNC_NAME VARCHAR2(100) :=
MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE( 'SAD' ,'bas_dml_lookup_pkg' ,'g_func_name' )::VARCHAR2(100);
    l_func_name VARCHAR2 ( 100 ) ;

BEGIN
    l_func_name := MIG_PV_VAL_DUMMY_G_PKG_NAME || '.' || MIG_PV_VAL_DUMMY_G_FUNC_NAME ;
    MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE
( 'SAD' ,'bas_dml_lookup_pkg' ,'g_func_name' ,MIG_PV_VAL_DUMMY_G_FUNC_NAME ) ;
    MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE
( 'SAD' ,'bas_dml_lookup_pkg' ,'g_pkg_name' ,MIG_PV_VAL_DUMMY_G_PKG_NAME ) ;

    RETURN l_func_name ;
END ;
/

CREATE OR REPLACE PROCEDURE SAD.bas_dml_lookup_pkg#data_change_logs
  ( pi_table_name IN VARCHAR2
    , pi_table_key_columns IN VARCHAR2
    , po_error_msg OUT VARCHAR2 )
PACKAGE
IS
BEGIN
        ...
END ;
/

GRANT EXECUTE ON FUNCTION SAD.bas_dml_lookup_pkg#bas_dml_lookup_pkg#func_name() TO eip_sad;
GRANT EXECUTE ON FUNCTION SAD.bas_dml_lookup_pkg#data_change_logs(VARCHAR2, VARCHAR2) TO
eip_sad;
```

6.6.20 Package Name List

Enable & Disable

Set package_name_list to bas_lookup_misc_pkg.

Enable and disable the function based on configuration parameters.

Input

```
If this parameter is enabled, the below line should be added before creating package objects.
SET
package_name_list = '<<package name>>';
If it is not enabled, this line should not be added
```

Output

```
If this parameter is enabled, the below line should be added before creating package objects.
SET
package_name_list = '<<package name>>';
If it is not enabled, this line should not be added.
```


6.6.21 Data Type

Subtype

Customized types in the package cannot be converted.

SUBTYPE error_msg IS sad_products_t.exception_description%TYPE;

SUBTYPE AR_FLAG IS SAD_RA_LINES_TI.AR_FLAG%TYPE;

SUBTYPE LOCK_FLAG IS SAD_SHIPMENT_BATCHES_T.LOCK_FLAG%TYPE;

bas_subtype_pkg.error_msg

Input:

```
CREATE OR REPLACE PACKAGE SAD.bas_subtype_pkg IS
 SUBTYPE func_name IS sad_products_t.func_name%TYPE;
END bas_subtype_pkg;
/
CREATE OR REPLACE PACKAGE BODY SAD.bas_subtype_pkg IS
BEGIN
  NULL;
END bas_subtype_pkg;
/
```

Output:

```
CREATE OR REPLACE PACKAGE BODY SAD.bas_dml_lookup_pkg IS
  g_pkg_name CONSTANT VARCHAR2(30) := 'bas_dml_ic_price_rule_pkg' ;
  g_func_name VARCHAR2(100);

  FUNCTION func_name
  RETURN VARCHAR2
  IS
    l_func_name bas_subtype_pkg.func_name;;
  BEGIN
     l_func_name := g_pkg_name || '.' || g_func_name ;
     RETURN l_func_name ;

   END func_name;

END bas_dml_lookup_pkg;
/
```

%ROWTYPE

The procedures/functions in the package contain the **%ROWTYPE** attribute of the **IN**/**OUT** parameter. This is not supported

Scripts: BAS_DML_SERVIECE_PKG.sql, BAS_LOOKUP_MISC_PKG.sql

INPUT

```
CREATE OR REPLACE PACKAGE BODY "SAD"."BAS_DML_SERVIECE_PKG" IS
PROCEDURE save_split_ou(pi_split_ou  IN split_ou%ROWTYPE,
po_error_msg OUT VARCHAR2) IS
---
BEGIN
---
end save_split_ou;
end BAS_DML_SERVIECE_PKG;
```

OUTPUT

```
CREATE
OR REPLACE PROCEDURE SAD.BAS_DML_SERVIECE_PKG#save_split_ou ( pi_split_ou IN split_ou%ROWTYPE
```



```
,po_error_msg OUT VARCHAR2 ) IS MIG_PV_VAL_DUMMY_G_FUNC_NAME VARCHAR2 ( 30 ) :=
MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE ( current_schema ( )
,'BAS_DML_SERVIECE_PKG'
,'g_func_name' ) ::VARCHAR2 ( 30 ) ;
ex_data_error
EXCEPTION ;
ex_prog_error
EXCEPTION ;
---
BEGIN
---
END;
```

Input

```
CREATE OR REPLACE PACKAGE BODY SAD.BAS_DML_SERVIECE_PKG IS
  PROCEDURE save_split_ou(pi_split_ou  IN split_ou%ROWTYPE,
                          po_error_msg OUT VARCHAR2) IS
  BEGIN
    UPDATE split_ou so
       SET so.auto_balance_flag      = pi_split_ou.auto_balance_flag,
           so.balance_start_date     = pi_split_ou.balance_start_date,
           so.balance_source         = pi_split_ou.balance_source
     WHERE so.dept_code = pi_split_ou.dept_code;
  EXCEPTION
    WHEN OTHERS THEN
      po_error_msg := 'Others Exception raise in ' || g_func_name || ',' ||
                      SQLERRM;
  END save_split_ou;
END bas_dml_serviece_pkg;
/
```

Output

```
CREATE TYPE mig_typ_split_ou AS ...;

CREATE OR REPLACE PROCEDURE SAD.BAS_DML_SERVIECE_PKG#save_split_ou
 ( pi_split_ou IN mig_typ_split_ou
    ,po_error_msg OUT VARCHAR2 )
PACKAGE
IS
BEGIN
         UPDATE split_ou so
         SET so.auto_balance_flag = pi_split_ou.auto_balance_flag
             ,so.balance_start_date = pi_split_ou.balance_start_date
             ,so.balance_source = pi_split_ou.balance_source
         WHERE so.dept_code = pi_split_ou.dept_code ;

