Data Warehouse Service

Tool Guide

Issue	01
Date	2023-03-28

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.

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 Client 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	A CLI tool running on the Linux OS. It works with foreign tables to quickly import and export data. 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	A CLI tool used for migrating SQL scripts from Teradata or Oracle to GaussDB(DWS) to rebuild a database on GaussDB(DWS). 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.
gs_sshexkey	During cluster installation, you need to execute commands and transfer files among hosts in the cluster. gs_sshexkey is used to help users establish mutual trust.

2 Downloading Client Tools

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

Step 2 In the navigation tree on the left, click **Connections**.

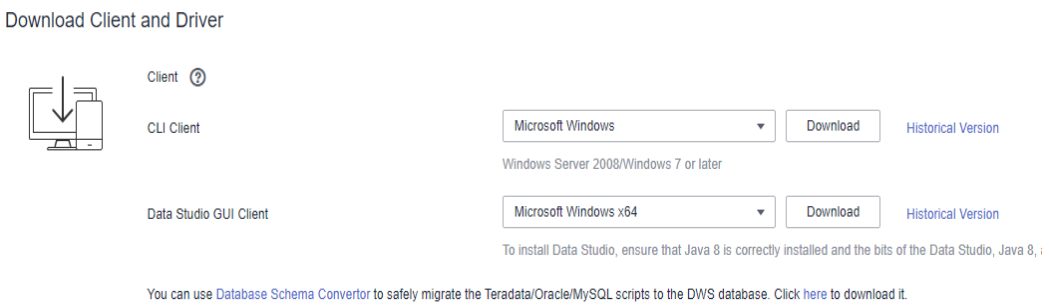
Step 3 In the **Download Client and Driver** area, select the tools corresponding to the cluster version based on the computer OS and download them.

You can download the following tools:

- gsql CLI client: The gsql tool package contains the gsql client tool, GDS (parallel data loading tool), gs_dump, gs_dumpall, and gs_restore tools.
- Data Studio GUI client
- DSC migration tool

The gsql and Data Studio client tools have multiple historical versions. You can click **Historical Version** to download the tools based on the cluster version. GaussDB(DWS) clusters are compatible with earlier versions of gsql and Data Studio tools. You are advised to download the matching tool version based on the cluster version.

Figure 2-1 Downloading the DSC client



----End

3 gsql - CLI Client

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 3-7**) 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**.

– 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)
```

- 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))`

Variable	Description	Example
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.
%>	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.

Symbol	Description
%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	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.
EDITOR	

Name	Description
VISUAL	

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 Instruction

Downloading and Installing gsql and Using It to Connect to the Cluster Database

For details about how to download and install gsql and connect it to the cluster database, see **Using the gsql CLI Client to Connect to a Cluster** in the *Data Warehouse Service (DWS) Management Guide*.

Example

The example shows how to spread a command over several lines of input. Pay attention to prompt changes:

```
postgres=# CREATE TABLE HR.areaS(
postgres(# area_ID    NUMBER,
postgres(# area_NAME VARCHAR2(25)
postgres-# )tablespace EXAMPLE;
CREATE TABLE
```

View the table definition.

```
\d HR.areaS
              Table "hr.areas"
  Column   |         Type          | Modifiers
-----------+-----------------------+-----------
 area_id   | numeric               | not null
 area_name | character varying(25) |
```

Insert four lines of data into **HR.areaS**.

```
INSERT INTO HR.areaS (area_ID, area_NAME) VALUES (1, 'Wood');
INSERT 0 1
INSERT INTO HR.areaS (area_ID, area_NAME) VALUES (2, 'Lake');
INSERT 0 1
```



```
INSERT INTO HR.areaS (area_ID, area_NAME) VALUES (3, 'Desert');
INSERT 0 1
INSERT INTO HR.areaS (area_ID, area_NAME) VALUES (4, 'Iron');
INSERT 0 1
```

Change the prompt.

```
\set PROMPT1 '%n@%m %~%R%#'
dbadmin@[local] postgres=#
```

View the table.

```
dbadmin@[local] postgres=#SELECT * FROM HR.areaS;
 area_id |       area_name
---------+------------------------
       1 | Wood
       4 | Iron
       2 | Lake
       3 | Desert
(4 rows)
```

Run the **\pset** command to display the table in different ways.

```
dbadmin@[local] postgres=#\pset border 2
Border style is 2.
dbadmin@[local] postgres=#SELECT * FROM HR.areaS;
+---------+------------------------+
| area_id |       area_name        |
+---------+------------------------+
|       1 | Wood                   |
|       2 | Lake                   |
|       3 | Desert                 |
|       4 | Iron                   |
+---------+------------------------+
(4 rows)
dbadmin@[local] postgres=#\pset border 0
Border style is 0.
dbadmin@[local] postgres=#SELECT * FROM HR.areaS;
area_id       area_name
------- ----------------------
      1 Wood
      2 Lake
      3 Desert
      4 Iron
(4 rows)
```

Use the meta-command.

```
dbadmin@[local] postgres=#\a \t \x
Output format is unaligned.
Showing only tuples.
Expanded display is on.
dbadmin@[local] postgres=#SELECT * FROM HR.areaS;
area_id|2
area_name|Lake

area_id|1
area_name|Wood

area_id|4
area_name|Iron

area_id|3
area_name|Desert
dbadmin@[local] postgres=#
```


3.3 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 3-6**.

Table 3-6 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

3.4 Command Reference

For details about gsql parameters, see **Table 3-7**, **Table 3-8**, **Table 3-9**, and **Table 3-10**.

Table 3-7 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 3-8 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 3-9 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 3-10 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.

3.5 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 3-11**, **Table 3-12**, **Table 3-13**, **Table 3-14**, **Table 3-16**, **Table 3-18**, **Table 3-19**, **Table 3-20**, and **Table 3-22**.

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 3-11 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. • 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 3-12 Buffer query meta-commands

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

Parameter	Description
\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 3-13 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 psql 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 3-14, **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 3-14 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 3-15**.	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
\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`
\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`

Parameter	Description	Value Range	Example

\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 3-15 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.
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.

Parameter	Description
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.
arwdDxtA	ALL PRIVILEGES: grants all available permissions to specified users or roles at a time.
*	Authorization options for preceding permissions

Table 3-16 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 3-17**.

Parameter	Description
\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 3-17 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
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.

Option	Description	Value Range
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.	-
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.

Option	Description	Value Range
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.
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.	-

Option	Description	Value Range