EXCEPTION
   WHEN OTHERS THEN
       po_error_msg := 'Others Exception raise in ' || g_func_name || ',' || SQLERRM ;
END ;
/
```

6.7 Netezza Syntax Migration

6.7.1 Tables

Distribution Key

DISTRIBUTE ON (column) should be migrated to DISTRIBUTE BY HASH (column).

Netezza Syntax	Syntax After Migration
CREATE TABLE N_AG_AMT_H (AG_NO national character varying(50) not null, AG_CATEG_CD national character varying(12) not null, AMT_TYPE_CD national character varying(12) not null, DATA_START_DT date not null, CCY_CD national character varying(3) not null, DATA_END_DT date) DISTRIBUTE ON (AG_NO, AG_CATEG_CD, AMT_TYPE_CD) ORGANIZE ON (AG_CATEG_CD, AMT_TYPE_CD, DATA_END_DT) ;	CREATE TABLE N_AG_AMT_H (AG_NO national character varying(50) not null, AG_CATEG_CD national character varying(12) not null, AMT_TYPE_CD national character varying(12) not null, DATA_START_DT date not null, CCY_CD national character varying(3) not null, DATA_END_DT date) WITH(ORIENTATION=COLUMN) DISTRIBUTE BY HASH (AG_NO, AG_CATEG_CD, AMT_TYPE_CD) /* ORGANIZE ON (AG_CATEG_CD, AMT_TYPE_CD, DATA_END_DT) */ ;

ORGANIZE ON

ORGANIZE ON will be commented out.

Netezza Syntax	Syntax After Migration
CREATE TABLE N_AG_AMT_H (AG_NO national character varying(50) not null, AG_CATEG_CD national character varying(12) not null, AMT_TYPE_CD national character varying(12) not null, DATA_START_DT date not null, CCY_CD national character varying(3) not null, DATA_END_DT date) DISTRIBUTE ON (AG_NO, AG_CATEG_CD, AMT_TYPE_CD) ORGANIZE ON (AG_CATEG_CD, AMT_TYPE_CD, DATA_END_DT) ;	CREATE TABLE N_AG_AMT_H (AG_NO national character varying(50) not null, AG_CATEG_CD national character varying(12) not null, AMT_TYPE_CD national character varying(12) not null, DATA_START_DT date not null, CCY_CD national character varying(3) not null, DATA_END_DT date) WITH(ORIENTATION=COLUMN) DISTRIBUTE BY HASH (AG_NO, AG_CATEG_CD, AMT_TYPE_CD) /* ORGANIZE ON (AG_CATEG_CD, AMT_TYPE_CD, DATA_END_DT)*/ ;

Large Field Type

The row-store supports BLOB and CLOB. Column storage does not support BLOB, but it supports CLOB.

Netezza Syntax	Syntax After Migration
CREATE TABLE prod (prod_no number(6) not null, prod_name national character varying(32) not null, prod_desc clob, prod_image blob) DISTRIBUTE ON (prod_no, prod_name) ORGANIZE ON (prod_no, prod_name) ;	CREATE TABLE prod (prod_no number(6) not null, prod_name national character varying(32) not null, prod_desc clob, prod_image bytea) WITH(ORIENTATION=COLUMN) DISTRIBUTE BY HASH (prod_no, prod_name) /* ORGANIZE ON (prod_no, prod_name) */ ;

6.7.2 PROCEDURE with RETURNS

PROCEDURE with RETURNS will be modified to FUNCTION with RETURN.

Netezza Syntax	Syntax After Migration

CREATE OR REPLACE PROCEDURE "DWDB"."EDW"."SP_O_HXYW_LNSACCTINFO_H"(CHARACTER VARYING(8))
RETURNS INTEGER
LANGUAGE NZPLSQL AS
BEGIN_PROC
DECLARE
V_PAR_DAY ALIAS for $1;
V_PRCNAME NVARCHAR(50):= 'SP_O_HXYW_LNSACCTINFO_H';
V_CNT INTEGER;
V_STEP_INFO NVARCHAR(500);
D_TIME_START TIMESTAMP:= CURRENT_TIMESTAMP;
O_RETURN INTEGER;
BEGIN
O_RETURN := 0;
 	
--Writes logs and starts the recording process.
CALL SP_LOG_EXEC(V_PRCNAME,V_PAR_DAY,D_TIME_START,0,0,'The process running ',' ');
 	
V_STEP_INFO := '1.Initialization';
BEGIN
--1.1
SELECT COUNT(*) INTO V_CNT FROM information_schema.columns WHERE TABLE_NAME='TMPO_HXYW_LNSACCTINFO_H1';
if V_CNT>0 then
EXECUTE IMMEDIATE 'DROP TABLE TMPO_HXYW_LNSACCTINFO_H1';
end if;
END;
 	
RETURN O_RETURN;
 	
END;
END_PROC;
	CREATE OR REPLACE FUNCTION "EDW"."SP_O_HXYW_LNSACCTINFO_H"(CHARACTER VARYING(8))
	RETURN INTEGER
	AS
	V_PAR_DAY ALIAS for $1;
	V_PRCNAME NCHAR VARYING(50):= 'SP_O_HXYW_LNSACCTINFO_H';
	V_CNT INTEGER;
	V_STEP_INFO NCHAR VARYING(500);
	D_TIME_START TIMESTAMP:= CURRENT_TIMESTAMP;
	O_RETURN INTEGER;
	BEGIN
	O_RETURN := 0;
	
	/* Writes logs and starts the recording process. */
	
	SP_LOG_EXEC(V_PRCNAME,V_PAR_DAY,D_TIME_START,0,0,'The process running',' ');
	
	V_STEP_INFO := '1.Initialization';
	BEGIN
	/* 1.1 */
	SELECT COUNT(*) INTO V_CNT FROM information_schema.columns WHERE TABLE_NAME=lower('TMPO_HXYW_LNSACCTINFO_H1');
	if V_CNT>0 then
	EXECUTE IMMEDIATE 'DROP TABLE TMPO_HXYW_LNSACCTINFO_H1';
	end if;
	END;
	
	RETURN O_RETURN;
	
	END;
	/

Qualifying Language

Migrate the nzplSQL language to the plpgSQL language or delete the language.

Netezza Syntax	Syntax After Migration

CREATE OR REPLACE PROCEDURE "DWDB"."EDW"."SP_O_HXYW_LNSACCTINFO_H"(CHARACTER VARYING(8))
RETURNS INTEGER
LANGUAGE NZPLSQL AS
BEGIN_PROC
DECLARE
V_PAR_DAY ALIAS for $1;
V_PRCNAME NVARCHAR(50):= 'SP_O_HXYW_LNSACCTINFO_H';
V_CNT INTEGER;
V_STEP_INFO NVARCHAR(500);
D_TIME_START TIMESTAMP:= CURRENT_TIMESTAMP;
O_RETURN INTEGER;
BEGIN
O_RETURN := 0;
 	
--Writes logs and starts the recording process.
CALL SP_LOG_EXEC(V_PRCNAME,V_PAR_DAY,D_TIME_START,0,0,'The process running ',' ');
 	
V_STEP_INFO := '1.Initialization';
BEGIN
--1.1
SELECT COUNT(*) INTO V_CNT FROM information_schema.columns WHERE TABLE_NAME='TMPO_HXYW_LNSACCTINFO_H1';
if V_CNT>0 then
EXECUTE IMMEDIATE 'DROP TABLE TMPO_HXYW_LNSACCTINFO_H1';
end if;
END;
 	
RETURN O_RETURN;
 	
END;
END_PROC;
	CREATE OR REPLACE FUNCTION "EDW"."SP_O_HXYW_LNSACCTINFO_H"(CHARACTER VARYING(8))
	RETURN INTEGER
	AS
	V_PAR_DAY ALIAS for $1;
	V_PRCNAME NCHAR VARYING(50):= 'SP_O_HXYW_LNSACCTINFO_H';
	V_CNT INTEGER;
	V_STEP_INFO NCHAR VARYING(500);
	D_TIME_START TIMESTAMP:= CURRENT_TIMESTAMP;
	O_RETURN INTEGER;
	BEGIN
	O_RETURN := 0;
	
	/* Writes logs and starts the recording process. */
	
	SP_LOG_EXEC(V_PRCNAME,V_PAR_DAY,D_TIME_START,0,0,'The process running',' ');
	
	V_STEP_INFO := '1.Initialization';
	BEGIN
	/* 1.1 */
	SELECT COUNT(*) INTO V_CNT FROM information_schema.columns WHERE TABLE_NAME=lower('TMPO_HXYW_LNSACCTINFO_H1');
	if V_CNT>0 then
	EXECUTE IMMEDIATE 'DROP TABLE TMPO_HXYW_LNSACCTINFO_H1';
	end if;
	END;
	
	RETURN O_RETURN;
	
	END;
	/

Process Compilation Specification

The process which is started with **Begin_PROC** and ended with **END_PROC** should be removed.

Netezza Syntax	Syntax After Migration

CREATE OR REPLACE PROCEDURE "DWDB"."EDW"."SP_O_HXYW_LNSACCTINFO_H"(CHARACTER VARYING(8))
RETURNS INTEGER
LANGUAGE NZPLSQL AS
BEGIN_PROC
DECLARE
V_PAR_DAY ALIAS for $1;
V_PRCNAME NVARCHAR(50):= 'SP_O_HXYW_LNSACCTINFO_H';
V_CNT INTEGER;
V_STEP_INFO NVARCHAR(500);
D_TIME_START TIMESTAMP:= CURRENT_TIMESTAMP;
O_RETURN INTEGER;
BEGIN
O_RETURN := 0;
 	
--Writes logs and starts the recording process.
CALL SP_LOG_EXEC(V_PRCNAME,V_PAR_DAY,D_TIME_START,0,0,'The process running ',' ');
 	
V_STEP_INFO := '1.Initialization';
BEGIN
--1.1
SELECT COUNT(*) INTO V_CNT FROM information_schema.columns WHERE TABLE_NAME='TMPO_HXYW_LNSACCTINFO_H1';
if V_CNT>0 then
EXECUTE IMMEDIATE 'DROP TABLE TMPO_HXYW_LNSACCTINFO_H1';
end if;
END;
 	
RETURN O_RETURN;
 	
END;
END_PROC;
	CREATE OR REPLACE FUNCTION "EDW"."SP_O_HXYW_LNSACCTINFO_H"(CHARACTER VARYING(8))
	RETURN INTEGER
	AS
	V_PAR_DAY ALIAS for $1;
	V_PRCNAME NCHAR VARYING(50):= 'SP_O_HXYW_LNSACCTINFO_H';
	V_CNT INTEGER;
	V_STEP_INFO NCHAR VARYING(500);
	D_TIME_START TIMESTAMP:= CURRENT_TIMESTAMP;
	O_RETURN INTEGER;
	BEGIN
	O_RETURN := 0;
	
	/* Writes logs and starts the recording process. */
	
	SP_LOG_EXEC(V_PRCNAME,V_PAR_DAY,D_TIME_START,0,0,'The process running',' ');
	
	V_STEP_INFO := '1.Initialization';
	BEGIN
	/* 1.1 */
	SELECT COUNT(*) INTO V_CNT FROM information_schema.columns WHERE TABLE_NAME=lower('TMPO_HXYW_LNSACCTINFO_H1');
	if V_CNT>0 then
	EXECUTE IMMEDIATE 'DROP TABLE TMPO_HXYW_LNSACCTINFO_H1';
	end if;
	END;
	
	RETURN O_RETURN;
	
	END;
	/

DECLARE Keyword to Declare the Local Variables

DECLARE should be modified to AS.

Netezza Syntax	Syntax After Migration

CREATE OR REPLACE PROCEDURE "DWDB"."EDW"."SP_O_HXYW_LNSACCTINFO_H"(CHARACTER VARYING(8))
RETURNS INTEGER
LANGUAGE NZPLSQL AS
BEGIN_PROC
DECLARE
V_PAR_DAY ALIAS for $1;
V_PRCNAME NVARCHAR(50):= 'SP_O_HXYW_LNSACCTINFO_H';
V_CNT INTEGER;
V_STEP_INFO NVARCHAR(500);
D_TIME_START TIMESTAMP:= CURRENT_TIMESTAMP;
O_RETURN INTEGER;
BEGIN
O_RETURN := 0;
 	
--Writes logs and starts the recording process.
CALL SP_LOG_EXEC(V_PRCNAME,V_PAR_DAY,D_TIME_START,0,0,'The process running ',' ');
 	
V_STEP_INFO := '1.Initialization';
BEGIN
--1.1
SELECT COUNT(*) INTO V_CNT FROM information_schema.columns WHERE TABLE_NAME='TMPO_HXYW_LNSACCTINFO_H1';
if V_CNT>0 then
EXECUTE IMMEDIATE 'DROP TABLE TMPO_HXYW_LNSACCTINFO_H1';
end if;
END;
 	
RETURN O_RETURN;
 	
END;
END_PROC;
	CREATE OR REPLACE FUNCTION "EDW"."SP_O_HXYW_LNSACCTINFO_H"(CHARACTER VARYING(8))
	RETURN INTEGER
	AS
	V_PAR_DAY ALIAS for $1;
	V_PRCNAME NCHAR VARYING(50):= 'SP_O_HXYW_LNSACCTINFO_H';
	V_CNT INTEGER;
	V_STEP_INFO NCHAR VARYING(500);
	D_TIME_START TIMESTAMP:= CURRENT_TIMESTAMP;
	O_RETURN INTEGER;
	BEGIN
	O_RETURN := 0;
	
	/* Writes logs and starts the recording process. */
	
	SP_LOG_EXEC(V_PRCNAME,V_PAR_DAY,D_TIME_START,0,0,'The process running',' ');
	
	V_STEP_INFO := '1.Initialization';
	BEGIN
	/* 1.1 */
	SELECT COUNT(*) INTO V_CNT FROM information_schema.columns WHERE TABLE_NAME=lower('TMPO_HXYW_LNSACCTINFO_H1');
	if V_CNT>0 then
	EXECUTE IMMEDIATE 'DROP TABLE TMPO_HXYW_LNSACCTINFO_H1';
	end if;
	END;
	
	RETURN O_RETURN;
	
	END;
	/

6.7.3 Procedure

Variable Data Type

NVARCHAR changed to NCHAR VARING.

Netezza Syntax	Syntax After Migration

CREATE OR REPLACE PROCEDURE
"NTZDB"."EDW"."SP_NTZ_NVARCHAR"
(CHARACTER VARYING(8))
RETURNS INTEGER
LANGUAGE NZPLSQL AS
BEGIN_PROC
DECLARE
V_PAR_DAY ALIAS for $1;
V_PRCNAME NVARCHAR(50):=
'SP_O_HXYW_LNSACCTINFO_H';
V_CNT INTEGER;
V_STEP_INFO NVARCHAR(500);
D_TIME_START TIMESTAMP:=
CURRENT_TIMESTAMP;
O_RETURN INTEGER;
BEGIN
O_RETURN := 0;
 	
--Writes logs and starts the recording process.
CALL
SP_LOG_EXEC(V_PRCNAME,V_PAR_DAY,D_TIME_ST
ART,0,0,'The process running ',' ');
 	
V_STEP_INFO := '1.Initialization';
 	
RETURN O_RETURN;
 	
END;
END_PROC;
	CREATE OR REPLACE FUNCTION
	"EDW"."SP_NTZ_NVARCHAR"
	(CHARACTER VARYING(8))
	RETURN INTEGER
	AS
	V_PAR_DAY ALIAS for $1;
	V_PRCNAME NCHAR VARYING(50):=
	'SP_O_HXYW_LNSACCTINFO_H';
	V_CNT INTEGER;
	V_STEP_INFO NCHAR VARYING(500);
	D_TIME_START TIMESTAMP:=
	CURRENT_TIMESTAMP;
	O_RETURN INTEGER;
	BEGIN
	O_RETURN := 0;
	
	/* Writes logs and starts the recording process. */
	
	SP_LOG_EXEC(V_PRCNAME,V_PAR_DAY,D_TIME_ST
	ART,0,0,'The process running',' ');
	
	V_STEP_INFO := '1.Initialization';
	
	RETURN O_RETURN;
	
	END;
	/

row counts

The **row_count** function is supported for affected row counting.

Netezza Syntax	Syntax After Migration
CREATE OR REPLACE PROCEDURE "NTZDB"."EDW"."SP_NTZ_ROWCOUNT" (CHARACTER VARYING(8)) RETURNS INTEGER LANGUAGE NZPLSQL AS BEGIN_PROC DECLARE V_PAR_DAY ALIAS for $1; O_RETURN INTEGER; BEGIN O_RETURN := 0; EXECUTE IMMEDIATE 'INSERT INTO TMPO_HXYW_LNSACCTINFO_H1 (ACCTNO, ACCTYPE, SUBCTRLCODE, CCY, NAME) SELECT ACCTNO, ACCTYPE, SUBCTRLCODE, CCY, NAME FROM O_HXYW_LNSACCTINFO T WHERE NOT EXISTS (SELECT 1 FROM O_HXYW_LNSACCT T1 WHERE T1.DATA_START_DT<='''||V_PAR_DAY||''' AND T.MD5_VAL=T1.MD5_VAL)'; O_RETURN := ROW_COUNT; RETURN O_RETURN; END; END_PROC;	CREATE OR REPLACE FUNCTION "EDW"."SP_NTZ_ROWCOUNT" (CHARACTER VARYING(8)) RETURN INTEGER AS V_PAR_DAY ALIAS for $1; O_RETURN INTEGER; BEGIN O_RETURN := 0; EXECUTE IMMEDIATE 'INSERT INTO TMPO_HXYW_LNSACCTINFO_H1 (ACCTNO, ACCTYPE, SUBCTRLCODE, CCY, NAME) SELECT ACCTNO, ACCTYPE, SUBCTRLCODE, CCY, NAME FROM O_HXYW_LNSACCTINFO T WHERE NOT EXISTS (SELECT 1 FROM O_HXYW_LNSACCT T1 WHERE T1.DATA_START_DT<='''||V_PAR_DAY||''' AND T.MD5_VAL=T1.MD5_VAL)'; O_RETURN := SQL%ROWCOUNT; RETURN O_RETURN; END; /

NOTE

ROW_COUNT identifies the number of rows associated with the previous SQL statement. If the previous SQL statement is a DELETE, INSERT, or UPDATE statement, ROW_COUNT identifies the number of rows that qualified for the operation.

System Tables

System **tables _V_SYS_COLUMNS** is replaced with **information_schema.columns**.

Netezza Syntax	Syntax After Migration
BEGIN SELECT COUNT(*) INTO V_CNT FROM _V_SYS_COLUMNS WHERE table_schem = 'SCOTT' AND TABLE_NAME='TMPO_HXYW_LNSACCTINFO_H1'; if V_CNT>0 then EXECUTE IMMEDIATE 'DROP TABLE TMPO_HXYW_LNSACCTINFO_H1'; end if; END;	BEGIN SELECT COUNT(*) INTO V_CNT FROM information_schema.columns WHERE table_schema = lower('SCOTT') AND table_name = lower('TMPO_HXYW_LNSACCTINFO_H1'); if V_CNT>0 then EXECUTE IMMEDIATE 'DROP TABLE TMPO_HXYW_LNSACCTINFO_H1'; end if; END;

NOTE

Column mapping:

- table_schem => table_schema
- table_name => table_name
- column_name => column_name
- ordinal_position => ordinal_position
- type_name => data_type
- is_nullable => is_nullable

For date subtraction, the corresponding Integer should be returned

Return value should be integer for date subtraction.

Netezza Syntax	Syntax After Migration
SELECT CAST(T1.Buyback_Mature_Dt - CAST('${gsTXDate}' AS DATE) AS CHAR(5)) FROM tab1 T1 WHERE T1.col1 > 10; ----- SELECT CURRENT_DATE - DATE '2019-03-30';	SELECT CAST(EXTRACT('DAY' FROM (T1.Buyback_Mature_Dt - CAST('${gsTXDate}' AS DATE))) AS CHAR(5)) FROM tab1 T1 WHERE T1.col1 > 10; ------- SELECT EXTRACT('DAY' FROM (CURRENT_DATE - CAST('2019-03-30' AS DATE)));

Support of TRANSLATE Function

The SQL TRANSLATE() function replaces a sequence of characters in a string with another sequence of characters. The function replaces a single character at a time.

Netezza Syntax	Syntax After Migration
TRANSLATE(param1) TRANSLATE(1st param, 2nd param, 3rd param) TRANSLATE(1st param, 2nd param, 3rd param, 4th param)	UPPER(param1) TRANSLATE(1st param, 3rd param, RPAD(2nd param, LENGTH(3rd param), ' ')) TRANSLATE(1st param, 3rd param, RPAD(2nd param, LENGTH(3rd param), 4th param))

NOTE

If it contains a single parameter, just execute the UPPER.

UPPER(param1)

If it contains two parameters, throw error.

If it contains three parameters, TRANSLATE(1st param, 3rd param, RPAD(2nd param, LENGTH(3rd param), ' ')).

If it contains four parameters, TRANSLATE(1st param, 3rd param, RPAD(2nd param, LENGTH(3rd param), 4th param)).

Data Type

NATIONAL CHARACTER VARYING (ANY)

Netezza Syntax	Syntax After Migration

CREATE OR REPLACE PROCEDURE
sp_ntz_nchar_with_any
(NATIONAL CHARACTER VARYING(10)
, NATIONAL CHARACTER VARYING(ANY))
RETURN NATIONAL CHARACTER VARYING(ANY)
LANGUAGE NZPLSQL AS
BEGIN_PROC
DECLARE
I_LOAD_DT ALIAS FOR $1 ;
-- ETL Date
V_TASK_ID ALIAS FOR $2 ;
BEGIN
RETURN I_LOAD_DT || ',' || V_TASK_ID;
END;
END_PROC;
	CREATE OR REPLACE FUNCTION
	sp_ntz_nchar_with_any
	(NATIONAL CHARACTER VARYING(10)
	, NATIONAL CHARACTER VARYING)
	RETURN NATIONAL CHARACTER VARYING
	AS
	I_LOAD_DT ALIAS FOR $1 ;
	/* ETL Date */
	V_TASK_ID ALIAS FOR $2 ;
	BEGIN
	RETURN I_LOAD_DT || ',' || V_TASK_ID;
	END;
	/

CHARACTER VARYING (ANY)

Netezza Syntax	Syntax After Migration

CREATE OR REPLACE PROCEDURE
sp_ntz_char_with_any
(NATIONAL CHARACTER VARYING(10)
, CHARACTER VARYING(ANY))
RETURN CHARACTER VARYING(ANY)
LANGUAGE NZPLSQL AS
BEGIN_PROC
DECLARE
I_LOAD_DT ALIAS FOR $1 ;
-- ETL Date
V_TASK_ID ALIAS FOR $2 ;
BEGIN
RETURN I_LOAD_DT || ',' || V_TASK_ID;
END;
END_PROC;
	CREATE OR REPLACE FUNCTION
	sp_ntz_char_with_any
	(NATIONAL CHARACTER VARYING(10)
	, CHARACTER VARYING)
	RETURN CHARACTER VARYING
	AS
	I_LOAD_DT ALIAS FOR $1 ;
	/* ETL Date */
	V_TASK_ID ALIAS FOR $2 ;
	BEGIN
	RETURN I_LOAD_DT || ',' || V_TASK_ID;
	END;
	/

Numeric (ANY)

Netezza Syntax	Syntax After Migration

CREATE or replace PROCEDURE sp_ntz_numeric_with_any
(NUMERIC(ANY)
, NUMERIC(ANY))
RETURNS NATIONAL CHARACTER VARYING(ANY)
LANGUAGE NZPLSQL
AS BEGIN_PROC
DECLARE
ERROR_INFO NVARCHAR(2000) := '';
V_VC_YCBZ NVARCHAR(1) := 'N';
V_VC_SUCCESS NVARCHAR(10) := 'SUCCESS';
 	
p_l_begindate ALIAS FOR $1;
p_l_enddate ALIAS FOR $2;
BEGIN
ERROR_INFO := CRHSP_CRH_ETL_EXCHDATE(p_l_begindate,p_l_enddate);
if ERROR_INFO != V_VC_SUCCESS then
V_VC_YCBZ := 'C';
end if;
 	
RETURN V_VC_SUCCESS;
 	
END;
END_PROC;
	CREATE or replace FUNCTION sp_ntz_numeric_with_any
	(NUMERIC
	, NUMERIC)
	RETURN NATIONAL CHARACTER VARYING
	AS
	ERROR_INFO NCHAR VARYING(2000) := '';
	V_VC_YCBZ NCHAR VARYING(1) := 'N';
	V_VC_SUCCESS NCHAR VARYING(10) := 'SUCCESS';
	
	p_l_begindate ALIAS FOR $1;
	p_l_enddate ALIAS FOR $2;
	BEGIN
	ERROR_INFO := CRHSP_CRH_ETL_EXCHDATE(p_l_begindate,p_l_enddate);
	if ERROR_INFO != V_VC_SUCCESS then
	V_VC_YCBZ := 'C';
	end if;
	
	RETURN V_VC_SUCCESS;
	
	END;
	/

Exception

TRANSACTION_ABORTED

Netezza Syntax	Syntax After Migration

CREATE OR REPLACE PROCEDURE sp_ntz_transaction_aborted
(NUMERIC(ANY)
, NUMERIC(ANY))
RETURNS NATIONAL CHARACTER VARYING(ANY)
LANGUAGE NZPLSQL
AS BEGIN_PROC
DECLARE
ERROR_INFO NVARCHAR(2000) := '';
 	
p_l_begindate ALIAS FOR $1;
p_l_enddate ALIAS FOR $2;
BEGIN
ERROR_INFO := CRHSP_CRH_ETL_EXCHDATE(p_l_begindate,p_l_enddate);
RETURN ERROR_INFO;
 	
EXCEPTION
WHEN TRANSACTION_ABORTED THEN
ROLLBACK;
BEGIN
ERROR_INFO := SQLERRM||' sp_o_transaction_aborted:';
RETURN ERROR_INFO;
END;
 	
WHEN OTHERS THEN
BEGIN
ERROR_INFO := SQLERRM||' sp_o_transaction_aborted:';
RETURN ERROR_INFO;
END;
END;
END_PROC;
	CREATE OR REPLACE FUNCTION sp_ntz_transaction_aborted
	(NUMERIC
	, NUMERIC)
	RETURN NATIONAL CHARACTER VARYING
	AS
	ERROR_INFO NCHAR VARYING(2000) := '';
	
	p_l_begindate ALIAS FOR $1;
	p_l_enddate ALIAS FOR $2;
	BEGIN
	ERROR_INFO := CRHSP_CRH_ETL_EXCHDATE(p_l_begindate,p_l_enddate);
	RETURN ERROR_INFO;
	
	EXCEPTION
	WHEN INVALID_TRANSACTION_TERMINATION THEN
	ROLLBACK;
	BEGIN
	ERROR_INFO := SQLERRM||' sp_o_transaction_aborted:';
	RETURN ERROR_INFO;
	END;
	
	WHEN OTHERS THEN
	BEGIN
	ERROR_INFO := SQLERRM||' sp_o_transaction_aborted:';
	RETURN ERROR_INFO;
	END;
	END;
	/

END statement is specified without semicolon (;)

END statement specified without semicolon (;) is migrated as follows:

END /

Netezza Syntax	Syntax After Migration

CREATE OR REPLACE PROCEDURE
sp_ntz_end_wo_semicolon
(NATIONAL CHARACTER VARYING(10))
RETURNS CHARACTER VARYING(ANY)
LANGUAGE NZPLSQL
AS
BEGIN_PROC
DECLARE
v_B64 Varchar(64) := 'ABCDEFGHIJKLMNOPQR-
STUVWXYZabcdefghijklmnopqrstuvw-
xyz0123456789+/';
v_I int := 0;
v_J int := 0;
v_K int := 0;
v_N int := 0;
v_out Numeric(38,0) := 0;
I_LOAD_DT ALIAS FOR $1;
 	
BEGIN
v_N:=Length(v_B64);
FOR v_I In Reverse 1..Length(IN_base64)
LOOP
v_J:=Instr(v_B64,Substr(IN_base64,v_I,1))-1;
If v_J <0 Then
RETURN -1;
End If;
V_Out:=V_Out+v_J*(v_N**v_K);
v_K:=v_K+1;
END LOOP;
RETURN V_Out;
END
END_PROC;
	CREATE OR REPLACE FUNCTION
	sp_ntz_end_wo_semicolon
	(NATIONAL CHARACTER VARYING(10))
	RETURN CHARACTER VARYING
	AS
	v_B64 Varchar(64) := 'ABCDEFGHIJKLMNOPQR-
	STUVWXYZabcdefghijklmnopqrstuvw-
	xyz0123456789+/';
	v_I int := 0;
	v_J int := 0;
	v_K int := 0;
	v_N int := 0;
	v_out Numeric(38,0) := 0;
	I_LOAD_DT ALIAS FOR $1;
	
	BEGIN
	v_N:=Length(v_B64);
	FOR v_I In Reverse 1..Length(IN_base64)
	LOOP
	v_J:=Instr(v_B64,Substr(IN_base64,v_I,1))-1;
	If v_J <0 Then
	RETURN -1;
	End If;
	V_Out:=V_Out+v_J*(v_N**v_K);
	v_K:=v_K+1;
	END LOOP;
	RETURN V_Out;
	END;
	/

LOOP

Netezza Syntax	Syntax After Migration

CREATE OR REPLACE PROCEDURE
sp_ntz_for_loop_with_more_dots
(INTEGER)
RETURNS CHARACTER VARYING(ANY)
LANGUAGE NZPLSQL
AS BEGIN_PROC
DECLARE p_abc INTEGER;
p_bcd INTEGER;
p_var1 ALIAS FOR $1;
BEGIN
p_bcd := ISNULL(p_var1, 10);
RAISE NOTICE 'p_bcd=%', p_bcd;
 	
FOR p_abc IN 0...(p_bcd)
LOOP
RAISE NOTICE 'hello world %', p_abc;
END LOOP;
 	
END;
END_PROC;
	CREATE OR REPLACE FUNCTION
	sp_ntz_for_loop_with_more_dots
	(INTEGER)
	RETURN CHARACTER VARYING
	AS
	p_abc INTEGER ;
	p_bcd INTEGER;
	p_var1 ALIAS FOR $1;
	
	BEGIN
	p_bcd := NVL(p_var1, 10);
	
	RAISE NOTICE 'p_bcd=%', p_bcd;
	
	FOR p_abc IN 0..(p_bcd)
	LOOP
	RAISE NOTICE 'hello world %', p_abc;
	
	END LOOP;
	
	END;
	/

GaussDB(DWS) Keywords

CURSOR

Netezza Syntax	Syntax After Migration

CREATE OR REPLACE PROCEDURE sp_ntz_keyword_cursor()
RETURNS INTEGER
LANGUAGE NZPLSQL
AS BEGIN_PROC
DECLARE
tablename NVARCHAR(100);
cursor RECORD;
BEGIN
FOR cursor IN SELECT t.TABLENAME FROM _V_TABLE t
WHERE TABLENAME LIKE 'T_ODS_CRM%'
LOOP
tablename := cursor.TABLENAME;
END LOOP;
END;
END_PROC;
	CREATE OR REPLACE FUNCTION sp_ntz_keyword_cursor()
	RETURN INTEGER
	AS
	tablename NCHAR VARYING(100);
	mig_cursor RECORD;
	BEGIN
	FOR mig_cursor IN (SELECT t.TABLENAME FROM _V_TABLE t
	WHERE TABLENAME LIKE 'T_ODS_CRM%')
	LOOP
	tablename := mig_cursor.TABLENAME;
	END LOOP;
	END;
	/

DECLARE

Netezza Syntax	Syntax After Migration

CREATE OR REPLACE PROCEDURE
sp_ntz_declare_inside_begin
(NATIONAL CHARACTER VARYING(10))
RETURNS INTEGER
LANGUAGE NZPLSQL
AS BEGIN_PROC
DECLARE
I_LOAD_DT ALIAS FOR $1;
BEGIN
DECLARE
MYCUR RECORD;
VIEWSQL1 NVARCHAR(4000);
BEGIN
FOR MYCUR IN (SELECT VIEWNAME,VIEWSQL
FROM
T_DDW_AUTO_F5_VIEW_DEFINE
WHERE OWNER = 'ODS_PROD')
LOOP
VIEWSQL1 := MYCUR.VIEWSQL;
WHILE INSTR(VIEWSQL1,'v_p_etldate')
> 0
LOOP
VIEWSQL1 :=
SUBSTR(VIEWSQL1,1,INSTR(VIEWSQL1,'v_p_etldat
e') - 1)||''''||I_LOAD_DT||''''||
SUBSTR(VIEWSQL1,INSTR(VIEWSQL1,'v_p_etldate')
+ 11);
END LOOP;
 	
EXECUTE IMMEDIATE VIEWSQL1;
END LOOP;
END;
 	
RETURN 0;
END;
END_PROC;
	CREATE OR REPLACE FUNCTION
	sp_ntz_declare_inside_begin
	(NATIONAL CHARACTER VARYING(10))
	RETURN INTEGER
	AS
	I_LOAD_DT ALIAS FOR $1;
	BEGIN
	DECLARE
	MYCUR RECORD;
	VIEWSQL1 NCHAR
	VARYING(4000);
	BEGIN
	FOR MYCUR IN (SELECT VIEWNAME,VIEWSQL
	FROM
	T_DDW_AUTO_F5_VIEW_DEFINE
	WHERE OWNER = 'ODS_PROD')
	LOOP
	VIEWSQL1 := MYCUR.VIEWSQL;
	WHILE INSTR(VIEWSQL1,'v_p_etldate')
	> 0
	LOOP
	VIEWSQL1 :=
	SUBSTR(VIEWSQL1,1,INSTR(VIEWSQL1,'v_p_etldat
	e') - 1)||''''||I_LOAD_DT||''''||
	SUBSTR(VIEWSQL1,INSTR(VIEWSQL1,'v_p_etldate')
	+ 11);
	END LOOP;
	
	EXECUTE IMMEDIATE VIEWSQL1;
	END LOOP;
	END;
	
	RETURN 0;
	END;
	/

EXECUTE AS CALLER

Netezza Syntax	Syntax After Migration

CREATE OR REPLACE PROCEDURE
sp_ntz_exec_as_caller
(CHARACTER VARYING(512))
RETURNS INTEGER
LANGUAGE NZPLSQL
EXECUTE AS CALLER
AS BEGIN_PROC
DECLARE
SQL ALIAS FOR $1;
BEGIN
EXECUTE IMMEDIATE SQL;
RETURN 0;
END;
END_PROC;
------------------------ 	
CREATE or replace PROCEDURE
sp_ntz_exec_as_owner
(CHARACTER VARYING(512))
RETURNS INTEGER
LANGUAGE NZPLSQL
EXECUTE AS OWNER
AS BEGIN_PROC
DECLARE
SQL ALIAS FOR $1;
BEGIN
EXECUTE IMMEDIATE SQL;
RETURN 0;
END;
END_PROC;
	CREATE OR REPLACE FUNCTION
	sp_ntz_exec_as_caller
	(CHARACTER VARYING(512))
	RETURN INTEGER
	SECURITY INVOKER
	AS
	SQL ALIAS FOR $1;
	BEGIN
	EXECUTE IMMEDIATE SQL;
	RETURN 0;
	END;
	/
	------------------------ 	
	CREATE OR REPLACE FUNCTION
	sp_ntz_exec_as_owner
	(CHARACTER VARYING(512))
	RETURN INTEGER
	SECURITY DEFINER
	AS
	SQL ALIAS FOR $1;
	BEGIN
	EXECUTE IMMEDIATE SQL;
	RETURN 0;
	END;
	/

Expression

SELECT result assign into variable.

Netezza Syntax	Syntax After Migration

CREATE OR REPLACE PROCEDURE
sp_sel_res_to_var
(NATIONAL CHARACTER VARYING(10))
RETURNS CHARACTER VARYING(ANY)
LANGUAGE NZPLSQL
AS BEGIN_PROC
DECLARE
counts INTEGER := 0 ;
I_LOAD_DT ALIAS FOR $1 ;
BEGIN
 	
COUNTS := SELECT COUNT(*)
FROM tb_sel_res_to_var
WHERE ETLDATE = I_LOAD_DT;
 	
EXECUTE IMMEDIATE 'insert into
TABLES_COUNTS values(''tb_sel_res_to_var'', ''' ||
I_LOAD_DT || ''', ' || COUNTS || ')' ;
 	
RETURN '0' ;
END;
END_PROC;
	CREATE OR REPLACE FUNCTION sp_sel_res_to_var
	(NATIONAL CHARACTER VARYING(10))
	RETURN CHARACTER VARYING
	AS
	counts INTEGER := 0 ;
	I_LOAD_DT ALIAS FOR $1 ;
	BEGIN
	
	SELECT COUNT(*)
	INTO COUNTS
	FROM tb_sel_res_to_var
	WHERE ETLDATE = I_LOAD_DT;
	
	EXECUTE IMMEDIATE 'insert into
	TABLES_COUNTS values(''tb_sel_res_to_var'', ''' ||
	I_LOAD_DT || ''', ' || COUNTS || ')' ;
	
	RETURN '0' ;
	END;
	/

6.7.4 System Functions

ISNULL()

Netezza Syntax	Syntax After Migration
SELECT A.ETL_DATE, A.BRANCH_CODE, A.CUST_NO , ISNULL (B.RES_STOCK,0) AS RES_STOCK , ISNULL (B.ZY_VOL ,0) AS ZY_VOL , ISNULL (B.ZJ_VOL,0) AS ZJ_VOL FROM tab123;	SELECT A.ETL_DATE, A.BRANCH_CODE, A.CUST_NO , NVL (B.RES_STOCK,0) AS RES_STOCK , NVL (B.ZY_VOL ,0) AS ZY_VOL , NVL (B.ZJ_VOL,0) AS ZJ_VOL FROM tab123;

NVL

Second parameter is missing.

Netezza Syntax	Syntax After Migration
SELECT NVL(SUM(A3.DA_CPTL_BAL_YEAR) / NULLIF(V_YEAR_DAYS, 0)) AS CPTL_BAL_AVE_YR , NVL(NVL(SUM (CASE WHEN A3.OPENACT_DT >= V_YEAR_START THEN A3.DA_CPTL_BAL_YEAR END) / NULLIF(V_YEAR_DAYS, 0)), 0) AS CPTL_BAL_AVE_YR_OP , NVL(SUM(A3.DA_CPTL_BAL) / NULLIF(V_YEAR_DAYS, 0)) AS CPTL_BAL_AVE FROM tab1 A3;	ELECT NVL(SUM(A3.DA_CPTL_BAL_YEAR) / NULLIF(V_YEAR_DAYS, 0), NULL) AS CPTL_BAL_AVE_YR , NVL(NVL(SUM (CASE WHEN A3.OPENACT_DT >= V_YEAR_START THEN A3.DA_CPTL_BAL_YEAR END) / NULLIF(V_YEAR_DAYS, 0), NULL), 0) AS CPTL_BAL_AVE_YR_OP , NVL(SUM(A3.DA_CPTL_BAL) / NULLIF(V_YEAR_DAYS, 0), NULL) AS CPTL_BAL_AVE FROM tab1 A3;

DATE

Casting the data type.

Netezza Syntax	Syntax After Migration
SELECT A1.ETL_DATE, A1.MARKET_CODE , A1.DECLARATION_DT , ROW_NUMBER() OVER(PARTITION BY A1.MARKET_CODE, A1.STOCK_CODE, DATE_PART('YEAR', DATE(A1.DECLARATION_DT)) ORDER BY A1.DECLARATION_DT DESC) AS RN FROM tb_date_type_casting A1; SELECT A1.ETL_DATE, A1.MARKET_CODE , A1.DECLARATION_DT , ROW_NUMBER() OVER(PARTITION BY A1.MARKET_CODE, A1.STOCK_CODE, DATE_PART('YEAR', DATE(A1.DECLARATION_DT)) ORDER BY A1.DECLARATION_DT DESC) AS RN FROM tb_date_type_casting A1;	SELECT A1.ETL_DATE, A1.MARKET_CODE , A1.DECLARATION_DT , ROW_NUMBER() OVER(PARTITION BY A1.MARKET_CODE, A1.STOCK_CODE, DATE_PART('YEAR', CAST(A1.DECLARATION_DT AS DATE)) ORDER BY A1.DECLARATION_DT DESC) AS RN FROM tb_date_type_casting A1;

analytic_function

Netezza Syntax	Syntax After Migration
SELECT COALESCE(NULLIF(GROUP_CONCAT(a.column_name),''),'*') FROM (SELECT a.column_name FROM tb_ntz_group_concat a WHERE UPPER(a.table_name) = 'EMP' ORDER BY a.column_pos) a; ---------- SELECT admin.group_concat('"top'|| lpad(a.table_name,2,'0')||'":{'||a.column_name||'}') topofund_data FROM (SELECT a.table_name, a.column_name FROM tb_ntz_group_concat a WHERE UPPER(a.table_name) = 'EMP' ORDER BY a.column_pos) a;	SELECT COALESCE(NULLIF(STRING_AGG(a.column_name,','),''),'*') FROM (SELECT a.column_name FROM tb_ntz_group_concat a WHERE UPPER(a.table_name) = 'EMP' ORDER BY a.column_pos) a; ---------- SELECT STRING_AGG('"top'|| lpad(a.table_name,3,'0')||'":{'||a.column_name||'}', ',') topofund_data FROM (SELECT a.table_name, a.column_name FROM tb_ntz_group_concat a WHERE UPPER(a.table_name) = 'EMP' ORDER BY a.column_pos) a;

Stored Procedure

Netezza Syntax	Syntax After Migration

CREATE OR REPLACE PROCEDURE
sp_ntz_proc_call
(CHARACTER VARYING(8))
RETURNS INTEGER
LANGUAGE NZPLSQL
AS
BEGIN_PROC
DECLARE
V_PAR_DAY ALIAS for $1;
V_PRCNAME NVARCHAR(50):=
'SP_O_HXYW_LNSACCTINFO_H';
D_TIME_START TIMESTAMP:=
CURRENT_TIMESTAMP;
O_RETURN INTEGER;
BEGIN
O_RETURN := 0;
CALL
SP_LOG_EXEC(V_PRCNAME,V_PAR_DAY,D_TIME_START,0,0);
 	
RETURN O_RETURN;
 	
END;
END_PROC;
	CREATE OR REPLACE FUNCTION sp_ntz_proc_call
	(CHARACTER VARYING(8))
	RETURN INTEGER
	AS
	V_PAR_DAY ALIAS for $1;
	V_PRCNAME NCHAR VARYING(50):=
	'SP_O_HXYW_LNSACCTINFO_H';
	D_TIME_START TIMESTAMP:=
	CURRENT_TIMESTAMP;
	O_RETURN INTEGER;
	BEGIN
	O_RETURN := 0;
	
	SP_LOG_EXEC(V_PRCNAME,V_PAR_DAY,D_TIME_START,0,0);
	
	RETURN O_RETURN;
	
	END;
	/

6.7.5 Operator

**

Netezza Syntax	Syntax After Migration
V_Out := V_Out + v_J * (v_N ** v_K) ;	V_Out := V_Out + v_J * (v_N ^ v_K) ;

NOTNULL and ISNULL

Netezza Syntax	Syntax After Migration
CASE WHEN ((A1.EXCEPT_OFF_SEQU_NO NOTNULL) AND (A2.LOG_SEQU_NO ISNULL)) THEN 0 ELSE 1 END AS FLG	CASE WHEN ((A1.EXCEPT_OFF_SEQU_NO NOTNULL) AND (A2.LOG_SEQU_NO ISNULL)) THEN 0 ELSE 1 END AS FLG

6.7.6 DML

GaussDB(DWS) keyword: SOURCE Specified As the Column Alias Without the AS Keyword

Netezza Syntax	Syntax After Migration
SELECT SUBSTR(OP_SOURCE ,1 ,4) SOURCE , ONLINE_FLAG, 'TRD' AS SRC_SYS , CURRENT_TIMESTAMP AS ETL_LOAD_TIME FROM tb_keyword_source;	SELECT SUBSTR(OP_SOURCE ,1 ,4) AS SOURCE , ONLINE_FLAG, 'TRD' AS SRC_SYS , CURRENT_TIMESTAMP AS ETL_LOAD_TIME FROM tb_keyword_source;

FREEZE

Netezza Syntax	Syntax After Migration
INSERT INTO tmp_tb_keyword_freeze (BRANCH_CODE, FREEZE, UNFREEZE, BRAN_JYSDM) SELECT BRANCH_CODE, FREEZE, UNFREEZE, BRAN_JYSDM FROM tb_keyword_freeze;	INSERT INTO tmp_tb_keyword_freeze (BRANCH_CODE, "FREEZE", UNFREEZE, BRAN_JYSDM) SELECT BRANCH_CODE, "FREEZE", UNFREEZE, BRAN_JYSDM FROM tb_keyword_freeze;

NOTE

A new configuration parameter "keywords_addressed_using_doublequote" should be introduced with the below value:

keywords_addressed_using_doublequote=freeze

keywords_addressed_using_as=owner,attribute,source,freeze

create table t12 (c1 int, FREEZE varchar(10)); ==> create table t12 (c1 int, "freeze" varchar(10));

select c1, Freeze from t12; ==> select c1, "freeze" from t12;

select c1 freeze from t12; ==> select c1 as freeze from t12;

OWNER (AS should be specified)

Netezza Syntax	Syntax After Migration
SELECT username owner FROM tb_ntz_keyword_owner;	SELECT username AS owner FROM tb_ntz_keyword_owner;

ATTRIBUTE (AS should be specified)

Netezza Syntax	Syntax After Migration
SELECT t1.etl_date, substr(t1.attribute,1,1) attribute , t1.cust_no, t1.branch_code FROM (SELECT etl_date,attribute,cust_no,branch_code FROM tb_ntz_keyword_attribute WHERE etl_date = CURRENT_DATE) t1;	SELECT t1.etl_date, substr(t1.attribute,1,1) AS attribute , t1.cust_no, t1.branch_code FROM (SELECT etl_date,attribute,cust_no,branch_code FROM tb_ntz_keyword_attribute WHERE etl_date = CURRENT_DATE) t1;

6.7.7 Unique Index

Unique Index

Netezza Syntax	Syntax After Migration
CREATE TABLE prod (prod_no number(6) not null unique, prod_name national character varying(32) not null, prod_desc clob) DISTRIBUTE ON (prod_no) ORGANIZE ON (prod_no, prod_name) ; ---------- CREATE TABLE prod (prod_no number(6) not null CONSTRAINT UQ_prod unique, prod_name national character varying(32) not null, prod_desc clob) DISTRIBUTE ON (prod_no) ORGANIZE ON (prod_no, prod_name) ; ---------- CREATE TABLE prod (prod_no number(6) not null PRIMARY KEY, prod_name national character varying(32) not null, prod_desc clob) DISTRIBUTE ON (prod_no) ORGANIZE ON (prod_no, prod_name) ; ---------- CREATE TABLE prod (prod_no number(6) not null, prod_name national character varying(32) not null, prod_desc clob, constraint uq_prod UNIQUE (prod_no)) DISTRIBUTE ON (prod_no) ORGANIZE ON (prod_no, prod_name) ; ---------- CREATE TABLE prod (prod_no number(6) not null, prod_name national character varying(32) not null, prod_desc clob) DISTRIBUTE ON (prod_no) ORGANIZE ON (prod_no, prod_name) ; ALTER TABLE prod ADD constraint uq_prod UNIQUE (prod_no);	CREATE TABLE prod (prod_no number(6) not null /* unique */, prod_name national character varying(32) not null, prod_desc clob) WITH(ORIENTATION=COLUMN) DISTRIBUTE BY HASH (prod_no) /* ORGANIZE ON (prod_no, prod_name) */ ; ---------- CREATE TABLE prod (prod_no number(6) not null /* CONSTRAINT UQ_prod unique */, prod_name national character varying(32) not null, prod_desc clob) WITH(ORIENTATION=COLUMN) DISTRIBUTE BY HASH (prod_no) /* ORGANIZE ON (prod_no, prod_name) */ ; ---------- CREATE TABLE prod (prod_no number(6) not null /* PRIMARY KEY */, prod_name national character varying(32) not null, prod_desc clob) WITH(ORIENTATION=COLUMN) DISTRIBUTE BY HASH (prod_no) /* ORGANIZE ON (prod_no, prod_name) */ ; ---------- CREATE TABLE prod (prod_no number(6) not null, prod_name national character varying(32) not null, prod_desc clob /*, constraint uq_prod UNIQUE (prod_no) */) WITH(ORIENTATION=COLUMN) DISTRIBUTE BY HASH (prod_no) /* ORGANIZE ON (prod_no, prod_name)*/ ; ---------- CREATE TABLE prod (prod_no number(6) not null, prod_name national character varying(32) not null, prod_desc clob) DISTRIBUTE BY HASH (prod_no) /*ORGANIZE ON (prod_no, prod_name)*/ ; /* ALTER TABLE prod

Netezza Syntax	Syntax After Migration
	ADD constraint uq_prod UNIQUE (prod_no); */

NOTE

This feature is applicable only for COLUMN store. For ROW store, Unique Index should not be commented.

6.8 FAQs

This section covers the frequently asked questions.

Q1: During installation, I get an error "Root privileged users are not allowed to install the DSC for Linux." What is the solution?

Answer: A root privileged user must not be used for installation and execution of the DSC for Linux. It is recommended that a user without root privileges be used to install and operate the DSC.

Q2: How do I configure the DSC to support GaussDB T, GaussDB A and GaussDB(DWS) version V100R002C60 for Teradata?

Answer: Perform the following steps to configure the DSC to support GaussDB T, GaussDB A and GaussDB(DWS) version V100R002C60 for Teradata:

1. Open the *features-teradata.properties* file in the *config* subfolder of TOOL_HOME.
2. Change the following variable values based on the requirement.
   - VOLATILE
   - PRIMARY INDEX

     For example,

     VOLATILE=*UNLOGGED | LOCAL TEMPORARY*
     PRIMARY INDEX=*ONE | MANY*

NOTE

The default variable value for **VOLATILE** is *LOCAL TEMPORARY* and for **PRIMARY INDEX** it is *MANY*.

6.9 Troubleshooting

This section contains a list of troubleshooting steps and solutions for issues encountered while using DSC.

The table lists the troubleshooting symptoms/issues along with their cause and solution.

Table 6-37 Error Message Reference

Symptom/Issue	Cause and Solution
Error occurred while formatting! Returning unformatted SQL: SELECT count(*) FROM table_temp;	**Cause**: This could be due to inconsistency in the number of opening and closing parentheses in the input file. **Solution**: Ensure each opening parenthesis has a corresponding closing parenthesis.
ERROR QueryConversionUtility:249 Query is not converted as it contains unsupported keyword: LAST	**Cause**: Input query file contains an unsupported keyword. **Solution**: Ensure no unsupported keywords are present in the scripts to be migrated.
Disk is almost full. Please clear the space and re-run the tool.	**Cause**: Insufficient space on the disk. **Solution**: Free space from the disk and retry the operation.
Please enter valid input parameters, Kindly refer the user manual to execute.	**Cause**: The possible cause could be: 1. No valid parameters are specified. 2. The short keys are in lower case. **Solutions**: 1. Provide all mandatory parameters when performing migration. 2. Ensure all short keys are in upper case.
No SQL files found in input folder. Hence stopping migration.	**Cause**: No valid SQL files are present in the input folder during the migration process. **Solution**: Ensure SQL files to be migrated are present in the input folder. For details, see **Migration Process**.
Migration Application failed to start : Currently we are not supporting this Database : *<database-name>*	**Cause**: Incorrect database name mentioned in the source database parameter. **Solution**: Use only Teradata or Oracle as values to the source database parameter.
Output folder is not set. Please enter an output folder and refer the user manual for syntax.	**Cause**: The output folder path is not specified. **Solution**: Specify a valid path for the output folder parameter.

Symptom/Issue	Cause and Solution
java.lang.OutOfMemoryError: GC overhead limit exceeded.	**Cause**: During migration, if the memory usage exceeds the preset value, DSC displays the error message and exits. **Solution**: Change the values of **initialJVMMemory** and **maxJVMMemory** in the **application.properties** configuration file to allocate more memory.
ascii "****" does not map to charset	**Cause**: DSC cannot detect the encoding format of the file input, and the character set configured in the system does not match that of the file input. As a result, an alarm is reported. **Solution**: Set **encodingFormat** to the actual encoding format and try again. Example: testmigration@BLR1000026522:~/18.1_RETEST/DSC/scripts/teradata> perl sqlTDtoGS.pl -i ../../PERL -o ../../PERL_OUT/ -m /home/testmigration/18.1_FORMAT_RETEST/sep6thpackage/DSC Extracting SQL contents from perl files started ascii "\xFF" does not map to Unicode at core/teradatacore.pm line 1270. ascii "\xFE" does not map to Unicode at core/teradatacore.pm line 1270. ascii "\xFE" does not map to Unicode at core/teradatacore.pm line 1270. ascii "\xFF" does not map to Unicode at core/teradatacore.pm line 1270. Extracting SQL contents from perl files completed ********** Schema Conversion Started ********** DSC process start time : Mon Jan 20 17:24:49 IST 2020 Statement count progress 100% completed [FILE(1/1)] Schema Conversion Progress 100% completed ********** Total number of files in input folder : 1 ********** Log file path :...../DSC/DSC/log/dsc.log DSC process end time : Mon Jan 20 17:24:49 IST 2020 DSC total process time : 0 seconds ********** Schema Conversion Completed **********

Error codes

Table 6-38 Error codes

Error Code	Error Information
Teradata	
DSC_ERR_003_001	Query/statement is not supported since the Teradata view "dbc.indices" is supported only for the indextype P and Q.
DSC_ERR_003_002	Error in Bteq processing. Something went wrong while processing the BTEQ commands.
DSC_ERR_003_003	Query/statement is not supported in ddl DSC. Please check the same and refer user manual for the supported feature list.
DSC_ERR_003_004	Unsupported format decimal format like ZZZ99Z, ZZZ.ZZ9.
DSC_ERR_003_005	The tool does not support the "IN/NOT IN to EXISTS/NOT EXISTS conversion" for the query in which its outer query refers multiple tables and the column(s) specified with IN / NOT IN operator do not have table reference.
DSC_ERR_003_006	The tool does not support the query in which its outer query refers multiple tables and the column(s) specified with IN / NOT IN operator do not have table reference.
DSC_ERR_003_007	Primary Index without column is not supported.
DSC_ERR_003_008	TeradataQuerySplitter config file contains list is not supported.
DSC_ERR_003_009	Gauss does not support WITH CHECK OPTION in CREATE VIEW. Please enable the config_param tdMigrateVIEWCHECKOPTION to comment the WITH CHECK OPTION syntax in the statement.

Error Code	Error Information
DSC_ERR_003_010	Gauss does not have an equivalent syntax for CHARACTER SET & CASE SPECIFIC option in column-level. Please enable the config_param tdMigrateCharsetCase to comment the CHARACTER SET & CASE SPECIFIC option syntax in the statement.
DSC_ERR_003_011	Gauss does not have equivalent syntax for LOCK option in CREATE VIEW and INSERT statement. You can rewrite this statement or set the configuration parameter tdMigrateLOCKOption to TRUE to comment the LOCK syntax in this statement.
DSC_ERR_003_012	Invalid width (Number of rows) parameter in MDIFF function.
DSC_ERR_003_013	First 2 parameters should be present in MDIFF function.
DSC_ERR_003_014	Query/statement is not supported as ORDER BY clause is not present in TOP WITH TIES. Please check the same and refer user manual.
DSC_ERR_003_015	Column mismatch for the TITLE conversion.
DSC_ERR_003_016	Query/statement is not supported as same Table alias is addressed in both inner and outer query. Please check the same and refer user manual for the supported feature list.
DSC_ERR_003_017	Sub query list does not have columns.
DSC_ERR_003_018	Number of expressions specified in the outer query does not match with inner query.
DSC_ERR_003_019	Error while loading the .RUN FILE from given location.
DSC_ERR_003_020	Unable to delete the file, file not found.
DSC_ERR_003_021	Unable to delete the file, failed with IOEXception.
DSC_ERR_003_022	Please specify the value for environment_file_path parameter in features-teradata.properties.

Error Code	Error Information
Application	
DSC_ERR_004_001	Application has timed out, exceeded the hours specified in the config file. Please configure the Timeout parameter in the application.properties to higher value.
DSC_ERR_004_002	Error while loading the property files from config directory.
DSC_ERR_004_003	Error while loading the property files from config directory, directory is not readable.
DSC_ERR_004_004	Error while loading the property file.
DSC_ERR_004_005	Unable to load the JSON file.
DSC_ERR_004_006	DSC tool does not support this Conversion type provided.
DSC_ERR_004_007	Error occurred while framing output replacement query.
DSC_ERR_004_008	Invalid index value while parsing the script.
DSC_ERR_004_009	Error in conversion process, unable to convert the script.
DSC_ERR_004_010	No SQL files found in the input directory with the extension specified in the fileExtension property in application.properties.
DSC_ERR_004_011	The query length parameter (MaxSqlLen) value is not valid.
DSC_ERR_004_012	Since the input folder has write privileges to Group and/or Others, process is stopped due to security reason.
DSC_ERR_004_013	Since the output directory has write privileges to Group and/or Others, process is stopped due to security reason.
DSC_ERR_004_014	Disk is almost full. Please clear the space and re-run the tool.
DSC_ERR_004_015	DSC has been cancelled as configured by the user.

Error Code	Error Information
DSC_ERR_004_016	Error occurred while formatting the sql scripts.
DSC_ERR_004_017	Invalid index specified for fetching the element from list while formatting the scripts
DSC_ERR_004_018	Error occurred while converting from string to integer.
DSC_ERR_004_019	Input File is modified while DSC is in progress.
DSC_ERR_004_020	Process is null, unable to read encoding format.
DSC_ERR_004_021	Target File does not have write permissions.
DSC_ERR_004_022	The target directory does not have write privileges to Group and/or Others, process is stopped due to security reason.
DSC_ERR_004_023	PL/SQL object contains incorrect DDL/ Query. Please check the script for the query position specified in the log.
DSC_ERR_004_024	PreQueryValidation failed due to bracket mismatch or invalid terminator.
DSC_ERR_004_025	Conversion task name is not valid.
DSC_ERR_004_026	Database entered by the user is not supported by the DSC tool.
DSC_ERR_004_027	Gauss db password should not be empty.
DSC_ERR_004_028	Gauss db password should not be empty.
DSC_ERR_004_029	Target db entered in the Gaussdb.properties is not valid.
DSC_ERR_004_030	User name entered in the Gaussdb.properties is empty.
DSC_ERR_004_031	Port entered in the Gaussdb.properties is not valid.
DSC_ERR_004_032	IP entered in the Gaussdb.properties is not valid.

Error Code	Error Information
DSC_ERR_004_033	Database name entered in the Gaussdb.properties is empty.
DSC_ERR_004_034	DSC Application failed to start.
DSC_ERR_004_035	Since the environment variable path has write privileges to Group and/or Others, process is stopped due to security reason.
DSC_ERR_004_036	Error while loading environment parameter File.
DSC_ERR_004_037	Invalid input (empty/space/string value) for the parameter NoOfThreads in application.properties. Hence taking the default processes.
DSC_ERR_004_038	Input for the parameter NoOfThreads in application.properties is less than 1. Hence taking the default processes.
DSC_ERR_004_039	Error in processing the DDL query.
DSC_ERR_004_040	Error in processing the PL/SQL query.
DSC_ERR_004_041	Error in post processing the query.
DSC_ERR_004_042	Invalid application timeout value, default to 4 hours.
DSC_ERR_004_043	Error in writing the output file.
DSC_ERR_004_044	Error in reading the input file.
DSC_ERR_004_045	No valid files found in the input directory for migration.
DSC_ERR_004_046	Query is not converted as it contains unsupported keyword.
DSC_ERR_004_047	Error while reading the property.
DSC_ERR_004_048	PreQueryValidation failed due to query exceeds maximum length (MaxSqlLen config parameter).
DSC_ERR_004_049	Thread count entered in the Gaussdb.properties is not valid.
Wrapper	
DSC_ERR_005_003	Reading file Failed with error: File not found Exception.

Error Code	Error Information
DSC_ERR_005_004	Reading file Failed with error: IOException.
DSC_ERR_005_005	Root privileged users are not allowed to execute the DSC tool.
DSC_ERR_005_006	Error while getting the id of os user used to execute the DSC tool.
DSC_ERR_005_007	Arguments specified is not valid, please check the user manual for the command line arguments.
DSC_ERR_005_008	File name is not specified for reading the encoding type.
DSC_ERR_005_009	Invalid argument specified for the encoding parameter.
DSC_ERR_005_010	Source database is not set. Please enter a valid source db and refer the user manual for syntax.
DSC_ERR_005_011	Commandline database specified for source to target is not supported by the DSC tool.
DSC_ERR_005_012	Error in loading config file with IOException.
DSC_ERR_005_013	Initial JVM memory is greater than maximum JVM memory.
DSC_ERR_005_014	Invalid value specified for configValue.
DSC_ERR_005_015	Invalid source database specified for source-db option.
DSC_ERR_005_016	Invalid target database specified for target-db option.
DSC_ERR_005_017	Invalid conversion type specified for dsc-type option.
DSC_ERR_005_018	Invalid application language specified for application-lang option.
DSC_ERR_005_019	Conversion-type should be DDL for application-lang type as perl.
DSC_ERR_005_020	Source-db should be teradata for application-lang type as perl.

Error Code	Error Information
DSC_ERR_005_021	Please use "-VN [V1R7 | V1R8_330]" or "--version-number [V1R7 | V1R8_330]" to specify the kernel version which can be either V1R7 or V1R8_330.
DSC_ERR_005_022	Input directory does not exist.
DSC_ERR_005_023	Getting path for input directory failed with IOException.
DSC_ERR_005_024	Getting path for output directory failed with IOException.
DSC_ERR_005_025	Setting file permission for output directory failed with IOException.
DSC_ERR_005_026	Creating output directory failed.
DSC_ERR_005_027	Setting file permissions for log directory/file failed with FileException.
DSC_ERR_005_028	Error while connecting to GaussDB, Failed with error.
DSC_ERR_005_029	Error occurred due to file permission while creating or executing the file.
DSC_ERR_005_030	No arguments specified in the commandline.
DSC_ERR_005_031	Error occurred in creating output directory.

6.10 Glossary

The following table lists the acronyms, abbreviations, terms, and their descriptions.

Term	Description
C	
Common Table Expression (CTE)	A common table expression is a temporary named result set defined in a query and can be referenced later in the main query.
D	

Term	Description
Databases (DB)	A database is a collection of related information, usually organized to make general retrieval simple and efficient. Database attributes: • Database name • Endian format (BIG_ENDIAN big-endian or LITTLE_ENDIAN little-endian) • Relationship • All databases have relationships with others.
Database administrator (DBA)	A database administrator is a person responsible for installing, configuring, upgrading, managing, monitoring, and maintaining a database in an organization. This role includes developing and designing database policies, monitoring and optimizing database performance and capacity, and planning future expansion. Database administrators can also plan, coordinate, and implement security measures to ensure the security of the database.
E	
Encoding	In information processing, encoding is a rule system that converts information such as letters, words, sounds, images, or gestures into a format. Sometimes, it shortens them or encrypts them for communication or storage.
I	
Indexes	An index is an ordered data structure in a database management system that facilitates the query and update of data in a table.
M	

Term	Description
Migration	Migration is the process to migrate scripts, queries, schemas, and data from a source database (such as Teradata) to a target database (such as GaussDB(DWS).
Metadata	Metadata is the information about data. Metadata defines data attributes and is used to specify the data storage location, historical data, indexing resources, and record information.
O	
Operating System (OS)	An operating system is a program loaded into the computer in prior to other programs through a boot program to manage all other programs.
Q	
Queries	A query is an information request sent to a database. A query executes the SQL statement and returns the result set defined by the statement.
S	
Structured Query Language (SQL)	It is a programming language widely used to access, update, manage, and query data in relational databases.
Schema	A schema is a structure described in a formal language supported by a database management system. A schema is an organization form of data. It describes how a database is built. (In a relational database, it describes how a database is divided into tables.)
T	
Teradata	Teradata is a relational database management system. It can be used to run multiple complex queries at the same time. It supports ad hoc queries using SQL statements and is widely used to manage large warehouse operations.

Term	Description
Tables	A table is a collection of closely related columns. A table consists of rows that contain different column values.
V	
Views	A view allows access to specific rows or columns of a table. A view can be created from one or more tables and is determined by the queries used to create the view.

7 DWS-Connector

7.1 DWS-Connector Version Description

Table 7-1 Change History

Version	Change Description	Remarks
1.0	This issue is the first official release.	dws-connector-flink only releases Scala2.11 Flink 1.12.
1.0.2	Optimized the exception retry logic of dwsclient. The retry mode is changed from retry upon all exceptions to retry only upon five types of exceptions: connection exception, database read-only, timeout, excessive connections, and lock exception.	Compatible versions of dws-connector-flink: Scala2.11: Flink 1.12 and 1.13 Scala2.12: Flink 1.12, 1.13, and 1.15

Version	Change Description	Remarks
1.0.3	1. Resolved known issues and optimized performance. 2. Supports the **update** write mode. 3. Supports unique indexes. 4. As the **update** mode is supported, to avoid ambiguity of the **upsert** interface in dwsClient, the **write** interface is used for write operations. The **write** interface is recommended for both the two type of writes.	-
1.0.4	Increased the SQL execution timeout interval to avoid long-time blocking.	-
1.0.5	Resolved the problem that duplicate data is lost when being written to a table without a primary key.	-

Version	Change Description	Remarks
1.0.6	1. Optimized the cache saving logic to improve the throughput when the CPU is insufficient. 2. Temporary tables are reused to prevent frequent creation of temporary tables in **COPY MERGE/ UPSERT** scenarios. 3. The CSV format is added for **COPY** to avoid that complex data cannot be imported to the database due to special characters.	-
1.0.7	1. Retry is supported after data fails to be written during database restart. 2. The AS mode is added to create temporary tables to solve the problem that **COPY MERGE/ UPSERT** cannot be used in tables with primary keys. 3. The database fields are case-insensitive by default. 4. The primary key printing parameter is added to Flink SQL statements to locate problems when data is missing.	-

Version	Change Description	Remarks
1.0.8	1. Fixed the problem that the case of the Flink SQL primary key must be the same as that in the database. 2. Added the parameter for setting sink concurrency.	-
1.0.9	Optimized the import of data of the time type.	-
1.0.10	1. Resolved the issue of data loss caused by concurrent delete and insert operations on the client. This could happen when the insert operation ran before the delete operation, and the same primary key was deleted and then inserted in the same cache batch. 2. Resolved the issue of occasional data loss when Kafka writes data to GaussDB(DWS). 3. The connector parameter **ignoreUpdateBefore** is added. Some main parameters are compatible with flink-connector-jdbc.	-

7.2 dws-client

Description

dws-client is a high-performance and convenient data import tool based on GaussDB(DWS) JDBC. Ensure that JDBC can be connected when using GaussDB(DWS) client. Using dws-client to import data has the following advantages:

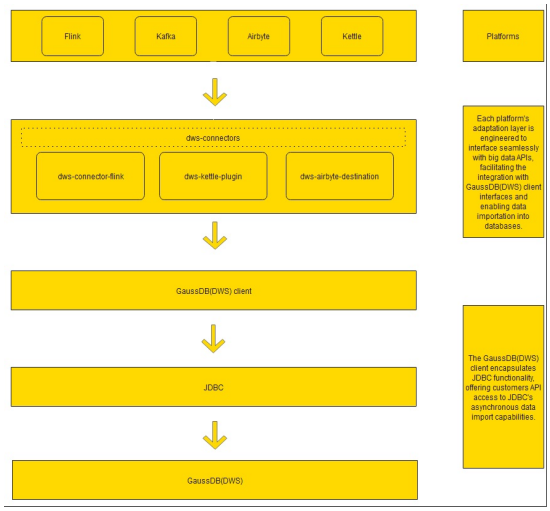
1. dws-client limits the cache space and time and supports batch import to improve the data import performance, meeting the real-time data import requirements during peak and off-peak hours.

   NOTE

   In scenarios that do not have strict real-time requirements, operations on single data records are cached until they form a batch. Then, they will be performed in a batch. This improves the write performance.

2. dws-client supports concurrent data import.
3. dws-client supports multiple high-performance import modes and primary key conflict policies to meet import requirements in various scenarios.
4. dws-client supports API-based interaction, making it easy to use.

Figure 7-1 dws-client interaction scenario



Dependency

dws-client has been added to the Maven repository. You can select the latest version from the repository. For details, visit **https://mvnrepository.com/artifact/com.huaweicloud.dws/dws-client.**