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 3-18 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 3-19 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.	-

Parameter	Description	Value Range
\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 3-20 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 underscores. **Table 3-21** 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 3-21 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**

Command	Description	Value Range
\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 3-22 Large object meta-commands

Parameter	Description
\lo_list	Shows a list of all GaussDB(DWS) large objects stored in the database, as well as the comments provided for them.

Table 3-23 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.

3.6 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 3-24**.

  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 3-24 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 PG_STAT_ACTIVITY; 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. For details, see "Configuration File Reference" in the *Developer Guide*.

  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 - Integrated Database Development Tool

4.1 About Data Studio

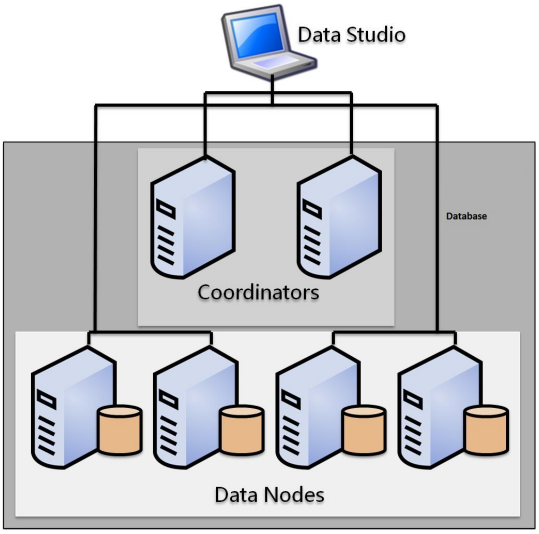
4.1.1 Overview

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.



4.1.2 Constraints and Limitations

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

Object Browser Filter Tree

The filter count and filtering status of the tree are not supported.

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. For details, see **Session Setting**.

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 **Edit Table Data** operations:
  - 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

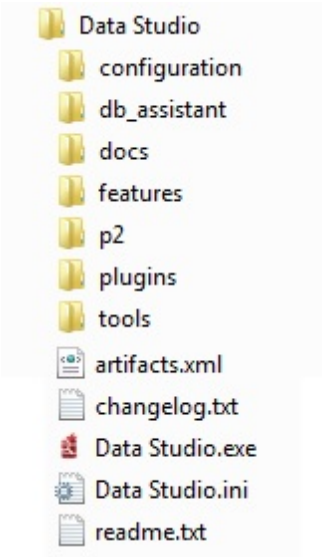
- If **Enable SSL** is not selected during connecting to Data Studio, data is not encrypted by default.
- If the security file is damaged during the SSL connection, Data Studio cannot perform any database operations. To resolve this problem, delete the security folder in the corresponding configuration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

The release package structure of Data Studio is as follows:



Folders/Files	Description
configuration	Contains information about the application launcher and the required **Eclipse** plug-in path.
db_assistant	Contains SQL assistant related files.
docs	• Contains **Data Studio User Manual.pdf** which provides you with details on using Data Studio. • Contains copyright notices, licenses, and the written offer for the open source libraries used in Data Studio.
features	Contains Eclipse (rich client protocol-GUI) and Data Studio features.

Folders/Files	Description
p2	Contains files required for provisioning and managing **Eclipse** and **Equinox**-based applications.
plugins	Contains the required **Eclipse** and Data Studio plugins.
tools	Contains Data Studio dependent tools.