```
<dependency>
  <groupId>com.huaweicloud.dws</groupId>
  <artifactId>dws-client</artifactId>
  <version>${version}</version>
</dependency>
```


Scenario & Usage

Prerequisite: The client has been initialized.

The following is a simple example. You only need to configure the database connection. Retain the default values for other parameters.

```
public DwsClient getClient(){
        DwsConfig config = DwsConfig
                .builder()
                .withUrl("jdbc:gaussdb://***/gaussdb")
                .withUsername("***")
                .withPassword("*****")
                .build();
        return new DwsClient(config);
    }
```

Scenario 1: Table-level parameter configuration

```
return DwsConfig.builder()
                .withUrl(System.getenv("db_url"))
                .withPassword(System.getenv("db_pwd"))
                .withUsername(System.getenv("db_username"))
.withAutoFlushBatchSize(1000) // The default batch size is 1000.
                .withTableConfig("test.t_c_batch_size_2", new TableConfig()
.withAutoFlushBatchSize(500)); //The batch size is 500 for table test.t_c_batch_size_2;
```

Scenario 2: Using a database connection to execute SQL statements

This API is mainly used for some special services when the currently supported functions cannot meet the requirements. For example, to query data, you can directly use the native JDBC connection to operate the database.

The API parameter is a function-based interface. The interface provides a database connection. The return value can be of any type, which is determined by the return type of the service.

```
public void sql() throws DwsClientException {
        Integer id = getClient().sql(connection -> {
            try (ResultSet resultSet = connection.createStatement().executeQuery("select id from test.user
where name = 'zhangsan'")) {
                if (resultSet.next()) {
                    return resultSet.getInt("id");
                }
            }
            return null;
        });
        System.out.println("zhangsan id = " + id);
    }
```

Scenario 3: Obtaining table information

The API can obtain the table structure (cached) based on a table name affixed with a schema name. The table structure definitions include all columns and primary keys.

```
public void getTableSchema() throws DwsClientException {
        TableSchema tableSchema = getClient().getTableSchema(TableName.valueOf("test.test"));
    }
```

Scenario 4: Importing data to a database

The client provides a write API for importing data to the database. The Operate API is used to operate table columns. If the API is submitted, the table operation is complete and the client start importing the data to the database. You can select synchronous or asynchronous when submitting the operation. When setting a field, you can choose whether to ignore the setting when a primary key conflict occurs.


```
public void write() throws DwsClientException {
        getClient().write("test.test")
                .setObject("id", 1)
                .setObject("name", "test")
//This setting takes effect only when data is inserted. If a primary key conflict occurs, the setting is not
updated.
                .setObject("age", 38, true)
// Asynchronously save the data to the database. The result is returned after data is stored in the
background cache.
                //.commit()
//  The result is returned after data is successfully saved to the database.
                .syncCommit();
    }
```

Scenario 5: Deleting data

The deletion API and import API are carried by Operate. However, the primary key column must be set during deletion, and the "column update does not take effect" is ignored.

```
public void delete() throws DwsClientException {
        getClient().delete("test.test")
                .setObject("id", 1)
// Asynchronously save the data to the database. The result is returned after data is stored in the
background cache.
                //.commit()
//  The result is returned after data is successfully saved to the database.
                .syncCommit();
    }
```

Scenario 6: Forcibly updating the cache to the database

```
public void flush() throws DwsClientException {
        getClient().flush();
    }
```

Scenario 7: Closing resources

When the close operation is performed, the cache is updated to the database. After the close operation is performed, APIs such as importing data to the database, deleting data, and executing SQL statements cannot be executed.

```
public void close() throws IOException {
        getClient().close();
    }
```

Listening to Data Import Events

In the asynchronous import scenario, if you want to know which data has been imported to the database, you can bind the flushSuccess function interface. This interface is called back to report the import information after the database transaction is submitted.

```
public DwsClient getClient() {
        DwsConfig config = DwsConfig
                .builder()
                .withUrl("jdbc:postgresql://***/gaussdb")
                .withUsername("***")
                .withPassword("****")
                .onFlushSuccess(records -> {
                    for (Record record : records) {
                        log.info("flush success. value = {}, pk = {}", RecordUtil.toMap(record),
RecordUtil.getRecordPrimaryKeyValue(record));
                    }
                })
                .build();
        return new DwsClient(config);
    }
```


Listening to Abnormal Background Tasks

In the asynchronous import scenario, data is imported to the database by a background task. You can bind the ERROR function interface to detect the background task failure. Otherwise, the exception can only be found when the data is submitted next time. If the bound interface does not throw an exception, the exception is cleared and will not be thrown when the data is submitted next time, otherwise, an interface exception is thrown to the service when the request is submitted next time.

```
public DwsClient getClient() {
        DwsConfig config = DwsConfig
                .builder()
                .withUrl("jdbc:postgresql://***/gaussdb")
                .withUsername("***")
                .withPassword("****")
                .onError((clientException, client) -> {
                    if (clientException instanceof DwsClientRecordException) {
                        DwsClientRecordException recordException = (DwsClientRecordException) clientException;
                        List<Record> records = recordException.getRecords();
                        List<DwsClientException> exceptions = recordException.getExceptions();
                        for (int i = 0; i < records.size(); i++) {
                            log.error("pk = {} . error = {}", RecordUtil.getRecordPrimaryKeyValue(records.get(i)),
exceptions.get(i));
                        }
                    }
                    if (clientException.getCode() != ExceptionCode.CONNECTION_ERROR &&
clientException.getCode() != ExceptionCode.LOCK_ERROR) {
                        throw clientException;
                    }
                    log.error("code = {}", clientException.getCode(), clientException.getOriginal());
                    return null;
                })
                .build();
        return new DwsClient(config);
    }
```

Exception Handling

Exceptions can be classified into three types:

1. InvalidException is not thrown and is triggered when the request parameter is invalid.
2. 2. DwsClientException encapsulates all exceptions, including the parsed code and original exceptions.
3. 3. DwsClientRecordException is an extension to DwsClientException. It includes the datasets written to the exception and the corresponding DwsClientException exception.

The following table lists the exception codes.

```
public enum ExceptionCode {
    /**
   /* Invalid parameter */
     */
    INVALID_CONFIG(1),

    /**
* Connection exception.
     */
    CONNECTION_ERROR(100),
    /**
* Read-only
```



```
     */
    READ_ONLY(101),
    /**
* Timeout
     */
    TIMEOUT(102),
    /**
* Too many connections
     */
    TOO_MANY_CONNECTIONS(103),
    /**
* Locking exception.
     */
    LOCK_ERROR(104),


    /**
* Authentication failed.
     */
    AUTH_FAIL(201),
    /**
* Closed
     */
    ALREADY_CLOSE(202),
    /**
* No permission.
     */
    PERMISSION_DENY(203),
    SYNTAX_ERROR(204),
    /**
* Internal exception.
     */
    INTERNAL_ERROR(205),
    /**
* Interruption exception.
     */
    INTERRUPTED(206),
    /**
* The table is not found.
     */
    TABLE_NOT_FOUND(207),
    CONSTRAINT_VIOLATION(208),
    DATA_TYPE_ERROR(209),
    DATA_VALUE_ERROR(210),

    /**
* Exceptions that cannot be parsed
     */
    UNKNOWN_ERROR(500);
    private final int code;

}
```

Detailed Configuration

Parameter	Description	Default Value	Supported Versions
url	JDBC address connecting to the GaussDB(DWS) database	-	1.0

Parameter	Description	Default Value	Supported Versions
username	GaussDB(DWS) database user name	-	
password	GaussDB database user password	-	
connectionMaxUseTimeSeconds	Maximum duration specified for a connection, in seconds. If the duration exceeds the value of this parameter, the current connection is forcibly closed and a new connection is obtained. The **COPY_MERGE** and **COPY_UPSERT** statements involve temporary tables. The schemas of the temporary tables are cleared only when the connection is disconnected. So, this parameter is introduced	3600	
connectionMaxIdleMs	Maximum idle time of a connection (ms)	60000	

Parameter	Description	Default Value	Supported Versions
metadataCacheSeconds	Metadata cache duration, in seconds. To improve performance, this parameter is used to set the cache expiration time for data that is not frequently changed, for example, the table structure.	180	
retryBaseTime	Sleep time during retry = retryBaseTime x Number of times + (0–retryRandomTime) ms. This parameter specifies the retry base time (ms).	1000	
retryRandomTime	Sleep time during retry = retryBaseTime x Number of times + (0–retryRandomTime) ms. This parameter specifies the random number range during retry. This parameter is used to separate the execution time of two tasks in the deadlock scenario.	300	
maxFlushRetryTimes	Maximum number of attempts to execute a database update task.	3	

Parameter	Description	Default Value	Supported Versions
autoFlushBatchSize	Database update policy: The number of cached records should be no less than the value of **autoFlushBatchSize**, or the difference between the current time and the cache start time is greater than or equal to the value of **autoFlushMaxIntervalMs**. This parameter specifies the maximum number of cached records.	5000	
autoFlushMaxIntervalMs	Database update policy: The number of cached records should be no less than the value of **autoFlushBatchSize**, or the difference between the current time and the cache start time is greater than or equal to the value of **autoFlushMaxIntervalMs**. This parameter specifies the maximum cache duration, in milliseconds.	3000	

Parameter	Description	Default Value	Supported Versions
copyWriteBatchSize	When **writeMode** is set to **AUTO** and the data volume is less than the value of **copyWriteBatchSize**, the **UPSERT** method is used to import data to the database. Otherwise, the **COPY**/**COPY +UPSERT** method is used to import data to the database based on whether the primary key exists.	6000	

Parameter	Description	Default Value	Supported Versions
writeMode	Data write methods: • AUTO: When importing data to the database, **UPSERT** is used if the data volume is less than the copyWriteBatchSize value. Otherwise, **COPY_UPSERT** is used instead. • COPY_MERGE: – If there is a primary key, the **COPY** + **MERGE** method is used to import data to the database. – If there is no primary key, the **COPY** method is used to import data to the database. • **COPY_UPSERT**: – If there is no primary key, the **COPY** method is used to import data to the database. – If there is a primary key,	AUTO	

Parameter	Description	Default Value	Supported Versions
	the **COPY** + **UPSERT** method is used to import data to the database. • **UPSERT**: – If there is no primary key, use **INSERT INTO** to import data to the database. – If there is a primary key, use **UPSERT** to import data to the database. • UPDATE: – Use the **UPDATE WHERE** syntax to update data. If the original table does not have a primary key, you can specify unique keys. A column specified as a unique key does not need to be a unique index, but a non-unique index may impact performanc e.		

Parameter	Description	Default Value	Supported Versions
	• COPY_UPDATE: – Data is imported to a temporary table by the **COPY** method. Temporary tables can be used to accelerate the update using **UPDATE FROM WHERE**. • UPDATE_AUTO: – If the batch size is less than **copyWriteBatchSize**, **UPDATE** is used. Otherwise, **COPY_UPDATE** is used.		

Parameter	Description	Default Value	Supported Versions
conflictStrategy	Primary key conflict policy when the database has primary keys: • **INSERT_OR_IGNORE**: Ignore new data when a primary key conflict occurs. • **INSERT_OR_UPDATE**: Use the new data column to update the original data column when a primary key conflict occurs. • **INSERT_OR_REPLACE**: Replace the original data with new data when a primary key conflict occurs. The data columns in the database that are not contained in the new data are set to null. The update of all columns is the same as that of **INSERT_OR_UPDATE**.	INSERT_OR_UPDATE	

Parameter	Description	Default Value	Supported Versions
threadSize	Number of concurrent tasks. Tasks are submitted asynchronously by table, and multiple tables can be executed concurrently. Each table may have a different number of field columns. For instance, if a batch contains 100 operations with fields A, B, and C, and 200 operations with fields A, B, and D, the same operation fields are classified into one type. Different types of fields can be imported concurrently to the database, and you can set this parameter based on the two scenarios to improve throughput.	3	
logSwitch	Log switch. If this function is enabled, detailed process logs are recorded for debugging or fault locating.	false	

Parameter	Description	Default Value	Supported Versions
logDataTables	Tables whose data needs to be printed during data import to the database for data comparison during fault locating.	-	
flushSuccessFunction	Callback function after data is successfully imported to the database	-	
errorFunction	Callback function when a background task fails to be executed	-	
batchOutWeighRatio	To improve the overall throughput, you can set this parameter when the requirement on **autoFlushBatchSize** is not strict. When data is submitted to the buffer and the data volume in the buffer is greater than **batchOutWeighRatio** x **autoFlushBatchSize**, the task of submitting data to the database will be executed. This parameter is used to preferably use background threads to submit import tasks, rather than using service threads.	1	

Parameter	Description	Default Value	Supported Versions
tableConfig	If you have multiple tables sharing one client and plan to set different values for **conflictStrategy**, **writeMode**, **copyWriteBatchSize**, **autoFlushMaxIntervalMs**, **autoFlushBatchSize**, and **batchOutWeighRatio**, use this parameter to configure them at the table level and have the configuration apply globally to tables that are not individually configured. **NOTE** Once a table-level parameter is configured, other table-level parameters will automatically be set to default values. Therefore, you must also set other table-level parameters.	-	

Parameter	Description	Default Value	Supported Versions
uniqueKeys	This parameter is a table-level parameter and must be configured through **tableConfig**. If a table does not have a primary key but has a unique index, this parameter is used to specify a column as a unique constraint when the table is imported to the database. In the **update** scenario, this column does not need to be a unique index or primary key, however, in the **upsert** scenario, it must be a unique index or primary key.	-	1.0.3

Parameter	Description	Default Value	Supported Versions
copyMode	The format of copying data to the database is as follows: **CSV**: Data is concatenated into a string in CSV format and enclosed in double quotation marks. Fields are separated by commas (,), and data is separated by line breaks. Data is imported to the database using the JDBC Copy API. The performance of this mode is slightly lower than that of the DELIMITER mode, but the performance is stable and reliable. **DELIMITER**: Use the copy API to import data fields to the database. Characters are separated by 0X1E, and data is separated by 0X1F. In this mode, the data does not contain delimiters. If the data contains delimiters, an error is reported and the data cannot be imported to the database. In addition, this	CSV	1.0.6

Parameter	Description	Default Value	Supported Versions
	mode defines the null string as null.		
caseSensitive	Indicates whether a table field is case sensitive.	false	1.0.7
createTempTable-Mode	Indicates the mode of creating a temporary table when copy merge/upsert is used. • **AS**: Use the **create temp table *** as select * from *** as** method. This method allows the use of tables with auto-increment fields, but it may result in lower performance. • **LIKE**: Use the **create temp table *** like** method. This method does not allow tables with auto-increment fields.	AS	1.0.7

Parameter	Description	Default Value	Supported Versions
numberAsEpochMsForDatetime	Indicates whether to convert the source data to the corresponding time type by millisecond if the database field is of the time type (date\time \timestamp) and the data source is of the number type. **NOTE** • This parameter does not take effect when data is copied to the database. • In versions earlier than this version, if this parameter is enabled and data is of the number type, strings will be regarded as timestamps.	false	1.0.9

Parameter	Description	Default Value	Supported Versions
stringToDatetime-Format	If the database field is of the time type (date\time \timestamp) and the data source is of the string type, **SimpleDateFormat** is used to convert the field to the date type based on the **stringToDatetimeFormat** format, and then the timestamps in dates are used to construct the data of the corresponding type in the database. **NOTE** If this parameter is set, the function is enabled. Do not set this parameter if it is not required.	null	
updateAll	Whether the set field contains the primary key during upsert.	true	1.0.10

7.3 dws-connector-flink

7.3.1 Dependency

Overview

The dws-connector-flink tool connects the GaussDB(DWS) client to Flink by encapsulating the GaussDB(DWS) client. Its import capability is equivalent to that of the GaussDB(DWS) client. The GaussDB(DWS) team created dws-connector-flink, and it will be continually improved to align with the GaussDB(DWS) database.

NOTE

dws connector of dws-flink-connector supports only single concurrent query of inventory data and does not support concurrent read.

How to Use

- Introduced in Maven mode

  dws-connector-flink has been added to the Maven repository. You can select the latest version. For details, visit **https://mvnrepository.com/artifact/com.huaweicloud.dws**.

```
<dependency>
   <groupId>com.huaweicloud.dws</groupId>
   <artifactId>dws-connector-flink_${scala.version}_${flink.version}</artifactId>
   <version>${version}</version>
</dependency>
```

- Introduced in Flink SQL mode

  When using Flink SQL to implement dws-connector-flink, you need to place the dws-connector-flink package and its dependencies in the Flink class loading directory. In 1.0.3 and later versions, the packages with dependencies have been released to the Maven repository. You can download the packages from the repository.

  a. Select the package that matches the Flink environment.

  Figure 7-2 Flink package

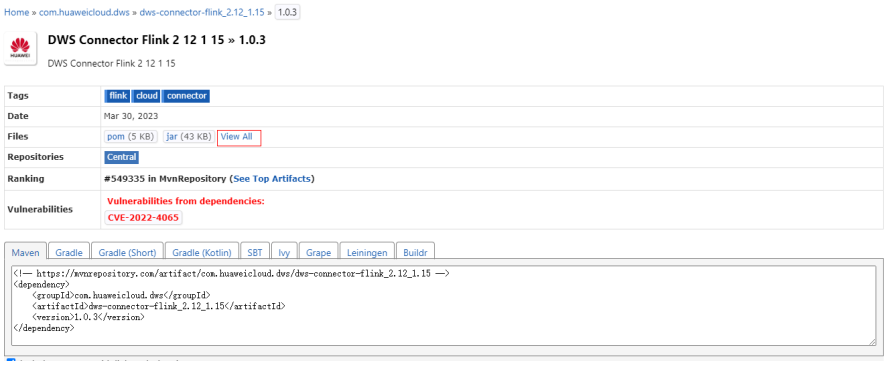
  b. Switch to the software package details page.

  Figure 7-3 Flink package details page

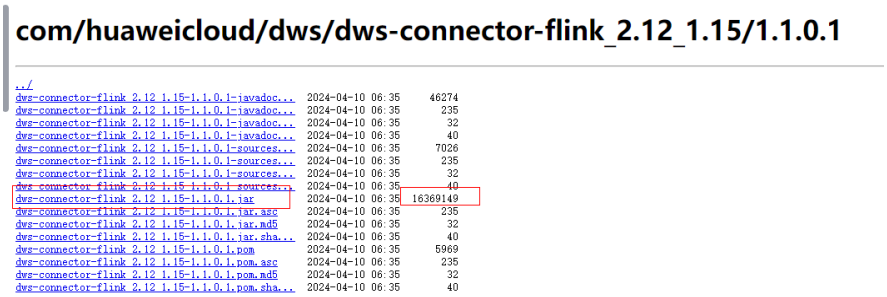
  c. Select the required version.

Figure 7-4 Selecting the version to be downloaded



d. Select to view all files.

Figure 7-5 Viewing all files



NOTE

- In the file list of the corresponding version, **javadoc** is the document, and **source** is the source code. Exclude these two file types. The package with dependencies is the largest JAR package. The file name extension may vary slightly depending on the version.
- If the Flink version does not exist, use a version similar to the current version.

e. Click to download files.

7.3.2 Stream API Job Type

When developing Flink jobs using APIs, the DWS-connector offers the **DwsGenericSinkFunction** class, which handles GaussDB(DWS) client initialization and Flink checkpoint APIs. Additionally, the DwsInvokeFunction interface manages the service logic. For details, see **Stream API Job Type**.

API Description

```
public static <T> SinkFunction<T> sink(DwsConfig config, JSONObject context, DwsInvokeFunction<T> invoke);
```

- The **config** parameter in GaussDB(DWS) client functions in the same way as the client itself.
- **context** is a global context provided for operations such as cache. It can be specified during GaussDB(DWS) client construction, and is called back each time with the data processing interface.

- **invoke** is a function interface used to process data.