UserData/ • Autosave • *Logs/* • *Preferences/* • *Profile/* – *History/* • *Security/*	Contains separate folders for each OS user of Data Studio. **Autosave** - Contains the auto saved information of unsaved queries and functions/procedures. **Logs** - Contains Data Studio.log which stores log information of all the operations performed in Data Studio. **Preferences** - Contains the **Preferences.prefs** file which stores the custom preferences. **Profile** - Contains the **connection.properties**, SQL History and **Profiles.txt** files required to manage connection profiles in Data Studio. **Security** - Contains files required to manage security in Data Studio. **NOTE** • The UserData folder is created only after the first user opens an instance of Data Studio. • The **Logs** folder, language, memory settings and log level are common for all users. • The **Logs** folder, **Data Studio.log** file, Preferences folder, **Preferences.prefs** file, Profile folder, **connection.properties** file, **Profiles.txt** file, and security folder are created after launching Data Studio. • If the **Logs** folder path is provided in the **Data Studio.ini** file, then logs are created in the specified path. • When user is not able to log in to the Data Studio because the security keys are corrupted, follow the steps to generate new security keys: 1. Delete the **Security** folder from the **Data Studio folder > UserData > Security** folder 2. Restart Data Studio
artifacts.xml	Contains the product build information.

Folders/Files	Description
changelog.txt	Contains the detailed change log information of release version.
Data Studio.exe/DataStudio.sh	Allows you to connect to the database and perform various operations (like managing database objects, editing or executing PL/SQL programs and so on).
Data Studio.ini	Contains run-time configuration information of Data Studio.
readme.txt	Contains the features or fixed issues of current release version.

4.1.4 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 4-1 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)
	SUSE Linux Enterprise Server 12	SP0 (SUSE 12.0)
		SP1 (SUSE 12.1)
		SP2 (SUSE 12.2)
		SP3 (SUSE 12.3)
		SP4 (SUSE 12.4)
	CentOS	7.4 (CentOS7.4)
		7.5 (CentOS7.5)
		7.6 (CentOS7.6)

Server	OS	Supported Version
TaiShan ARM server	NeoKylin	7.0

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 4-2 Data Studio software requirement

Software	Specifications
Java	Open JDK 1.8 or later corresponding to the OS bit is recommended.
GTK	For Linux OSs, GTK 2.24 or later is required.
GNU libc	DDL can be displayed, imported, exported; and data operations can be performed only in libc 2.17 and later in GN.

Table 4-3 Supported database versions

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

NOTE

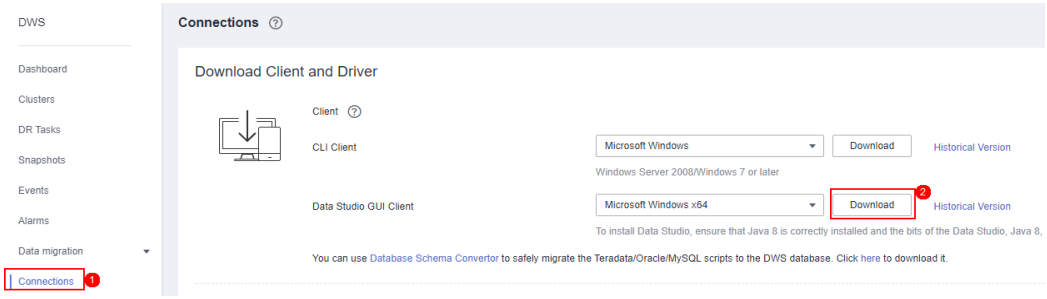
The recommended minimum screen resolution is 1080 x 768. If the resolution is lower than this value, the page display will be abnormal.

4.2 Installing and Configuring Data Studio

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

Installation Preparations

On the **Connections** page of the GaussDB(DWS) console, download the Data Studio GUI client.



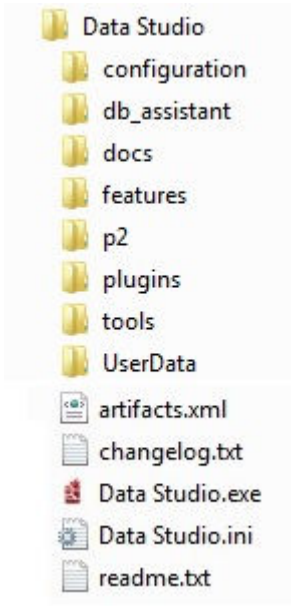
Installing Data Studio

Data Studio can be run after decompression of the package.

Follow the steps below to install Data Studio:

Step 1 Unzip the required package (32-bit or 64-bit) to the **program files (x86)** or **program files** folder respectively. If the user prefers to install in other folder, then the admin should control the folder access permissions to users.

You will see the following files and folders:



Step 2 Locate and double-click the **Data Studio.exe** to launch Data Studio.

NOTE

The **UserData** folder is created after the first user launches Data Studio. Refer to **Quick Start** in case of any error while launching Data Studio.

----End

See **Adding a Connection** to create a database connection.