```
/**
 *  Execute data processing callback when Flink executes invoke.
 * @param value indicates the current received data.
 * @param client indicates the GaussDB(DWS) client constructed based on the configuration.
 * @param context indicates a global object specified during construction. This parameter is
attached to each callback and can be used as a global cache.
 * @throws DwsClientException
 */
void accept(T value, DwsClient client, JSONObject context) throws DwsClientException;
```

Examples

The DwsSink interface is used for quick construction. The following is an example:

```
SinkFunction<EventLog> dwsSink = DwsSink.sink(DwsConfig.builder()
            .withUrl("")
            .withPassword("")
            .withUsername("").build(), null, (DwsInvokeFunction<EventLog>) (eventLog, dwsClient, context) -
> {
        dwsClient.write("test.test")
                .setObject("id", eventLog.getGuid())
                .setObject("name", eventLog.getEventId(), true)
                .commit();
    });
    DataStreamSource<EventLog> source = evn.addSource(new LogSourceFunction());
    source.addSink(dwsSink);
    evn.execute();
```

7.3.3 Flink SQL Job Type

7.3.3.1 Introduction to Flink SQL

DWS-connector can connect to GaussDB(DWS) tables as a source table, result table, and dimension table for Flink jobs by implementing the **DynamicTableSourceFactory** and **DynamicTableSinkFactory** interfaces.

When used as the source table, DWS-Connector can push down **limit** and **where** conditions to the database for execution by implementing the **SupportsLimitPushDown** and **SupportsFilterPushDown** interfaces.

When DWS-Connector is used as the result table, the SQL syntax format may vary slightly in different Flink environments. The precise SQL syntax format is contingent upon the specific environment format.

For details, see **Source Table**, **Result Table**, and **Dimension Table**.

7.3.3.2 Source Table

Syntax

When used as the source table, DWS-Connector can push down **limit** and **where** conditions to the database for execution by implementing the **SupportsLimitPushDown** and **SupportsFilterPushDown** interfaces.

```
create table dwsSource (
  attr_name attr_type
  (',' attr_name attr_type)*
  (','PRIMARY KEY (attr_name, ...) NOT ENFORCED)
  (',' watermark for rowtime_column_name as watermark-strategy_expression)
)
```



```
with (
  'connector' = 'dws',
  'url' = '',
  'tableName' = '',
  'username' = '',
  'password' = ''
);
```

Parameter Description

Table 7-2 Database configurations

Parameter	Description	Default Value
connector	The Flink framework differentiates connector parameters. This parameter is fixed to **dws**.	-
url	Database connection address	-
username	Configured connection user	-
password	Configured password	-
tableName	GaussDB(DWS) table	-

Table 7-3 Query parameters

Parameter	Description	Default Value
fetchSize	The **fetchSize** parameter in the JDBC statement is used to control the number of records returned by the database.	1000
enablePushDown	Enabling condition pushdown will result in the **limit** and **where** conditions being pushed down to the database for execution.	true

Examples

In this example, data is read from the GaussDB(DWS) data source and written to the **Print** result table. The procedure is as follows:

1. Create a table named **dws_order** in GaussDB(DWS).

```
create table public.dws_order(
order_id VARCHAR,
order_channel VARCHAR,
order_time VARCHAR,
pay_amount FLOAT8,
real_pay FLOAT8,
pay_time VARCHAR,
user_id VARCHAR,
user_name VARCHAR,
area_id VARCHAR);
```

2. Insert data into the **dws_order** table.

```
insert into public.dws_order
  (order_id,
  order_channel,
  order_time,
  pay_amount,
  real_pay,
  pay_time,
  user_id,
  user_name,
  area_id) values
  ('202103241000000001', 'webShop', '2021-03-24 10:00:00', '100.00', '100.00', '2021-03-24 10:02:03',
'0001', 'Alice', '330106'),
  ('202103251202020001', 'miniAppShop', '2021-03-25 12:02:02', '60.00', '60.00', '2021-03-25 12:03:00',
'0002', 'Bob', '330110');
```

3. Execute the SQL.

```
CREATE TABLE dwsSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'dws',
  'url' = 'jdbc:gaussdb://DWSIP:DWSPort/DWSdbName',
  'tableName' = 'dws_order',
  'username' = 'DWSUserName',
  'password' = 'DWSPassword'
);
CREATE TABLE printSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'print'
);
insert into printSink select * from dwsSource;
```

4. The command output is shown in the following figure.

```
+I[202103241000000001, webShop, 2021-03-24 10:00:00, 100.0, 100.0, 2021-03-24 10:02:03, 0001, Alice, 330106]
+I[202103251202020001, miniAppShop, 2021-03-25 12:02:02, 60.0, 60.0, 2021-03-25 12:03:00, 0002, Bob, 330110]
```


7.3.3.3 Result Table

Syntax

Different Flink environments may have slightly varying SQL syntax formats. Check the event environment format for more details. The parameter names and values that come after **with** are specific to this document.

```
create table dwsSink (
  attr_name attr_type
  (',' attr_name attr_type)*
  (',PRIMARY KEY (attr_name, ...) NOT ENFORCED)
)
with (
  'connector' = 'dws',
  'url' = '',
  'tableName' = '',
  'username' = '',
  'password' = ''
);
```

Flink SQL Configuration Parameters

NOTE

Primary keys set in Flink SQL are automatically mapped to unique keys in GaussDB(DWS) client. The parameters are released with the client version. The parameter functions are the same as those on the client. The following parameters are the latest parameters.

Table 7-4 Database configurations

Parameter	Description	Default Value
connector	The Flink framework differentiates connector parameters. This parameter is fixed to **dws**.	-
url	Database connection address	-
username	Configured connection user	-
password	Configured password	-
tableName	GaussDB(DWS) table	-

Table 7-5 Connection configurations

Parameter	Description	Default Value
connectionSize	Number of concurrent requests at GaussDB(DWS) client initialization	1
connectionMaxUseTime-Seconds	Number of seconds after which a connection is forcibly released. The unit is second.	3,600 (one hour)
connectionMaxIdleMs	Maximum idle time of a connection, in milliseconds. If the idle time of a connection exceeds the value, the connection is released.	60,000 (one minute)

Table 7-6 Writing parameters

Parameter	Description	Default Value
conflictStrategy	Primary key conflict policy when data is written to a table with a primary key. The options are as follows: • **ignore**: Retain the original data and ignore the updated data. • **update**: Use the non-primary key column in the new data to update the corresponding column in the original data. • **replace**: Replace the original data with the new data. **NOTE** The **UPDATE** and **REPLACE** operations are equivalent when all columns are upserted. When some columns are upserted, the **REPLACE** operation sets the columns that are not contained in the original data to null.	update

Parameter	Description	Default Value
writeMode	Import modes: • **auto**: The system automatically selects a mode. • **copy_merge**: If there is a primary key, data is imported to a temporary table using the **COPY** method and then merged from the temporary table to the target table. If no primary key exists, data is directly imported to the target table using the **COPY** method. • **copy_upsert**: If there is a primary key, data is imported to a temporary table using the **COPY** method, then imported to the target table using the **UPSERT** method. If no primary key exists, data is directly copied to the target table. • **upsert**: If there is a primary key, use **UPSERT SQL** to import data to the database. If there is no primary key, use **INSERT INTO** to import data to the database. • **UPDATE**: Use the **UPDATE WHERE** syntax to update data. If the original table does not have a primary key, you can specify unique keys. A column specified as a unique key does not need to be a unique index, but a non-	auto

Parameter	Description	Default Value
	unique index may impact performance. • **COPY_UPDATE**: Data is imported to a temporary table by the **COPY** method. Temporary tables can be used to accelerate the update using **UPDATE FROM WHERE**. • **UPDATE_AUTO**: If the batch size is less than **copyWriteBatchSize**, **UPDATE** is used. Otherwise, **COPY_UPDATE** is used.	
maxFlushRetryTimes	Maximum number of attempts to import data to the database. If the execution is successful with attempts less than this value, no exception is thrown. The retry interval is 1 second multiplied by the number of attempts.	3
autoFlushBatchSize	Batch size for automatic database update (batch size)	5000
autoFlushMaxInterval	Maximum interval for automatic database update (duration for forming a batch).	5s
copyWriteBatchSize	When **writeMode** is set to **auto**, the batch size in the **COPY** method is used.	5000
ignoreDelete	Whether to ignore **delete** in Flink tasks.	false (The default value is **true** before 1.0.10.)

Parameter	Description	Default Value
ignoreNullWhenUpdate	Whether to ignore the update of columns with null values in Flink. This parameter is valid only when **conflictStrategy** is set to **update**.	false
metadataCacheSeconds	Maximum cache duration of metadata in the system, for example, table definitions (unit: second).	180
copyMode	Format for copying data to the database: ● **CSV**: Data is concatenated into a CSV file and imported to the database. This mode is stable but has low performance. ● **DELIMITER**: Use separators to concatenate data before import it to the database. This mode requires that the data does not contain delimiters.	CSV
createTempTableMode	Temporary table creation methods, which include: ● AS ● LIKE	AS
numberAsEpochMsFor-Datetime	Whether to convert data as a timestamp to the corresponding time type if the database is of the time type and the data source is of the numeric type.	false
stringToDatetimeFormat	Format for converting the data source to the time type if the database is of the time type and the data source is of the string type. If this parameter is set, it is enabled.	null

Parameter	Description	Default Value
sink.parallelism	Flink system parameter, which is used to set the number of concurrent sinks.	Follow the upstream operator.
printDataPk	Whether to print the data primary key when the connector receives data. It can be used for troubleshooting.	false
ignoreUpdateBefore	Whether to ignore **update_before** in Flink tasks. You need to enable this parameter for partial updates on large tables. Otherwise, the update will erase other columns and set them to **null**, since the data is deleted before being inserted.	true

Examples

Here is an example of how to read data from the Kafka data source and store it in the GaussDB(DWS) result table. Each batch can hold a maximum of 30,000 data records and must be stored within 10 seconds.

1. Create the **public.dws_order** table in the GaussDB(DWS) database.

```
create table public.dws_order(
  order_id VARCHAR,
  order_channel VARCHAR,
  order_time VARCHAR,
  pay_amount FLOAT8,
  real_pay FLOAT8,
  pay_time VARCHAR,
  user_id VARCHAR,
  user_name VARCHAR,
  area_id VARCHAR
  );
```

2. The data in the **order_test topic** of the Kafka is used as the data source, the **public.dws_order** is used as the result table, the Kafka data is in JSON format, and the field names correspond to the database field names.

```
CREATE TABLE kafkaSource (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
```



```
  'connector' = 'kafka',
  'topic' = 'order_test',
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',
  'properties.group.id' = 'GroupId',
  'scan.startup.mode' = 'latest-offset',
  'format' = 'json'
);

CREATE TABLE dwsSink (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string
) WITH (
  'connector' = 'dws',
  'url' = 'jdbc:gaussdb://DWSAddress:DWSPort/DWSdbName',
  'tableName' = 'dws_order',
  'username' = 'DWSUserName',
  'password' = 'DWSPassword',
  'autoFlushMaxInterval' = '10s',
  'autoFlushBatchSize' = '30000'
);

insert into dwsSink select * from kafkaSource;
```

3. Insert test data to Kafka.

```
{"order_id":"202103241000000001", "order_channel":"webShop", "order_time":"2021-03-24 10:00:00", "pay_amount":"100.00", "real_pay":"100.00", "pay_time":"2021-03-24 10:02:03", "user_id":"0001", "user_name":"Alice", "area_id":"330106"}
```

4. Wait for 10 seconds and query the result in the GaussDB(DWS) table.

```
select * from dws_order
```

The result is shown in the following figure.

order_id	order_channel	order_time	pay_amount	real_pay	pay_time	user_id	user_name	area_id
202103241000000001	webShop	2021-03-24 10:00:00	100	100	2021-03-24 10:02:03	0001	Alice	330106

FAQs

- Q: What is the proper value of the **writeMode** parameter?

  A: There are two types of service scenarios: **update** and **upsert**. **update** only modifies existing data, while **upsert** updates data if the primary key already exists and adds a new record if it does not. It is recommended to use the auto mode, where the system selects a value based on the data volume. If the data volume is large, increasing the autoFlushBatchSize value can improve the performance of importing data to the database.

- Q: How to set **autoFlushBatchSize** and **autoFlushMaxInterval** properly?

  A: The **autoFlushBatchSize** parameter sets the maximum number of batches that can be stored, while the **autoFlushMaxInterval** parameter sets the maximum time interval for storing batches. These two parameters control the number of batches that can be stored in terms of both time and space.

  – When dealing with small data volumes, setting a value for **autoFlushMaxInterval** can ensure timely updates. But if timeliness is not a top priority, it is better to avoid setting a small value. It is advisable to either stick with the default value or set it to a value greater than or equal to three seconds.

  - The **autoFlushBatchSize** parameter lets you limit the number of data records in a batch. Generally, a higher number of records in a batch results in better data import performance. It is best to set this parameter to a larger value, while considering the service data size and Flink running memory to prevent memory overflow problems.

    For most services, you do not need to set **autoFlushMaxInterval**. Set **autoFlushBatchSize** to **50000**.

- Q: What can I do if a database deadlock occurs?

  A: Deadlocks are classified into row deadlocks and distributed deadlocks.

  - Row lock occurs when multiple updates are made to data with the same primary key. To solve this issue, you can perform **key by** on the data, based on the primary key of the database. This ensures that data with the same primary key is in the same concurrency, eliminating the possibility of concurrent updates and preventing deadlocks. To perform **key by** using Flink SQL, Flink must support it. Both DLI and MRS can implement **key by**. For instance, you can set **key-by-before-sink** to **true** in MRS Flink to implement **key by**. To learn how to use the API, refer to the implementation party for more details. If the API is not usable, it is recommended to import it to the database using the API.
  - Distributed deadlock occurs when column-store tables are concurrently updated, and it cannot be resolved at present. To avoid this, it is recommended to use row-store or **hstore**.

7.3.3.4 Dimension Table

Syntax

```
create table dwsSource (
  attr_name attr_type
  (',' attr_name attr_type)*
)
with (
  'connector' = 'dws',
  'url' = '',
  'tableName' = '',
  'username' = '',
  'password' = ''
);
```

Parameter Description

Table 7-7 Database configurations

Parameter	Description	Default Value
connector	The Flink framework differentiates connector parameters. This parameter is fixed to **dws**.	-
url	Database connection address	-

Parameter	Description	Default Value
username	Configured connection user	-
password	Database user password	-

Table 7-8 Connection configuration parameters

Parameter	Name	Type	Description	Default Value
connectionSize	Size of the read thread pool	int	The number of threads used for operations is equal to the number of database connections, which is also equivalent to the size of the write thread.	1
readBatchSize	Maximum number of **GET** requests that can be combined and submitted at a time.	int	Maximum number of query requests that can be processed in batches once they have been stacked.	128
readBatchQueueSize	Size of the buffer pool for **GET** requests	int	Maximum number of stacked query requests.	256

Parameter	Name	Type	Description	Default Value
readTimeoutMs	Timeout interval of the **GET** request (ms)	int	Setting the value to 0 means there will be no timeout, and it applies to two situations: • The time a user waits to perform a get operation before submitting it to GaussDB(DWS). • The execution of the **get sql** statement times out.	0
readSyncThreadEnable	Whether to enable the thread pool during non-asynchronous query	boolean	Enabling this function will cause future.get() to be blocked asynchronously, while disabling it will result in the main thread being blocked synchronously.	true
lookupScanEnable	Whether to enable scan query	boolean	Whether to enable scan query when the association condition is not all primary key matching. If the value is set to **false**, all join conditions must be primary keys, otherwise, an exception will be thrown.	false

Parameter	Name	Type	Description	Default Value
fetchSize / lookupScanFetchSize	Size of a scan query	int	Maximum number of records returned in a conditional query when using non-full primary key matching mode. By default, **fetchSize** is used, but if **fetchSize** is set to **0**, **lookupScanFetchSize** will be used instead.	1000
lookupScanTimeoutMs	Timeout interval of the **SCAN** request (ms)	int	Timeout limit for a conditional query in non-full primary key matching mode (ms).	60000
lookupAsync	Whether to obtain data in asynchronous mode	boolean	Set the query mode to synchronous or asynchronous.	true

Parameter	Name	Type	Description	Default Value
lookupCacheType	Cache policy	LookupCacheType	Set the following cache policies (case insensitive): • **None**: By default, there is no cache LRU. However, some data in the dimension table may still be cached. When data is retrieved from the source table, the system checks the cache for a match. If no match is found, the system searches the physical dimension table for the data. • **ALL**: All data is cached, which is applicable to small tables that are not frequently updated.	LookupCacheType.LRU
lookupCacheMaxRows	Cache size	long	After selecting the **LRU** cache policy, you can set the cache size.	1000

Parameter	Name	Type	Description	Default Value
lookupCacheExpireAfterAccess	Timeout interval for calculation after reading	Duration	After the **LRU** cache policy is selected, you can set the timeout period to be prolonged after each read. By default, the LRU cache policy does not take effect.	null
lookupCacheExpireAfterWrite	Timeout interval for calculation after data is written	Duration	If the **LRU** cache policy is selected, you can set the timeout period to a fixed value after each write, regardless of whether the data is accessed.	10s
lookupCacheMissingKey	Write data to the cache if the data does not exist	boolean	After the **LRU** cache policy is selected, the dimension table data does not exist and the data is cached.	false
lookupCacheReloadStrategy	Full cache reloading policy	ReloadStrategy	If the **ALL** cache policy is selected, you can set the following data reloading policies: • **PERIODIC**: periodic data reloading. • **TIMED**: scheduled data reloading, in days.	ReloadStrategy.PERIODIC

Parameter	Name	Type	Description	Default Value
lookupCachePeriodicReloadInterval	Data reloading interval	Duration	When the **PERIOD** reloading policy is selected, you can set the full cache reloading interval.	1h
lookupCachePeriodicReloadMode	Data reloading mode	ScheduleMode	When the **PERIOD** reloading policy is selected, you can set the following reloading modes (case insensitive): • **FIXED_DELAY**: The reloading interval is calculated from the end of the previous loading. • **FIXED_RATE**: The reloading interval is calculated from the last loading.	ScheduleMode.FIXED_DELAY
lookupCacheTimedReloadTime	Scheduled scheduling time for data reloading	string	When the **TIMED** reloading policy is selected, you can set the full cache reloading time in ISO-8601 format, For example, **10:15**.	00:00

Parameter	Name	Type	Description	Default Value
lookupCacheTimedReloadIntervalDays	Interval for scheduling data reloading	int	When the **TIMED** reloading policy is selected, you can set the scheduling interval (in days) of the full cache period.	1

Examples

Read data from a Kafka source table, use a GaussDB(DWS) table as the dimension table. Write wide table information generated by the source and dimension tables to the **print** result table. The procedure is as follows:

1. Connect to the GaussDB(DWS) database instance and create a table in GaussDB(DWS) as a dimension table. The table name is **area_info**. The SQL statement is as follows:

```
create table public.area_info(
  area_id VARCHAR,
  area_province_name VARCHAR,
  area_city_name VARCHAR,
  area_county_name VARCHAR,
  area_street_name VARCHAR,
  region_name VARCHAR,
  PRIMARY KEY(area_id)
);
```

2. Connect to the GaussDB(DWS) database and run the following statement to insert test data into the dimension table **area_info**:

```
insert into area_info
  (area_id, area_province_name, area_city_name, area_county_name, area_street_name, region_name)
  values
  ('330102', 'a1', 'b1', 'c1', 'd1', 'e1'),
  ('330106', 'a1', 'b1', 'c2', 'd2', 'e1'),
  ('330108', 'a1', 'b1', 'c3', 'd3', 'e1'),
  ('330110', 'a1', 'b1', 'c4', 'd4', 'e1');
```

3. Run the **flink sql** command to create the source table, result table, and dimension table and run the following SQL statements:

```
CREATE TABLE orders (
  order_id string,
  order_channel string,
  order_time string,
  pay_amount double,
  real_pay double,
  pay_time string,
  user_id string,
  user_name string,
  area_id string,
  proctime as Proctime()
) WITH (
  'connector' = 'kafka',
  'topic' = 'order_test',
  'properties.bootstrap.servers' = 'KafkaAddress1:KafkaPort,KafkaAddress2:KafkaPort',
  'properties.group.id' = 'dws-order',
  'scan.startup.mode' = 'latest-offset',
```



```
 'format' = 'json'
);
--Create the address dimension table.
create table area_info (
    area_id string,
    area_province_name string,
    area_city_name string,
    area_county_name string,
    area_street_name string,
    region_name string
) WITH (
 'connector' = 'dws',
 'url' = 'jdbc:gaussdb://DwsAddress:DwsPort/DwsDbName',
 'tableName' = 'area_info',
 'username' = 'DwsUserName',
 'password' = 'DwsPassword',
 'lookupCacheMaxRows' = '10000',
 'lookupCacheExpireAfterAccess' = '2h'
);
--Create a wide table based on the address dimension table for detailed order information.
create table order_detail(
    order_id string,
    order_channel string,
    order_time string,
    pay_amount double,
    real_pay double,
    pay_time string,
    user_id string,
    user_name string,
    area_id string,
    area_province_name string,
    area_city_name string,
    area_county_name string,
    area_street_name string,
    region_name string
) with (
 'connector' = 'print'

);
insert into order_detail
    select orders.order_id, orders.order_channel, orders.order_time, orders.pay_amount, orders.real_pay,
orders.pay_time, orders.user_id, orders.user_name,
           area.area_id, area.area_province_name, area.area_city_name, area.area_county_name,
           area.area_street_name, area.region_name  from orders
    left join area_info for system_time as of orders.proctime as area on orders.area_id = area.area_id;
```

4. Write data to Kafka.

```
{"order_id":"202103241606060001", "order_channel":"appShop", "order_time":"2021-03-24 16:06:06",
"pay_amount":"200.00", "real_pay":"180.00", "pay_time":"2021-03-24 16:10:06", "user_id":"0001",
"user_name":"Alice", "area_id":"330106"}
{"order_id":"202103251202020001", "order_channel":"miniAppShop", "order_time":"2021-03-25
12:02:02", "pay_amount":"60.00", "real_pay":"60.00", "pay_time":"2021-03-25 12:03:00",
"user_id":"0002", "user_name":"Bob", "area_id":"330110"}
{"order_id":"202103251505050001", "order_channel":"qqShop", "order_time":"2021-03-25 15:05:05",
"pay_amount":"500.00", "real_pay":"400.00", "pay_time":"2021-03-25 15:10:00", "user_id":"0003",
"user_name":"Cindy", "area_id":"330108"}
```

5. The following figure shows the result.

```
 maybeSeekUnvalidated - [Consumer clientId=consumer-dws-order-2, groupId=dws-order] Resetting offset for partition order_test-0 to offset 11.
+I[202103241000000001, webShop, 2021-03-24 10:00:00, 100.0, 100.0, 2021-03-24 10:02:03, 0001, Alice, 330106, a1, b1, c2, d2, e1]
+I[202103241606060001, appShop, 2021-03-24 16:06:06, 200.0, 180.0, 2021-03-24 16:10:06, 0001, Alice, 330106, a1, b1, c2, d2, e1]
+I[202103251202020001, miniAppShop, 2021-03-25 12:02:02, 60.0, 60.0, 2021-03-25 12:03:00, 0002, Bob, 330110, a1, b1, c4, d4, e1]
+I[202103251505050001, null, 2021-03-25 15:05:05, 500.0, 400.0, 2021-03-25 15:10:00, 0003, Cindy, 330108, a1, b1, c3, d3, e1]
```


8 Server Tool

8.1 gs_dump

Context

gs_dump is tool provided by GaussDB(DWS) to export database information. You can export a database or its objects, such as schemas, tables, and views. The database can be the default **postgres** database or a user-specified database.

When **gs_dump** is used to export data, other users can still access the database (readable or writable).

gs_dump can export complete, consistent data. For example, if **gs_dump** is started to export database A at T1, data of the database at that time point will be exported, and modifications on the database after that time point will not be exported.

gs_dump can export database information to a plain-text SQL script file or archive file.

- Plain-text SQL script: It contains the SQL statements required to restore the database. You can use **gsql** to execute the SQL script. With only a little modification, the SQL script can rebuild a database on other hosts or database products.
- Archive file: It contains data required to restore the database. It can be a tar-, directory-, or custom-format archive. For details, see **Table 8-1**. The export result must be used with **gs_restore** to restore the database. The system allows you to select or even to sort the content to import.

Functions

gs_dump can create export files in four formats, which are specified by **-F** or **--format=**, as listed in **Table 8-1**.

Table 8-1 Formats of exported files

Format	Value of -F	Description	Suggestion	Corresponding Import Tool
Plain-text	p	A plain-text script file containing SQL statements and commands. The commands can be executed on **gsql**, a command line terminal, to recreate database objects and load table data.	You are advised to use plain-text export files for small databases.	Before using **gsql** to restore database objects, you can use a text editor to edit the exported plain-text file as required.
Custom	c	A binary file that allows the restoration of all or selected database objects from an exported file.	You are advised to use custom-format archive files for medium or large database.	You can use **gs_restore** to import database objects from a custom-format archive.
Directory	d	A directory containing directory files and the data files of tables and BLOB objects.	-	
.tar	t	A tar-format archive that allows the restoration of all or selected database objects from an exported file. It cannot be further compressed and has an 8-GB limitation on the size of a single table.	-	

NOTE

To reduce the size of an exported file, you can use the **gs_dump** tool to compress it to a plain-text file or custom-format file. By default, a plain-text file is not compressed when generated. When a custom-format archive is generated, a medium level of compression is applied by default. Archived exported files cannot be compressed using **gs_dump**.

Precautions

Do not modify an exported file or its content. Otherwise, restoration may fail.

To ensure the data consistency and integrity, **gs_dump** acquires a share lock on a table to be dumped. If another transaction has acquired a share lock on the table,

gs_dump waits until this lock is released and then locks the table for dumping. If the table cannot be locked within the specified time, the dump fails. You can customize the timeout duration to wait for lock release by specifying the **--lock-wait-timeout** parameter.

Syntax

```
gs_dump [OPTION]... [DBNAME]
```

NOTE

DBNAME does not follow a short or long option. It specifies the database to connect to.

For example:

Specify *DBNAME* without a **-d** option preceding it.

```
gs_dump -p port_number  postgres -f dump1.sql
```

Or

```
export PGDATABASE=postgres
 gs_dump -p port_number -f dump1.sql
```

Environment variable: *PGDATABASE*

Parameter Description

Common parameters:

- -f, --file=FILENAME

  Sends the output to the specified file or directory. If this parameter is omitted, the standard output is generated. If the output format is **(-F c/-F d/-F t)**, the **-f** parameter must be specified. If the value of the **-f** parameter contains a directory, the directory has the read and write permissions to the current user.

- -F, --format=c|d|t|p

  Selects the exported file format. Its format can be:

  – **p|plain**: Generates a text SQL script file. This is the default value.

  – **c|custom**: Outputs a custom-format archive as a directory to be used as the input of **gs_restore**. This is the most flexible output format in which users can manually select it and reorder the archived items during the restore process. An archive in this format is compressed by default.

  – d|directory: A directory containing directory files and the data files of tables and BLOB objects.

  – t|tar: Outputs a tar format as the archive form that is suitable for the input of **gs_restore**. The .tar format is compatible with the directory format. Extracting a .tar archive generates a valid directory-format archive. However, the .tar archive cannot be further compressed and has an 8-GB limitation on the size of a single table. The order of table data items cannot be changed during restoration.

    A .tar archive can be used as input of **gsql**.

- -v, --verbose

  Specifies the verbose mode. If it is specified, **gs_dump** writes detailed object comments and the number of startups/stops to the dump file, and progress messages to standard error.

- -V, --version

  Prints the *gs_dump* version and exits.

- -Z, --compress=0-9

  Specifies the used compression level.

  Value range: 0 to 9

  - **0** indicates no compression.
  - **1** indicates that the compression ratio is the lowest and processing speed the fastest.
  - **9** indicates the compression ratio is the highest and processing speed the slowest.

  For the custom-format archive, this option specifies the compression level of a single table data segment. By default, data is compressed at a medium level. Setting the non-zero compression level will result in that the entire text output files are to be compressed, as if the text has been compressed using the gzip tool, but the default method is non-compression. The .tar archive format does not support compression currently.

- --lock-wait-timeout=TIMEOUT

  Do not keep waiting to obtain shared table locks at the beginning of the dump. Consider it as failed if you are unable to lock a table within the specified time. The timeout duration can be specified in any of the formats accepted by **SET statement_timeout**.

- -?, --help

  Shows help about **gs_dump** parameters and exits.

Dump parameters:

- -a, --data-only

  Generates only the data, not the schema (data definition). Dumps the table data, big objects, and sequence values.

- -b, --blobs

  Specifies a reserved port for function expansion. This parameter is not recommended.

- -c, --clean

  Before writing the command of creating database objects into the backup file, write the command of clearing (deleting) database objects to the backup files. (If no objects exist in the target database, **gs_restore** probably displays some error information.)

  This parameter is used only for the plain-text format. For the archive format, you can specify the option when using **gs_restore**.

- -C, --create

  The backup file content starts with the commands of creating the database and connecting to the created database. (If the script is in this format, any database to be connected is allowed before running the script.)

  This parameter is used only for the plain-text format. For the archive format, you can specify the option when using **gs_restore**.

- -E, --encoding=ENCODING

  Creates a dump file in the specified character set encoding. By default, the dump file is created in the database encoding. (Alternatively, you can set the environment variable **PGCLIENTENCODING** to the required dump encoding.)

- -n, --schema=SCHEMA

  Dumps only schemas matching the schema names. This option contains the schema and all its contained objects. If this option is not specified, all non-system schemas in the target database will be dumped. Multiple schemas can be selected by specifying multiple **-n** options. The schema parameter is interpreted as a pattern according to the same rules used by the **\d** command of **gsql**. Therefore, multiple schemas can also be selected by writing wildcard characters in the pattern. When you use wildcards, quote the pattern to prevent the shell from expanding the wildcards.

  NOTE

  - If **-n** is specified, **gs_dump** does not dump any other database objects that the selected schemas might depend upon. Therefore, there is no guarantee that the results of a specific-schema dump can be automatically restored to an empty database.
  - If **-n** is specified, the non-schema objects are not dumped.

  Multiple schemas can be dumped. Entering **-n** *schemaname* multiple times dumps multiple schemas.

  For example:

  ```
  gs_dump -h host_name -p port_number postgres -f backup/bkp_shl2.sql -n sch1 -n sch2
  ```

  In the preceding example, **sch1** and **sch2** are dumped.

- -N, --exclude-schema=SCHEMA

  Does not dump any tables matching the table pattern. The pattern is interpreted according to the same rules as for **-n**. **-N** can be specified multiple times to exclude schemas matching any of the specified patterns.

  When both **-n** and **-N** are specified, the schemas that match at least one **-n** option but no **-N** is dumped. If **-N** is specified and **-n** is not, the schemas matching **-N** are excluded from what is normally dumped.

  Dump allows you to exclude multiple schemas during dumping.

  Specifies **-N exclude schema name** to exclude multiple schemas while dumping.

  For example:

  ```
  gs_dump -h host_name -p port_number postgres -f backup/bkp_shl2.sql -N sch1 -N sch2
  ```

  In the preceding example, **sch1** and **sch2** will be excluded during the dumping.

- -o, --oids

  Dumps object identifiers (OIDs) as parts of the data in each table. Use this parameter if your application references the OID columns in some way (for example, in a foreign key constraint). If the preceding situation does not occur, do not use this parameter.

- -O, --no-owner

  Do not output commands to set ownership of objects to match the original database. By default, **gs_dump** issues the **ALTER OWNER** or **SET SESSION AUTHORIZATION** command to set ownership of created database objects. These statements will fail when the script is running unless it is started by a system administrator (or the same user that owns all of the objects in the script). To make a script that can be stored by any user and give the user ownership of all objects, specify **-O**.

This parameter is used only for the plain-text format. For the archive format, you can specify the option when using **gs_restore**.

- -s, --schema-only

  Dumps only the object definition (schema) but not data.

- -S, --sysadmin=NAME

  Specifies a reserved port for function expansion. This parameter is not recommended.

- -t, --table=TABLE

  Specifies a list of tables, views, sequences, or foreign tables to be dumped. You can use multiple **-t** parameters or wildcard characters to specify tables.

  When using wildcards to specify dump tables, quote the pattern to prevent the shell from expanding the wildcards.

  The **-n** and **-N** options have no effect when **-t** is used, because tables selected by using **-t** will be dumped regardless of those options, and non-table objects will not be dumped.

  NOTE

  The number of **-t** parameters must be less than or equal to 100.

  If the number of **-t** parameters is greater than 100, you are advised to use the **--include-table-file** parameter to replace some **-t** parameters.

  If **-t** is specified, **gs_dump** does not dump any other database objects that the selected tables might depend upon. Therefore, there is no guarantee that the results of a specific-table dump can be automatically restored to an empty database.

  -t tablename only dumps visible tables in the default search path. **-t '*.tablename'** dumps *tablename* tables in all the schemas of the dumped database. **-t schema.table** dumps tables in a specific schema.

  -t tablename does not export the trigger information from a table.

  For example:

  ```
  gs_dump -h host_name -p port_number postgres -f backup/bkp_shl2.sql -t schema1.table1 -t schema2.table2
  ```

  In the preceding example, **schema1.table1** and **schema2.table2** are dumped.

- --include-table-file=FILENAME

  Specifies the table file to be dumped.

- -T, --exclude-table=TABLE

  Specifies a list of tables, views, sequences, or foreign tables not to be dumped. You can use multiple **-t** parameters or wildcard characters to specify tables.

  When **-t** and **-T** are input, the object will be stored in **-t** list not **-T** table object.

  For example:

  ```
  gs_dump -h host_name -p port_number postgres -f backup/bkp_shl2.sql -T table1 -T table2
  ```

  In the preceding example, **table1** and **table2** are excluded from the dumping.

- --exclude-table-file=FILENAME

  Specifies the table file to be dumped.

NOTE

Same as **--include-table-file**, the content format of this parameter is as follows:

schema1.table1

schema2.table2

...

- -x, --no-privileges|--no-acl

  Prevents the dumping of access permissions (grant/revoke commands).

- --column-inserts|--attribute-inserts

  Exports data by running the **INSERT** command with explicit column names {INSERT INTO table (column, ...) VALUES ...}. This will cause a slow restoration. However, since this option generates an independent command for each row, an error in reloading a row causes only the loss of the row rather than the entire table content.

- --disable-dollar-quoting

  Disables the use of dollar sign ($) for function bodies, and forces them to be quoted using the SQL standard string syntax.

- --disable-triggers

  Specifies a reserved port for function expansion. This parameter is not recommended.

- --exclude-table-data=TABLE

  Does not dump data that matches any of table patterns. The pattern is interpreted according to the same rules as for **-t**.

  --exclude-table-data can be entered more than once to exclude tables matching any of several patterns. When the user needs the specified table definition rather than data in the table, this option is helpful.

  To exclude data of all tables in the database, see **--schema-only**.

- --inserts

  Dumps data when the **INSERT** statement (rather than **COPY**) is issued. This will cause a slow restoration.

  However, since this option generates an independent command for each row, an error in reloading a row causes only the loss of the row rather than the entire table content. The restoration may fail if you rearrange the column order. The **--column-inserts** option is unaffected against column order changes, though even slower.

- --no-security-labels

  Specifies a reserved port for function expansion. This parameter is not recommended.

- --no-tablespaces

  This parameter is no longer used in 8.2.0.100 and is only kept for compatibility with earlier versions.

  Does not issue commands to select tablespaces. All the objects will be created during the restoration process, no matter which tablespace is selected when using this option.

  This parameter is used only for the plain-text format. For the archive format, you can specify the option when using **gs_restore**.

- --no-unlogged-table-data

  Specifies a reserved port for function expansion. This parameter is not recommended.

- --non-lock-table

  Specifies a reserved port for function expansion. This parameter is not recommended.

- --quote-all-identifiers

  Forcibly quotes all identifiers. This parameter is useful when you dump a database for migration to a later version, in which additional keywords may be introduced.

- --section=SECTION

  Specifies dumped name sections (pre-data, data, or post-data).

- --serializable-deferrable

  Uses a serializable transaction for the dump to ensure that the used snapshot is consistent with later database status. Perform this operation at a time point in the transaction flow, at which everything is normal. This ensures successful transaction and avoids serialization failures of other transactions, which requires serialization again.

  This option has no benefits for disaster recovery. During the upgrade of the original database, load a database copy as a report or other shared read-only dump is helpful. The option does not exist, dump reveals a status which is different from the submitted sequence status of any transaction.

  This option will make no difference if there are no active read-write transactions when **gs_dump** is started. If the read-write transactions are in active status, the dump start time will be delayed for an uncertain period.

- --use-set-session-authorization

  Specifies that the standard SQL **SET SESSION AUTHORIZATION** command rather than **ALTER OWNER** is returned to ensure the object ownership. This makes dumping more standard. However, if a dump file contains objects that have historical problems, restoration may fail. A dump using **SET SESSION AUTHORIZATION** requires the system administrator rights, whereas **ALTER OWNER** requires lower permissions.

- --with-encryption=AES128

  Specifies that dumping data needs to be encrypted using AES128.

- --with-key=KEY

  Specifies that the key length of AES128 must be 16 bytes.

- --include-nodes

  Includes the **TO NODE** or **TO GROUP** statement in the dumped **CREATE TABLE** or **CREATE FOREIGN TABLE** statement. This parameter is valid only for HDFS and foreign tables.

- --include-extensions

  Includes extensions in the dump.

- --include-depend-objs

  Includes information about the objects that depend on the specified object in the backup result. This parameter takes effect only if the **-t** or **--include-table-file** parameter is specified.

- --exclude-self

  Excludes information about the specified object from the backup result. This parameter takes effect only if the **-t** or **--include-table-file** parameter is specified.

- --cstore-fine-disaster (Discarded)

  If this parameter is selected and a table whose **fine_disaster_table_role** is **primary** is dumped, a table definition whose **fine_disaster_table_role** is **standby** is generated.

  This parameter is discarded in 8.2.1.

- --only-publications

  If this parameter is specified, only the definitions of all publications (publication) in the current database are dumped. This parameter is supported by version 8.2.1 or later clusters.

- --dont-overwrite-file

  The existing files in plain-text, .tar, and custom formats will be overwritten. This parameter is not used for the directory format.

  For example:

  Assume that the **backup.sql** file exists in the current directory. If you specify **-f backup.sql** in the input command, and the **backup.sql** file is generated in the current directory, the original file will be overwritten.

  If the backup file already exists and **--dont-overwrite-file** is specified, an error will be reported with the message that the dump file exists.

  ```
  gs_dump -p port_number postgres -f backup.sql -F plain --dont-overwrite-file
  ```

  NOTE

  - The **-s/--schema-only** and **-a/--data-only** parameters do not coexist.
  - The **-c/--clean** and **-a/--data-only** parameters do not coexist.
  - **--inserts/--column-inserts** and **-o/--oids** do not coexist, because **OIDS** cannot be set using the **INSERT** statement.
  - **--role** must be used in conjunction with **--rolepassword**.
  - **--binary-upgrade-usermap** must be used in conjunction with **--binary-upgrade**.
  - **--include-depend-objs**/**--exclude-self** takes effect only when **-t**/**--include-table-file** is specified.
  - **--exclude-self** must be used with **--include-depend-objs**.

Connection parameters:

- -h, --host=HOSTNAME

  Specifies the host name. If the value begins with a slash (/), it is used as the directory for the UNIX domain socket. The default is taken from the PGHOST environment variable (if available). Otherwise, a Unix domain socket connection is attempted.

  This parameter is used only for defining names of the hosts outside a cluster. The names of the hosts inside the cluster must be 127.0.0.1.

  Example: the host name

  Environment Variable: *PGHOST*

- -p, --port=PORT

  Specifies the host port.

Environment variable: *PGPORT*

- -U, --username=NAME

  Specifies the user name of the host to connect to.

  Environment variable: *PGUSER*

- -w, --no-password

  Never issue a password prompt. The connection attempt fails if the host requires password verification and the password is not provided in other ways. This parameter is useful in batch jobs and scripts in which no user password is required.

- -W, --password=PASSWORD

  Specifies the user password to connect to. If the host uses the trust authentication policy, the administrator does not need to enter the **-W** option. If the **-W** option is not provided and you are not a system administrator, the Dump Restore tool will ask you to enter a password.

- --role=ROLENAME

  Specifies a role name to be used for creating the dump. If this option is selected, the **SET ROLE** command will be issued after the database is connected to **gs_dump**. It is useful when the authenticated user (specified by **-U**) lacks the permissions required by **gs_dump**. It allows the user to switch to a role with the required permissions. Some installations have a policy against logging in directly as a system administrator. This option allows dumping data without violating the policy.

- --rolepassword=ROLEPASSWORD

  Password for the role

Description

Scenario 1

If your database cluster has any local additions to the template1 database, restore the output of **gs_dump** into an empty database with caution. Otherwise, you are likely to obtain errors due to duplicate definitions of the added objects. To create an empty database without any local additions, copy data from template0 rather than template1. Example:

```
CREATE DATABASE foo WITH TEMPLATE template0;
```

The .tar format file size must be smaller than 8 GB. (This is the tar file format limitations.) The total size of a .tar archive and any of the other output formats are not limited, except possibly by the OS.

The dump file generated by **gs_dump** does not contain the statistics used by the optimizer to make execution plans. Therefore, you are advised to run **ANALYZE** after restoring from a dump file to ensure optimal performance. The dump file does not contain any **ALTER DATABASE ... SET** commands; these settings are dumped by **gs_dumpall**, along with database users and other installation settings.

Scenario 2

When the value of **SEQUENCE** reaches the maximum or minimum value, backing up the value of **SEQUENCE** using **gs_dump** will exit due to an execution error. Handle the problem by referring to the following example:

1. The value of **SEQUENCE** reaches the maximum value, but the maximum value is less than **2^63-2**.

Error message example:

Object defined by sequence

```
CREATE SEQUENCE seq INCREMENT 1 MINVALUE 1 MAXVALUE 3 START WITH 1;
```

Perform the **gs_dump** backup.

```
gs_dump -U dbadmin -W {password} -p 37300 postgres -t PUBLIC.seq -f backup/MPPDB_backup.sql
gs_dump[port='37300'][postgres][2019-12-27 15:09:49]: The total objects number is 337.
gs_dump[port='37300'][postgres][2019-12-27 15:09:49]: WARNING:  get invalid xid from GTM because connection is not established
gs_dump[port='37300'][postgres][2019-12-27 15:09:49]: WARNING:  Failed to receive GTM rollback transaction response  for aborting prepared (null).
gs_dump: [port='37300'] [postgres] [archiver (db)] [2019-12-27 15:09:49] query failed: ERROR:  Can not connect to gtm when getting gxid, there is a connection error.
gs_dump: [port='37300'] [postgres] [archiver (db)] [2019-12-27 15:09:49] query was: RELEASE bfnextval
```

Handling procedure:

Run the following SQL statement to connect to the PostgreSQL database and change the maximum value of **sequence seq1**:

```
gsql -p 37300 postgres -r -c "ALTER SEQUENCE PUBLIC.seq MAXVALUE 10;"
```

Use the dump tool to back up the data.

```
gs_dump -U dbadmin -W {password} -p 37300 postgres -t PUBLIC.seq -f backup/MPPDB_backup.sql
gs_dump[port='37300'][postgres][2019-12-27 15:10:53]: The total objects number is 337.
gs_dump[port='37300'][postgres][2019-12-27 15:10:53]: [100.00%] 337 objects have been dumped.
gs_dump[port='37300'][postgres][2019-12-27 15:10:53]: dump database postgres successfully
gs_dump[port='37300'][postgres][2019-12-27 15:10:53]: total time: 230  ms
```

2. The value of **SEQUENCE** reaches the minimum or the maximum value of **2^63-2**.

The **gs_dump** command does not support backup of the **SEQUENCE** value in this scenario.

NOTE

The SQL end does not support the modification of **MAXVALUE** when **SEQUENCE** reaches the maximum value of **2^63-2** or the modification of **MINVALUE** when **SEQUENCE** reaches the minimum value.

Scenario 3

gs_dump is mainly used to export metadata of the entire database. The performance of exporting a single table is optimized, but the performance of exporting multiple tables is poor. If multiple tables need to be exported, you are advised to export them one by one. Example:

```
gs_dump -U dbadmin -W {password} -p 37300 postgres -t public.table01 -s -f backup/table01.sql
gs_dump -U dbadmin -W {password} -p 37300 postgres -t public.table02 -s -f backup/table02.sql
```

When services are stopped or during off-peak hours, you can increase the value of **--non-lock-table** to improve the **gs_dump** performance. Example:

```
gs_dump -U dbadmin -W {password} -p 37300 postgres -t public.table03 -s --non-lock-table -f backup/table03.sql
```

Examples

Use **gs_dump** to dump a database as a SQL text file or a file in other formats.

In the following examples, **password** indicates the password configured by the database user. **backup/MPPDB_backup.sql** indicates an exported file where **backup** indicates the relative path of the current directory. **37300** indicates the port ID of the database server. **postgres** indicates the name of the database to be accessed.

NOTE

Before exporting files, ensure that the directory exists and you have the read and write permissions on the directory.

Example 1: Use **gs_dump** to export the full information of the **postgres** database. The exported **MPPDB_backup.sql** file is in plain-text format.