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. All the following mentioned parameters are not mandatory.

List of configuration parameters used in Data Studio:

Table 4-4 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.v 20150511-1540.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.equino x.launcher.win32. win32.x86_1.1.300 .v20150602-1417** or **plugins/ org.eclipse.equino x.launcher.win32. win32.x86_64_1.1. 300.v20150602-14 17** depending on the installation package used
-clearPersistedStat e	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>	Defines the relative path to Java executable	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 4-5 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 4-5 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.

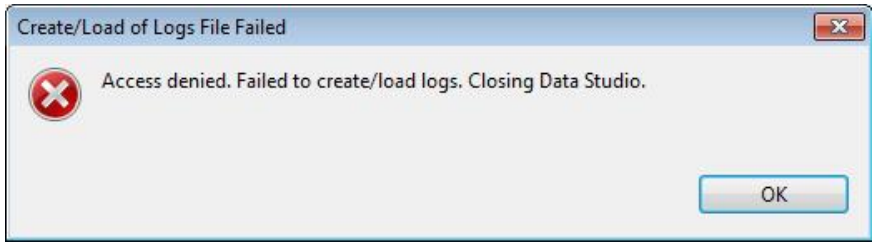
For example:

-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

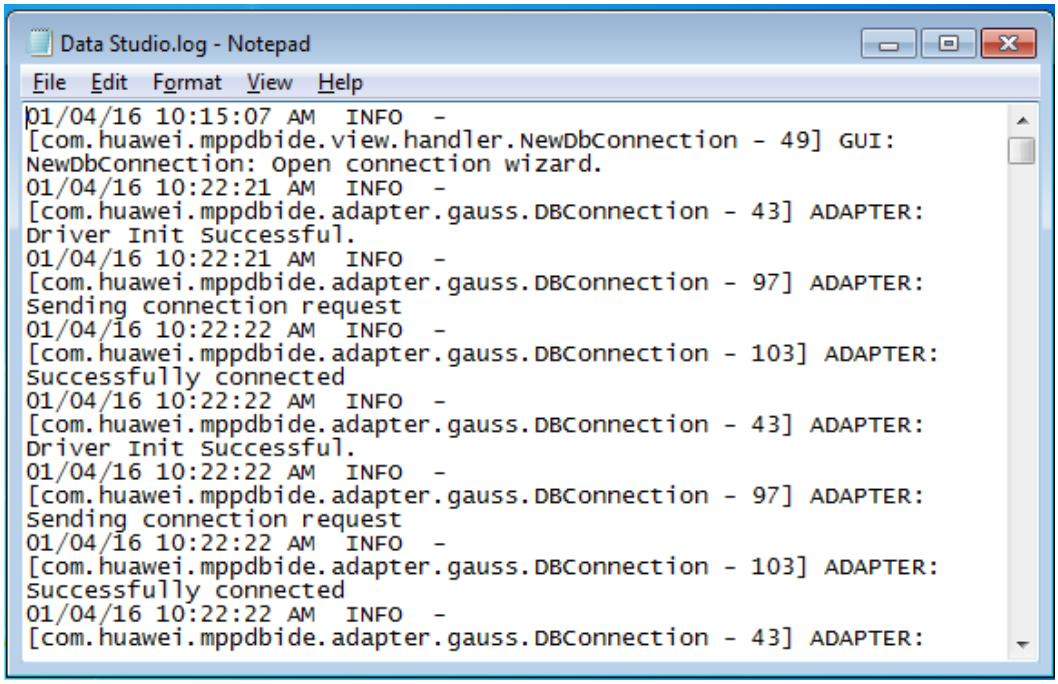
The **Data Studio.log** file will be created in the **Data Studio\UserData\<user name>\logs** path if:

- 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.

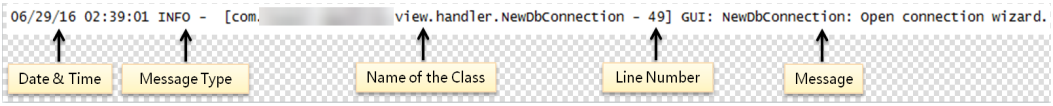
-detailLogging=false

If the flag value is **true**, then the stack trace details of exception, error or throwable will be saved in the log file.

If the flag value is **false**, then no stack trace details will be saved in the log file.

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. The **Data Studio** deletes the old log file that is **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 4-6 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						

4.3 Quick Start

This section describes how to start Data Studio.

Prerequisites

The **StartDataStudio.bat** batch file checks the version of Operating System (OS), Java and Data Studio as a prerequisite to run Data Studio.

Step 1 In the **Release package** navigate to **Tools** folder, locate and double-click **StartDataStudio.bat** to execute and check Java version compatibility.

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 **error message** is 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)	Outcome
32	32	32	Data Studio is launched.
32	64	32	Data Studio is launched.
32	64	64	**Error Message** is displayed.
64	32	32	**Error Message** is displayed.

DS Installation (32-bit/64-bit)	OS (Bit)	Java (bit)	Outcome
64	64	32	**Error Message** is displayed.
64	64	64	Data Studio is launched.

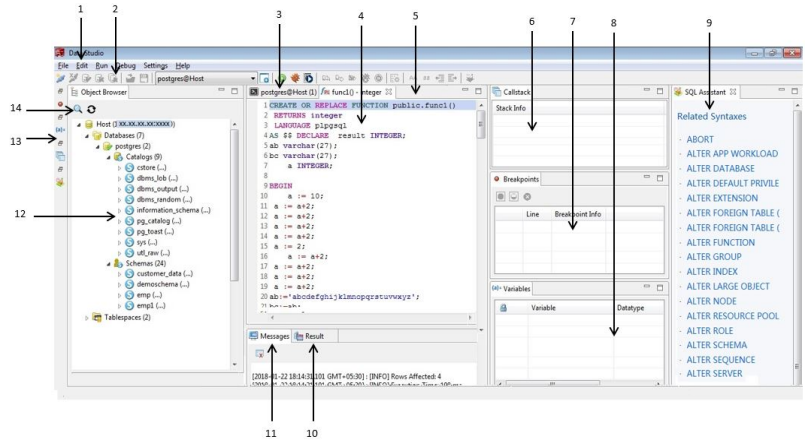
----End

4.4 Data Studio GUI

This section describes the Data Studio GUI.

The Data Studio GUI contains the following:

1. **Main Menu** provides basic operations.
2. **Toolbar** contains buttons for easy access to frequently used operations.
3. **SQL Terminal** tab is used to execute SQL statements and functions/procedures.
4. **PL/SQL Viewer** tab displays the content of functions/procedures.
5. **Editor Area** is used to perform edit operations.
6. **Callstack** pane shows the execution stack.
7. **Breakpoints** pane shows any breakpoints that have been set.
8. **Variables** pane shows variables and their values.
9. **SQL Assistant** tab displays suggestion or reference for the information entered in the SQL Terminal and PL/SQL Viewer.
10. **Result** tab displays the result(s) of an executed function/procedure, or an SQL statement.
11. **Message** tab displays the process output, standard input, standard output, and standard errors.