```
gs_dump -U dbadmin -W {password} -f backup/MPPDB_backup.sql -p 37300 postgres -F p
gs_dump[port='37300'][postgres][2018-06-27 09:49:17]: The total objects number is 356.
gs_dump[port='37300'][postgres][2018-06-27 09:49:17]: [100.00%] 356 objects have been dumped.
gs_dump[port='37300'][postgres][2018-06-27 09:49:17]: dump database postgres successfully
gs_dump[port='37300'][postgres][2018-06-27 09:49:17]: total time: 1274  ms
```

Use **gsql** to import data from the export plain-text file.

Example 2: Use **gs_dump** to export the full information of the **postgres** database. The exported **MPPDB_backup.tar** file is in .tar format.

```
gs_dump -U dbadmin -W {password} -f backup/MPPDB_backup.tar -p 37300 postgres -F t
gs_dump[port='37300'][postgres][2018-06-27 10:02:24]: The total objects number is 1369.
gs_dump[port='37300'][postgres][2018-06-27 10:02:53]: [100.00%] 1369 objects have been dumped.
gs_dump[port='37300'][postgres][2018-06-27 10:02:53]: dump database postgres successfully
gs_dump[port='37300'][postgres][2018-06-27 10:02:53]: total time: 50086  ms
```

Example 3: Use **gs_dump** to export the full information of the **postgres** database. The exported **MPPDB_backup.dmp** file is in custom format.

```
gs_dump -U dbadmin -W {password} -f backup/MPPDB_backup.dmp -p 37300 postgres -F c
gs_dump[port='37300'][postgres][2018-06-27 10:05:40]: The total objects number is 1369.
gs_dump[port='37300'][postgres][2018-06-27 10:06:03]: [100.00%] 1369 objects have been dumped.
gs_dump[port='37300'][postgres][2018-06-27 10:06:03]: dump database postgres successfully
gs_dump[port='37300'][postgres][2018-06-27 10:06:03]: total time: 36620  ms
```

Example 4: Use **gs_dump** to export the full information of the **postgres** database. The exported **MPPDB_backup** file is in directory format.

```
gs_dump -U dbadmin -W {password} -f backup/MPPDB_backup -p 37300  postgres -F d
gs_dump[port='37300'][postgres][2018-06-27 10:16:04]: The total objects number is 1369.
gs_dump[port='37300'][postgres][2018-06-27 10:16:23]: [100.00%] 1369 objects have been dumped.
gs_dump[port='37300'][postgres][2018-06-27 10:16:23]: dump database postgres successfully
gs_dump[port='37300'][postgres][2018-06-27 10:16:23]: total time: 33977  ms
```

Example 5: Use **gs_dump** to export the information of the **postgres** database, excluding the information of the table specified in the **/home/MPPDB_temp.sql** file. The exported **MPPDB_backup.sql** file is in plain-text format.

```
gs_dump -U dbadmin -W {password} -p 37300 postgres --exclude-table-file=/home/MPPDB_temp.sql -f
backup/MPPDB_backup.sql
gs_dump[port='37300'][postgres][2018-06-27 10:37:01]: The total objects number is 1367.
gs_dump[port='37300'][postgres][2018-06-27 10:37:22]: [100.00%] 1367 objects have been dumped.
gs_dump[port='37300'][postgres][2018-06-27 10:37:22]: dump database postgres successfully
gs_dump[port='37300'][postgres][2018-06-27 10:37:22]: total time: 37017  ms
```

Example 6: Use **gs_dump** to export only the information about the views that depend on the **testtable** table. Create another **testtable** table, and then restore the views that depend on it.

Back up only the views that depend on the **testtable** table.


```
gs_dump -s -p 37300 postgres -t PUBLIC.testtable --include-depend-objs --exclude-self -f backup/
MPPDB_backup.sql -F p
gs_dump[port='37300'][postgres][2018-06-15 14:12:54]: The total objects number is 331.
gs_dump[port='37300'][postgres][2018-06-15 14:12:54]: [100.00%] 331 objects have been dumped.
gs_dump[port='37300'][postgres][2018-06-15 14:12:54]: dump database postgres successfully
gs_dump[port='37300'][postgres][2018-06-15 14:12:54]: total time: 327  ms
```

Change the name of the **testtable** table.

```
gsql -p 37300 postgres -r -c "ALTER TABLE PUBLIC.testtable RENAME TO testtable_bak;"
```

Create a **testtable** table.

```
CREATE TABLE PUBLIC.testtable(a int, b int, c int);
```

Restore the views for the new **testtable** table.

```
gsql -p 37300 postgres -r -f backup/MPPDB_backup.sql
```

Helpful Links

gs_dumpall and **gs_restore**

8.2 gs_dumpall

Background

gs_dumpall is a tool provided by GaussDB(DWS) to export all database information, including the data of the default **postgres** database, data of user-specified databases, and global objects of all databases in a cluster.

When **gs_dumpall** is used to export data, other users still can access the databases (readable or writable) in a cluster.

gs_dumpall can export complete, consistent data. For example, if **gs_dumpall** is started to export all databases from a cluster at T1, data of the databases at that time point will be exported, and modifications on the databases after that time point will not be exported.

To export all databases in a cluster:

- **gs_dumpall** exports all global objects, including information about database users and groups, tablespaces, and attributes (for example, global access permissions).
- **gs_dumpall** invokes **gs_dump** to export SQL scripts, which contain all the SQL statements required for restoring databases.

Both of the preceding exported files are plain-text SQL scripts. Use **gsql** to execute them to restore databases.

Precautions

- Do not modify an exported file or its content. Otherwise, restoration may fail.
- To ensure the data consistency and integrity, **gs_dumpall** sets a share lock for a table to be dumped. If a share lock has been set for the table in other transactions, **gs_dumpall** locks the table after it is released. If the table cannot be locked in the specified time, the dump will fail. You can customize

the timeout duration to wait for lock release by specifying the **--lock-wait-timeout** parameter.

- During an export, **gs_dumpall** reads all tables in a database. Therefore, you need to connect to the database as a cluster administrator to export a complete file. When using **gsql** to execute the SQL scripts, you need the administrator permissions so that you can add users and user groups, and create databases.

Syntax

```
gs_dumpall [OPTION]...
```

Parameter Description

Common parameters:

- -f, --filename=FILENAME

  Sends the output to the specified file. If this parameter is omitted, the standard output is used.

- -v, --verbose

  Specifies the verbose mode. This will cause **gs_dumpall** to output detailed object comments and start/stop times to the dump file, and progress messages to standard error.

- -V, --version

  Prints the version information for **gs_dumpall** and exits.

- --lock-wait-timeout=TIMEOUT

  Do not keep waiting to obtain shared table locks at the beginning of the dump. Consider it as failed if you are unable to lock a table within the specified time. The timeout duration can be specified in any of the formats accepted by **SET statement_timeout**.

- -?, --help

  Shows help about the command line parameters for **gs_dumpall** and exits.

Dump parameters:

- -a, --data-only

  Dumps only the data, not the schema (data definition).

- -c, --clean

  Runs SQL statements to delete databases before rebuilding them. Statements for dumping roles and tablespaces are added.

- -g, --globals-only

  Dumps only global objects (roles and tablespaces) but no databases.

- -o, --oids

  Dumps object identifiers (OIDs) as parts of the data in each table. Use this parameter if your application references the OID columns in some way (for example, in a foreign key constraint). If the preceding situation does not occur, do not use this parameter.

- -O, --no-owner

Do not output commands to set ownership of objects to match the original database. By default, **gs_dumpall** issues the **ALTER OWNER** or **SET SESSION AUTHORIZATION** statement to set ownership of created schema elements. These statements will fail when the script is running unless it is started by a system administrator (or the same user that owns all of the objects in the script). To make a script that can be stored by any user and give the user ownership of all objects, specify **-O**.

- -r, --roles-only

  Dumps only roles but not databases or tablespaces.

- -s, --schema-only

  Dumps only the object definition (schema) but not data.

- -S, --sysadmin=NAME

  Name of the system administrator during the dump.

- -t, --tablespaces-only

  Dumps only tablespaces but not databases or roles.

- -x, --no-privileges

  Prevents the dumping of access permissions (grant/revoke commands).

- --column-inserts|--attribute-inserts

  Exports data by running the **INSERT** command with explicit column names {INSERT INTO table (column, ...) VALUES ...}. This will cause a slow restoration. However, since this option generates an independent command for each row, an error in reloading a row causes only the loss of the row rather than the entire table content.

- --disable-dollar-quoting

  Disables the use of dollar sign ($) for function bodies, and forces them to be quoted using the SQL standard string syntax.

- --disable-triggers

  Specifies a reserved port for function expansion. This parameter is not recommended.

- --inserts

  Dumps data when the **INSERT** statement (rather than **COPY**) is issued. This will cause a slow restoration. The restoration may fail if you rearrange the column order. The **--column-inserts** parameter is safer against column order changes, though even slower.

- --no-security-labels

  Specifies a reserved port for function expansion. This parameter is not recommended.

- --no-tablespaces

  This parameter is no longer used in 8.2.0.100 and is only kept for compatibility with earlier versions.

  Do not output statements to create tablespaces or select tablespaces for objects. All the objects will be created during the restoration process, no matter which tablespace is selected when using this option.

- --no-unlogged-table-data

  Specifies a reserved port for function expansion. This parameter is not recommended.

- --quote-all-identifiers

  Forcibly quotes all identifiers. This parameter is useful when you dump a database for migration to a later version, in which additional keywords may be introduced.

- --dont-overwrite-file

  Do not overwrite the current file.

- --use-set-session-authorization

  Specifies that the standard SQL **SET SESSION AUTHORIZATION** command rather than **ALTER OWNER** is returned to ensure the object ownership. This makes dumping more standard. However, if a dump file contains objects that have historical problems, restoration may fail. A dump using **SET SESSION AUTHORIZATION** requires the system administrator rights, whereas **ALTER OWNER** requires lower permissions.

- --with-encryption=AES128

  Specifies that dumping data needs to be encrypted using AES128.

- --with-key=KEY

  Specifies that the key length of AES128 must be 16 bytes.

- --include-extensions

  Backs up all CREATE EXTENSION statements if the **include-extensions** parameter is set.

- --include-templatedb

  Includes template databases during the dump.

- --dump-nodes

  Includes nodes and Node Groups during the dump.

- --include-nodes

  Includes the **TO NODE** statement in the dumped **CREATE TABLE** statement.

- --include-buckets

  Specifies a reserved port for function expansion. This parameter is not recommended.

- --dump-wrm

  Includes workload resource manager (resource pool, load group, and load group mapping) during the dump.

- --binary-upgrade

  Specifies a reserved port for function expansion. This parameter is not recommended.

- --binary-upgrade-usermap="USER1=USER2"

  Specifies a reserved port for function expansion. This parameter is not recommended.

- --tablespaces-postfix

  Specifies a reserved port for function expansion. This parameter is not recommended.

- --parallel-jobs

  Specifies the number of concurrent backup processes. The value range is 1-1000.

NOTE

- The **-g/--globals-only** and **-r/--roles-only** parameters do not coexist.
- The **-g/--globals-only** and **-t/--tablespaces-only** parameters do not coexist.
- The **-r/--roles-only** and **-t/--tablespaces-only** parameters do not coexist.
- The **-s/--schema-only** and **-a/--data-only** parameters do not coexist.
- The **-r/--roles-only** and **-a/--data-only** parameters do not coexist.
- The **-t/--tablespaces-only** and **-a/--data-only** parameters do not coexist.
- The **-g/--globals-only** and **-a/--data-only** parameters do not coexist.
- **--tablespaces-postfix** must be used in conjunction with **--binary-upgrade**.
- **--parallel-jobs** must be used in conjunction with **-f/--file**.

Connection parameters:

- -h, --host

  Specifies the host name. If the value begins with a slash (/), it is used as the directory for the UNIX domain socket. The default value is taken from the *PGHOST* environment variable. If it is not set, a UNIX domain socket connection is attempted.

  This parameter is used only for defining names of the hosts outside a cluster. The names of the hosts inside the cluster must be 127.0.0.1.

  Environment Variable: *PGHOST*

- -l, --database

  Specifies the name of the database connected to dump all objects and discover other databases to be dumped. If this parameter is not specified, the **postgres** database will be used. If the **postgres** database does not exist, **template1** will be used.

- -p, --port

  Specifies the TCP port listened to by the server or the local UNIX domain socket file name extension to ensure a correct connection. The default value is the *PGPORT* environment variable.

  Environment variable: *PGPORT*

- -U, --username

  Specifies the user name to connect to.

  Environment variable: *PGUSER*

- -w, --no-password

  Never issue a password prompt. The connection attempt fails if the host requires password verification and the password is not provided in other ways. This parameter is useful in batch jobs and scripts in which no user password is required.

- -W, --password

  Specifies the user password to connect to. If the host uses the trust authentication policy, the administrator does not need to enter the **-W** option. If the **-W** option is not provided and you are not a system administrator, the Dump Restore tool will ask you to enter a password.

- --role

  Specifies a role name to be used for creating the dump. This option causes **gs_dumpall** to issue the **SET ROLE** statement after connecting to the

database. It is useful when the authenticated user (specified by **-U**) lacks the permissions required by **gs_dumpall**. It allows the user to switch to a role with the required permissions. Some installations have a policy against logging in directly as a system administrator. This option allows dumping data without violating the policy.

- --rolepassword

  Specifies the password of the specific role.

Description

The **gs_dumpall** internally invokes **gs_dump**. For details about the diagnosis information, see **gs_dump**.

Once **gs_dumpall** is restored, run ANALYZE on each database so that the optimizer can provide useful statistics.

gs_dumpall requires all needed tablespace directories to exit before the restoration. Otherwise, database creation will fail if the databases are in non-default locations.

Examples

Run **gs_dumpall** to export all databases from a cluster at a time.

NOTE

gs_dumpall supports only plain-text format export. Therefore, only **gsql** can be used to restore a file exported using **gs_dumpall**.

```
gs_dumpall -f backup/bkp2.sql -p 37300
gs_dump[port='37300'][dbname='postgres'][2018-06-27 09:55:09]: The total objects number is 2371.
gs_dump[port='37300'][dbname='postgres'][2018-06-27 09:55:35]: [100.00%] 2371 objects have been dumped.
gs_dump[port='37300'][dbname='postgres'][2018-06-27 09:55:46]: dump database dbname='postgres' successfully
gs_dump[port='37300'][dbname='postgres'][2018-06-27 09:55:46]: total time: 55567  ms
gs_dumpall[port='37300'][2018-06-27 09:55:46]: dumpall operation successful
gs_dumpall[port='37300'][2018-06-27 09:55:46]: total time: 56088  ms
```

Helpful Links

gs_dump and **gs_restore**

8.3 gs_restore

Context

gs_restore is a tool provided by GaussDB(DWS) to import data that was exported using **gs_dump**. It can also be used to import files that were exported using **gs_dump**.

It has the following functions:

- Imports data to the database.

  If a database is specified, data is imported in the database. For parallel import, the password for connecting to the database is required.

- Imports data to the script file.

  If the database storing imported data is not specified, a script containing the SQL statement to recreate the database is created and written to a file or standard output. This script output is equivalent to the plain text output format of **gs_dump**.

Syntax

```
gs_restore [OPTION]... FILE
```

NOTE

- The **FILE** does not have a short or long parameter. It is used to specify the location for the archive files.
- The **dbname** or **-l** parameter is required as prerequisites. Users cannot enter **dbname** and **-l** parameters at the same time.
- **gs_restore** incrementally imports data by default. To prevent data exception caused by multiple import jobs, use the **-e** and **-c** parameters during the jobs. In this way, existing database objects in a target database are deleted before import; and errors during import will be ignored to proceed the import and the error information will be displayed after the import.

Parameter Description

Common parameters:

- -d, --dbname=NAME

  Connects to the **dbname** database and imports data to the database.

- -f, --file=FILENAME

  Specifies the output file for the generated script, or uses the output file in the list specified using **-l**.

  The default is the standard output.

  NOTE

  -f cannot be used in conjunction with **-d**.

- -F, --format=c|d|t

  Specifies the format of the archive. The format does not need to be specified because the *gs_restore* determines the format automatically.

  Value range:

  – **c/custom**: The archive form is the customized format in section 4.21-gs_dump.

  – **d/directory**: The archive form is a directory archive format.

  – t/tar: The archive form is a tar archive format.

- -l, --list

  Lists the forms of the archive. The operation output can be used for the input of the **-L** parameter. If filtering parameters, such as **-n** or **-t**, are used together with **-l**, they will restrict the listed items.

- -v, --verbose

  Specifies the verbose mode.

- -V, --version

Prints the **gs_restore** version and exits.

- -?, --help

  Shows help information about the parameters of **gs_restore** and exits.

Import parameters

- -a, -data-only

  Imports only the data, not the schema (data definition). **gs_restore** incrementally imports data.

- -c, --clean

  Cleans (deletes) existing database objects in the database to be restored before recreating them.

- -C, --create

  Creates the database before importing data to it. When this parameter is used, the database named with **-d** is used to issue the initial **CREATE DATABASE** command. All data is imported to the database that appears in the archive files.

- -e, --exit-on-error

  Exits if an error occurs when you send the SQL statement to the database. If you do not exit, the commands will still be sent and error information will be displayed when the import ends.

- -I, --index=NAME

  Imports only the definition of the specified index. Multiple indexes can be imported. Enter *-I index* multiple times to import multiple indexes.

  For example:

  ```
  gs_restore -h host_name -p port_number -d gaussdb -I Index1 -I Index2 backup/MPPDB_backup.tar
  ```

  In this example, *Index1* and *Index2* will be imported.

- -j, --jobs=NUM

  Specifies the number of concurrent, the most time-consuming jobs of **gs_restore** (such as loading data, creating indexes, or creating constraints). This parameter can greatly reduce the time to import a large database to a server running on a multiprocessor machine.

  Each job is one process or one thread, depending on the OS; and uses a separate connection to the server.

  The optimal value of this option depends on the hardware settings of the server, the client, the network, the number of CPU cores, and hard disk settings. It is recommended that the parameter be set to the number of CPU cores on the server. In addition, a larger value can also lead to faster import in many cases. However, an overly large value will lead to decreased performance because of thrashing.

  This parameter supports custom-format archives only. The input file must be a regular file (not the pipe file). This parameter can be ignored when you select the script method rather than connect to a database server. In addition, multiple jobs cannot be used in conjunction with the **--single-transaction** parameter.

- -L, --use-list=FILENAME

  Imports only archive elements that are listed in **list-file** and imports them in the order that they appear in the file. If filtering parameters, such as **-n** or **-t**,

are used in conjunction with **-L**, they will further limit the items to be imported.

list-file is normally created by editing the output of a previous **-l** parameter. File lines can be moved or removed, and can also be commented out by placing a semicolon (;) at the beginning of the row. An example is provided in this document.

- -n, --schema=NAME

  Restores only objects that are listed in schemas.

  This parameter can be used in conjunction with the **-t** parameter to import a specific table.

  Entering **-n** *schemaname* multiple times can import multiple schemas.

  For example:

  ```
  gs_restore -h host_name -p port_number -d gaussdb -n sch1 -n sch2 backup/MPPDB_backup.tar
  ```

  In this example, **sch1** and **sch2** will be imported.

- -O, --no-owner

  Do not output commands to set ownership of objects to match the original database. By default, **gs_restore** issues the **ALTER OWNER** or **SET SESSION AUTHORIZATION** statement to set ownership of created schema elements. Unless the system administrator or the user who has all the objects in the script initially accesses the database. Otherwise, the statement will fail. Any user name can be used for the initial connection using **-O**, and this user will own all the created objects.

- -P, --function=NAME(args)

  Imports only listed functions. You need to correctly spell the function name and the parameter based on the contents of the dump file in which the function exists.

  Entering **-P** alone means importing all function-name(args) functions in a file. Entering **-P** with **-n** means importing the function-name(args) functions in a specified schema. Entering **-P** multiple times and using **-n** once means that all imported functions are in the **-n** schema by default.

  You can enter **-n schema-name -P 'function-name(args)'** multiple times to import functions in specified schemas.

  For example:

  ```
  ./gs_restore -h host_name -p port_number -d gaussdb -n test1 -P 'Func1(integer)' -n test2 -P 'Func2(integer)' backup/MPPDB_backup.tar
  ```

  In this example, both **Func1 (i integer)** in the **test1** schema and **Func2 (j integer)** in the **test2** schema will be imported.

- -s, --schema-only

  Imports only schemas (data definitions), instead of data (table content). The current sequence value will not be imported.

- -S, --sysadmin=NAME

  Specifies a reserved port for function expansion. This parameter is not recommended.

- -t, --table=NAME

  Imports only listed table definitions or data, or both. This parameter can be used in conjunction with the **-n** parameter to specify a table object in a schema. When **-n** is not entered, the default schema is PUBLIC. Entering **-n**

schemaname **-t** *tablename* multiple times can import multiple tables in a specified schema.

For example:

Import **table1** in the **PUBLIC** schema.

```
gs_restore -h host_name -p port_number -d gaussdb -t table1 backup/MPPDB_backup.tar
```

Import **test1** in the **test1** schema and **test2** in the **test2** schema.

```
gs_restore -h host_name -p port_number -d gaussdb -n test1 -t test1 -n test2 -t test2 backup/MPPDB_backup.tar
```

Import **table1** in the **PUBLIC** schema and **test1** in the **test1** schema.

```
gs_restore -h host_name -p port_number -d gaussdb -n PUBLIC -t table1 -n test1 -t table1 backup/MPPDB_backup.tar
```

NOTICE

-t does not support the **schema_name.table_name** input format.

- -T, --trigger=NAME

  This parameter is reserved for extension.

- -x, --no-privileges/--no-acl

  Prevents the import of access permissions (**GRANT**/**REVOKE** commands).

- -1, --single-transaction

  Executes import as a single transaction (that is, commands are wrapped in **BEGIN**/**COMMIT**).

  This parameter ensures that either all the commands are completed successfully or no application is changed. This parameter means **--exit-on-error**.

- --disable-triggers

  Specifies a reserved port for function expansion. This parameter is not recommended.

- --no-data-for-failed-tables

  By default, table data will be imported even if the statement to create a table fails (for example, the table already exists). Data in such table is skipped using this parameter. This operation is useful if the target database already contains the desired table contents.

  This parameter takes effect only when you import data directly into a database, not when you output SQL scripts.

- --no-security-labels

  Specifies a reserved port for function expansion. This parameter is not recommended.

- --no-tablespaces

  This parameter is no longer used in 8.2.0.100 and is only kept for compatibility with earlier versions.

  Does not issue commands to select tablespaces. If this parameter is used, all objects will be created during the import process no matter which tablespace is selected.

- --section=SECTION

  Imports the listed sections (such as pre-data, data, or post-data).

- --use-set-session-authorization

  Is used for plain-text backup.

  Outputs the **SET SESSION AUTHORIZATION** statement instead of the **ALTER OWNER** statement to determine object ownership. This parameter makes dump more standards-compatible. If the records of objects in exported files are referenced, import may fail. Only administrators can use the **SET SESSION AUTHORIZATION** statement to dump data, and the administrators must manually change and verify the passwords of exported files by referencing the **SET SESSION AUTHORIZATION** statement before import. The **ALTER OWNER** statement requires lower permissions.

- --with-key=KEY

  Specifies that the key length of AES128 must be 16 bytes.

  NOTE

  If the dump is encrypted, enter the **--with-key <keyname>** parameter in the **gs_restore** command. If it is not entered, you will receive an error message.

  Enter the same key while entering the dump.

NOTICE

- If any local additions need to be added to the template1 database during the installation, restore the output of **gs_restore** into an empty database with caution. Otherwise, you are likely to obtain errors due to duplicate definitions of the added objects. To create an empty database without any local additions, copy data from template0 rather than template1. Example:

```
CREATE DATABASE foo WITH TEMPLATE template0;
```

- **gs_restore** cannot import large objects selectively. For example, it can only import the objects of a specified table. If an archive contains large objects, all large objects will be imported, or none of them will be restored if they are excluded by using **-L**, **-t**, or other parameters.

NOTE

1. The **-d/--dbname** and **-f/--file** parameters do not coexist.
2. The **-s/--schema-only** and **-a/--data-only** parameters do not coexist.
3. The **-c/--clean** and **-a/--data-only** parameters do not coexist.
4. When **--single-transaction** is used, **-j/--jobs** must be a single job.
5. **--role** must be used in conjunction with **--rolepassword**.

Connection parameters:

- -h, --host=HOSTNAME

  Specifies the host name. If the value begins with a slash (/), it is used as the directory for the UNIX domain socket. The default value is taken from the *PGHOST* environment variable. If it is not set, a UNIX domain socket connection is attempted.

This parameter is used only for defining names of the hosts outside a cluster. The names of the hosts inside the cluster must be 127.0.0.1.

- -p, --port=PORT

  Specifies the TCP port listened to by the server or the local UNIX domain socket file name extension to ensure a correct connection. The default value is the *PGPORT* environment variable.

- -U, --username=NAME

  Specifies the user name to connect to.

- -w, --no-password

  Never issue a password prompt. The connection attempt fails if the host requires password verification and the password is not provided in other ways. This parameter is useful in batch jobs and scripts in which no user password is required.

- -W, --password=PASSWORD

  Specifies the user password to connect to. If the host uses the trust authentication policy, the administrator does not need to enter the **-W** parameter. If the **-W** parameter is not provided and you are not a system administrator, **gs_restore** will ask you to enter a password.

- --role=ROLENAME

  Specifies a role name for the import operation. If this parameter is selected, the **SET ROLE** statement will be issued after **gs_restore** connects to the database. It is useful when the authenticated user (specified by **-U**) lacks the permissions required by **gs_restore**. This parameter allows the user to switch to a role with the required permissions. Some installations have a policy against logging in directly as the initial user. This parameter allows data to be imported without violating the policy.

- --rolepassword=ROLEPASSWORD

  Specifies the password of the specific role.

Examples

Special case: Execute the **gsql** tool. Run the following commands to import the **MPPDB_backup.sql** file in the exported folder (in plain-text format) generated by **gs_dump/gs_dumpall** to the **gaussdb** database:

```
gsql -d gaussdb -p 8000 -W {password} -f /home/omm/test/MPPDB_backup.sql
SET
SET
SET
SET
SET
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
CREATE INDEX
CREATE INDEX
CREATE INDEX
SET
CREATE INDEX
REVOKE
REVOKE
GRANT
```



```
GRANT
total time: 30476  ms
```

gs_restore is used to import the files exported by **gs_dump**.

Example 1: Execute the **gs_restore** tool to import the exported **MPPDB_backup.dmp** file (in custom format) to the **gaussdb** database.

```
gs_restore -W {password} backup/MPPDB_backup.dmp -p 8000 -d gaussdb
gs_restore: restore operation successful
gs_restore: total time: 13053  ms
```

Example 2: Execute the **gs_restore** tool to import the exported **MPPDB_backup.tar** file (in tar format) to the **gaussdb** database.

```
gs_restore backup/MPPDB_backup.tar -p 8000 -d gaussdb
gs_restore[2017-07-21 19:16:26]: restore operation successful
gs_restore[2017-07-21 19:16:26]: total time: 21203  ms
```

Example 3: Execute the **gs_restore** tool to import the exported **MPPDB_backup** file (in directory format) to the **gaussdb** database.

```
gs_restore backup/MPPDB_backup -p 8000 -d gaussdb
gs_restore[2017-07-21 19:16:26]: restore operation successful
gs_restore[2017-07-21 19:16:26]: total time: 21003  ms
```

Example 4: Execute the **gs_restore** tool and run the following commands to import the **MPPDB_backup.dmp** file (in custom format). Specifically, import all the object definitions and data in the **PUBLIC** schema. Existing objects are deleted from the target database before the import. If an existing object references to an object in another schema, you need to manually delete the referenced object first.

```
gs_restore backup/MPPDB_backup.dmp -p 8000 -d gaussdb -e -c -n PUBLIC
gs_restore: [archiver (db)] Error while PROCESSING TOC:
gs_restore: [archiver (db)] Error from TOC entry 313; 1259 337399 TABLE table1 gaussdba
gs_restore: [archiver (db)] could not execute query: ERROR:  cannot drop table table1 because other objects depend on it
DETAIL:  view t1.v1 depends on table table1
HINT:  Use DROP ... CASCADE to drop the dependent objects too.
   Command was: DROP TABLE public.table1;
```

Manually delete the referenced object and create it again after the import is complete.

```
gs_restore backup/MPPDB_backup.dmp -p 8000 -d gaussdb -e -c -n PUBLIC
gs_restore[2017-07-21 19:16:26]: restore operation successful
gs_restore[2017-07-21 19:16:26]: total time: 2203  ms
```

Example 5: Execute the **gs_restore** tool and run the following commands to import the **MPPDB_backup.dmp** file (in custom format). Specifically, import only the definition of **table1** in the **PUBLIC** schema.

```
gs_restore backup/MPPDB_backup.dmp -p 8000 -d gaussdb -e -c -s -n PUBLIC -t table1
gs_restore[2017-07-21 19:16:26]: restore operation successful
gs_restore[2017-07-21 19:16:26]: total time: 21000  ms
```

Example 6: Execute the **gs_restore** tool and run the following commands to import the **MPPDB_backup.dmp** file (in custom format). Specifically, import only the data of **table1** in the **PUBLIC** schema.

```
gs_restore backup/MPPDB_backup.dmp -p 8000 -d gaussdb -e -a -n PUBLIC -t table1
gs_restore[2017-07-21 19:16:26]: restore operation successful
gs_restore[2017-07-21 19:16:26]: total time: 20203  ms
```


NOTE

When a cluster is created, the scheduler is started and some resources of the scheduler are created, including the schema scheduler where the tables of the scheduler are located, and the tables created when the scheduler is running, such as **bandwidth_history_table**, **cpu_template_storage**, **io_template_storage**, **mem_template_storage**, **scheduler_config**, **scheduler_storage**, **task_history_storage**, **task_storage**, **vacuum_full_rslt**, **function scheduler_workload_query_func**, and **pg_task**. When **gs_restore** is executed, the tables, schemas, and indexes of the scheduler are also restored. The scheduler is a resident process, and the new cluster automatically creates these objects. Therefore, an error message is displayed when gs_restore is executed, indicating that the objects of the scheduler exist. This error has no impact on normal cluster operations and can be ignored.

Helpful Links

gs_dump and **gs_dumpall**

8.4 gds_check

Context

gds_check is used to check the GDS deployment environment, including the OS parameters, network environment, and disk usage. It also supports the recovery of system parameters. This helps detect potential problems during GDS deployment and running, improving the execution success rate.

Precautions

- Set environment variables before executing the script. For details, see "Importing Data > Using a Foreign Table to Import Data In Parallel > Installing, Configuring, and Starting GDS" in the *Developer Guide*.
- The script must be executed in the Python 3 environment.
- This script must be run by user **root**.
- The **-t** and **--host** parameters must be specified.
- If **--host** specifies the network address 0.0.0.0 or 127.0.0.1, the MTU and NIC multi-queue are not checked.
- The NIC multi-queue check and recovery require that the NIC be at least 10 GE.
- The passwords of all nodes specified by the **--host** parameter must be the same so that the script can perform remote check successfully.
- During the recovery, set the OS configuration items according to the recommended values. For details, see the following table.

Table 8-2 OS configuration items

Parameter	Recommended Value
net.core.somaxconn	65535
net.ipv4.tcp_max_syn_backlog	65535

Parameter	Recommended Value
net.core.netdev_max_backlog	65535
net.ipv4.tcp_retries1	5
net.ipv4.tcp_retries2	12
net.ipv4.ip_local_port_range	26000 to 65535
MTU	1500
net.core.wmem_max	21299200
net.core.rmem_max	21299200
net.core.wmem_default	21299200
net.core.rmem_default	21299200
max handler	1000000
vm.swappiness	10

Table 8-3 Disk check

Check Item	Warning
Disk space usage	Greater than or equal to 70% and less than 90%
Inode usage	Greater than or equal to 70% and less than 90%

Table 8-4 Network conditions

Check Item	Error
Network connectivity	100% packet loss
NIC multi-queue	When NIC multi-queue is enabled and different CPUs are bound, **fix** can be modified.

Syntax

- **check** command
  ```
  gds_check -t check --host [/path/to/hostfile | ipaddr1,ipaddr2...] --ping-host [/path/to/pinghostfile | ipaddr1,ipaddr2...] [--detail]
  ```
- **fix** command
  ```
  gds_check -t fix --host [/path/to/hostfile | ipaddr1,ipaddr2...] [--detail]
  ```


Parameter Description

- -t

  Operation type, indicating check or recovery.

  The value can be **check** or **fix**.

- --host

  IP addresses of the nodes to be checked or recovered.

  Value: IP address list in the file or character string format

  – File format: Each IP address occupies a row, for example:

  192.168.1.200

  192.168.1.201

  – Character string format: IP addresses are separated by commas (,), for example:

  192.168.1.200,192.168.1.201

- --ping-host

  Destination IP address for the network ping check on each node to be checked.

  Value: IP address list in the file or character string format. Generally, the value is the IP address of a DN, CN, or gateway.

  – File format: Each IP address occupies a row, for example:

  192.168.2.200

  192.168.2.201

  – Character string format: IP addresses are separated by commas (,), for example:

  192.168.2.200,192.168.2.201

- --detail

  Displays detailed information about check and repair items and saves the information to logs.

- -V

  Version information.

- -h, --help

  Help information.

Examples

Perform a check. Both **--host** and --**ping-host** are in character string format.

```
gds_check -t check --host 192.168.1.100,192.168.1.101 --ping-host 192.168.2.100
```

Perform a check. **--host** is in character string format and **--ping-host** is in file format.

```
gds_check -t check --host 192.168.1.100,192.168.1.101 --ping-host /home/gds/iplist

cat /home/gds/iplist
192.168.2.100
192.168.2.101
```


Perform a check. **--host** is in file format and **--ping-host** is in character string format.

```
gds_check -t check --host  /home/gds/iplist --ping-host 192.168.1.100,192.168.1.101
```

Perform a recovery. **--host** is in character string format.

```
gds_check -t fix --host 192.168.1.100,192.168.1.101
```

Run the following command to perform the check, print the detailed information, and save the information to logs:

```
gds_check -t check --host 192.168.1.100 --detail
```

Run the following command to perform the repair, print the detailed information, and save the information to logs:

```
gds_check -t fix --host 192.168.1.100 --detail
```

8.5 gds_install

Context

gds_install is a script tool used to install GDS in batches, improving GDS deployment efficiency.

Precautions

- Configure environment variables before executing the script. For details, see "Importing Data > Using a Foreign Table to Import Data in Parallel > Installing, Configuring, and Starting GDS" in the *Developer Guide*.
- The script must be executed in the Python 3 environment.
- This script must be run by user **root**.
- Check the permission on the upper-layer directory to ensure that the GDS user has the read and write permissions on the installation operation directory, installation directory, and installation package.
- Cross-platform installation and deployment are not supported.
- The node where the command is executed must be one of the installation and deployment nodes.
- The passwords of all nodes specified by the **--host** parameter must be the same so that the script can be executed to perform remote deployment successfully.

Syntax

```
gds_install -I /path/to/install_dir -U user -G user_group --pkg /path/to/pkg.tar.gz --host [/path/to/hostfile | ipaddr1,ipaddr2...] [--ping-host [/path/to/hostfile | ipaddr1,ipaddr2...]]
```

Parameter Description

- -I

  Installation directory.

  Default value: **/opt/${*gds_user*}/packages/**, in which **${*gds_user*}** indicates the operating system user of the GDS service.

- -U

  GDS user.

- -G

  Group to which the GDS user belongs.

- --pkg

  Path of the GDS installation package, for example, **/path/to/GaussDB-8.2.1-REDHAT-x86_64bit-Gds.tar.gz**.

- --host

  IP addresses of the nodes to be installed. The value can be a file name or a string.

  – File format: Each row contains an IP address, for example:

    192.168.2.200

    192.168.2.201

  – String format: IP addresses are separated by commas (,), for example:

    192.168.2.200,192.168.2.201

  NOTE

  The node where the command is executed must be one of the nodes to be deployed. The IP address of the node must be in the list.

- --ping-host

  Destination IP address for the network ping check on each target node when **gds_check** is called.

  Value: IP address list in the file or string format. Generally, the value is the IP address of a DN, CN, or gateway.

  – File format: Each row contains an IP address, for example:

    192.168.2.200

    192.168.2.201

  – String format: IP addresses are separated by commas (,), for example:

    192.168.2.200,192.168.2.201

- -V

  Version information.

- -h, --help

  Help information.

Examples

Install GDS on nodes 192.168.1.100 and 192.168.1.101, and specify **/opt/gdspackages/install_dir** as the installation directory. The GDS user is **gds_test:wheel**.

```
gds_install -I /opt/gdspackages/install_dir --host 192.168.1.100,192.168.1.101 -U gds_test -G wheel --pkg /home/gds_test/GaussDB-8.2.1-REDHAT-x86_64bit-Gds.tar.gz
```


8.6 gds_uninstall

Background

gds_uninstall is a script tool used to uninstall GDS in batches.

Precautions

- Set environment variables before executing the script. For details, see "Importing Data > Using a Foreign Table to Import Data In Parallel > Installing, Configuring, and Starting GDS" in the *Developer Guide*.
- The script must be executed in the Python 3 environment.
- You must execute the **gds_uninstall** script as the **root** user.
- The **--host** and **-U** parameters must be contained.
- Currently, cross-platform uninstallation is not supported.
- The passwords of all nodes specified by the **--host** parameter must be the same to ensure that the script can be remotely uninstalled.

Syntax

```
gds_uninstall --host [/path/to/hostfile | ipaddr1,ipaddr2...] –U gds_user [--delete-user | --delete-user-and-group]
```

Parameter Description

- --host

  IP addresses of the nodes to be uninstalled. The value can be a file name or a string:

  – File format: Each IP address occupies a row, for example:

    192.168.2.200

    192.168.2.201

  – String format: IP addresses are separated by commas (,), for example:

    192.168.2.200,192.168.2.201

- -U

  GDS user.

- --delete-user

  The user is deleted when GDS is uninstalled. The user to be deleted cannot be the **root** user.

- --delete-user-and-group

  When GDS is uninstalled, the user and the user group to which the user belongs are deleted. You can delete a user group only when the user to be deleted is the only user of the user group. The user group cannot be the **root** user group.

- -V

  Version information.

- -h, --help

  Help information.

Example

Uninstall the GDS folders and environment variables installed and deployed by the **gds_test** user on nodes 192.168.1.100 and 192.168.1.101.

```
gds_uninstall -U gds_test --host 192.168.1.100,192.168.1.101
```

The user is deleted when GDS is uninstalled.

```
gds_uninstall -U gds_test --host 192.168.1.100,192.168.1.101 --delete-user
```

During the uninstallation, the user and user group are deleted at the same time.

```
gds_uninstall -U gds_test --host 192.168.1.100,192.168.1.101 --delete-user-and-group
```

8.7 gds_ctl

Context

gds_ctl is a script tool used for starting or stopping GDS service processes in batches. You can start or stop GDS service processes, which use the same port, on multiple nodes at a time, and set a daemon for each GDS process during the startup.

Precautions

- Before running the script, switch to a GDS user. You must run the **gds_ctl** script as a common user.
- The script must be executed in the Python 3 environment.
- **gds_ctl** inherits the main command-line parameters of GDS. Except **-p** and **-h**, the meanings of other parameters remain unchanged. In **gds_ctl**, **-p** is used only to specify a port.
- The nodes to be operated in batches using **gds_ctl** must have been installed and deployed using **gds_install**.

Syntax

- Startup command

  ```
  gds_ctl start --host [/path/to/hostfile | ipaddr1,ipaddr2...] -p PORT -d DATADIR -H ALLOW_IPs [gds other original options]
  ```

- Stop command

  ```
  gds_ctl stop --host [/path/to/hostfile | ipaddr1,ipaddr2...] -p PORT
  ```

- Restart command

  ```
  gds_ctl restart --host [/path/to/hostfile | ipaddr1,ipaddr2...] -p PORT
  ```

Parameter Description

- --host

  IP addresses of the GDS nodes to be run. The value can be a file name or a character string.

  - File format: Each IP address occupies a row, for example:

    192.168.2.200

    192.168.2.201
  - Character string format: IP addresses are separated by commas (,), for example:

    192.168.2.200,192.168.2.201

- -p

  Listening port.

  Value range: an integer ranging from 1024 to 65535

  Default value: **8098**

- --help

  Help information.

- -V

  Version information.

Compatible with Original GDS Parameters

- -d dir

  Sets the directory of the data file to be imported. If the GDS process has the permission, the directory specified by **-d** will be automatically created.

- -l log_file

  Sets the log file.

  This parameter is used together with the **-R** parameter to support automatic log splitting. After the **-R** parameter is set, GDS generates a new file based on the set value to prevent a single log file from being too large.

  Generation rule: By default, GDS identifies only files with the **.log** extension name and generates new log files.

  For example, if **-l** is set to **gds.log** and **-R** is set to 20 MB, a **gds-2020-01-17_115425.log** file will be created when the size of **gds.log** reaches 20 MB.

  If the log file name specified by **-l** does not end with **.log**, for example, **gds.log.txt**, the name of the new log file is **gds.log-2020-01-19_122739.txt**.

  When GDS is started, it checks whether the log file specified by **-l** exists. If the log file exists, a new log file is generated based on the current date and time, and the original log file is not overwritten.

- -H address_string

  Sets the hosts that can be connected to GDS. This parameter must be the CIDR format and it supports the Linux system only. If multiple network segments need to be configured, use commas (,) to separate them. For example, **-H 10.10.0.0/24, 10.10.5.0/24**.

- -e dir

  Sets the saving path of error logs generated when data is imported.

  Default value: data file directory

- -E size

  Sets the upper threshold of error logs generated when data is imported.

Value range: 0 < size < 1 TB. The value must be a positive integer plus the unit. The unit can be KB, MB, or GB.

- -S size

  Sets the upper limit of the exported file size.

  Value range: 1 MB < size < 100 TB. The value must be a positive integer plus the unit. The unit can be KB, MB, or GB. If KB is used, the value must be greater than 1024 KB.

- -R size

  Sets the maximum size of a single GDS log file specified by **-l**.

  Value range: 1 MB < size < 1 TB. The value must be a positive integer plus the unit. The unit can be KB, MB, or GB. If KB is used, the value must be greater than 1024 KB.

  Default value: **16 MB**

- -t worker_num

  Sets the number of concurrent imported and exported working threads.

  Value range: The value is a positive integer ranging between 0 and 200 (included).

  Default value: **8**

  Recommended value: 2 x CPU cores in the common file import and export scenario; in the pipe file import and export scenario, set the value to **64**.

  NOTE

  If a large number of pipe files are imported and exported concurrently, the value of this parameter must be greater than or equal to the number of concurrent services.

- -s status_file

  Sets the status file. This parameter supports the Linux system only.

- -D

  Sets the background GDS. Only the Linux OS is supported.

- -r

  Traverse files in the recursion directory and this parameter supports the Linux system only.

- --enable-ssl

  Uses the SSL authentication mode to communicate with clusters.

- --ssl-dir cert_file

  Sets the path for storing the authentication certificates when the SSL authentication mode is used.

- --debug-level

  Sets the debug log level of the GDS to control the output of GDS debug logs.

  Value range: 0, 1, and 2

  – **0**: Only the file list related to log import and export is printed. If the log volume is small, set the parameter to this value only when the system is at normal state.

  – **1**: All the log information is printed, including the connection information, session switch information, and statistics on each node. You

are advised to set the parameter to this value only during troubleshooting.

  - **2**: Detailed interaction logs and their status are printed to generate a huge number of debug logs to help identify the fault causes. You are advised to set the parameter to this value only during troubleshooting.

- --pipe-timeout

  Sets the timeout period for GDS to wait for operating a pipe.

  Value range: greater than 1s. Use a positive integer with the time unit, seconds (s), minutes (m), or hours (h). Example: **3600s**, **60m**, or **1h**, indicating one hour.

  Default value: **1h**/**60m**/**3600s**

  NOTE

  - This parameter is used to prevent the following situation: One end of the pipe file is not read or written for a long time due to human or program problems. As a result, the read or write operation on the other end of the pipe is hung.
  - This parameter does not indicate the maximum duration of a data import or export task. It indicates the maximum timeout duration of each read, open, or write operation on the pipe. If the timeout duration exceeds the value of **--pipe-timeout**, an error is reported to the frontend.

Examples

Start a GDS process. Its data files are stored in the **/data** directory, the IP address is 192.168.0.90, and the listening port number is 5000.

```
gds_ctl start --host 192.168.0.90 -d /data/ -p 5000 -H 10.10.0.1/24 -D
```

Start GDS processes in batches. The data files are stored in the **/data** directory, the IP addresses are 192.168.0.90, 192.168.0.91, and 192.168.0.92, and the listening port number is 5000.

```
gds_ctl start --host 192.168.0.90,192.168.0.91,192.168.0.92 -d /data/ -p 5000 -H 0/0 -D
```

Stop GDS processes on nodes 192.168.0.90, 192.168.0.91, and 192.168.0.92 whose port number is 5000 in batches.

```
gds_ctl stop --host 192.168.0.90,192.168.0.91,192.168.0.92 -p 5000
```

Restart GDS processes on nodes 192.168.0.90, 192.168.0.91, and 192.168.0.92 whose port number is 5000 in batches.

```
gds_ctl restart  --host 192.168.0.90,192.168.0.91,192.168.0.92 -p 5000
```