12. **Object Browser** contains a hierarchical tree display of database connection(s) and related database objects to which the user has access. All default created schemas except for public are grouped under **Catalogs** and user schemas are grouped under **Schemas** below the respective database.
13. **Minimized Window Panel** is used to open Callstack, Breakpoints and Variables pane. This panel is displayed only when Callstack, Breakpoints or Variables pane or all three are minimized.
14. **Search Toolbar** is used to search objects from the Object browser.



4.5 Data Studio Menus

4.5.1 File

The **File** menu contains database connection options. Click **File** from main menu or press **Alt**+**F** to open the **File** menu.

Function	Button	Shortcut Key	Description
New Connection		Ctrl+N	Creates and adds a new database connection to the **Object Browser** and **SQL Terminal**.
Remove Connection		-	Deletes the selected database connection from **Object Browser**.
Connect To DB		-	Connects to the database.
Disconnect From DB		Ctrl+Shift+D	Disconnects from the selected database.
Disconnect All		-	Disconnects all the databases of a specified connection.
Open		Ctrl+O	Loads SQL queries into the **SQL Terminal**.

Function	Button	Shortcut Key	Description
Save		Ctrl+S	Saves the SQL scripts of the **SQL Terminal** to an SQL file.
Save As		CTRL+ALT+S	Saves the SQL scripts of the **SQL Terminal** to a new SQL file.
Exit	-	Alt+F4	Exits from Data Studio and ends the connection. **NOTE** Any unsaved changes will be lost.

Stopping Data Studio

Follow the steps below to stop Data Studio:

Step 1 Click the  button.

Alternatively choose **File > Exit**.

The **Exit Application** dialog box is displayed prompting you to take the required actions.

Step 2 Click the corresponding buttons as required.

- **Force Exit**: Exits the application without saving the SQL execution history.

  NOTE

  If you click **Force Exit**, the SQL execution history that is not saved may be lost.

- **Graceful Exit**: Saves the SQL execution history that has not been saved to the disk before exiting.
- **Cancel**: Cancels the exit from the application.

----End

4.5.2 Edit

The **Edit** menu contains clipboard, **Format**, **Find and Replace**, and **Search Objects** operations to use in the **PL/SQL Viewer** and **SQL Terminal** tab. Press **Alt** +**E** to open the **Edit** menu.

Function	Shortcut Key	Description
Cut	Ctrl+X	Cuts the selected text

Function	Shortcut Key	Description
Copy	Ctrl+C	Copies the selected text or qualified object name
Paste	Ctrl+V	Pastes the selected text or qualified object name
Format	Ctrl+Shift+F	Formats all SQL statements and functions/procedures.
Select All	Ctrl+A	Selects all the text in SQL Terminal
Find and Replace	Ctrl+F	Finds and replaces text in SQL Terminal
Search Objects	Ctrl+Shift+S	Searches for objects within a connected database.
Undo	Ctrl+Z	Undoes the previous operation
Redo	Ctrl+Y	Redoes the previous operation
UPPERCASE	Ctrl+Shift+U	Changes the case of the selected text to uppercase
lowercase	Ctrl+Shift+L	Changes the case of the selected text to lowercase
Go To Line	Ctrl+G	Skips to a specific line in the Terminal or PL/SQL Viewer
Comment/Uncomment Lines	Ctrl+/	Comments/Uncomments each selected line
Comment/Uncomment Block	Ctrl+Shift+/	Comments/Uncomments all selected lines or a selected block

Copy

Copy can also be used to copy objects from Object Browser.

The format of copied object name is:

Table 4-7

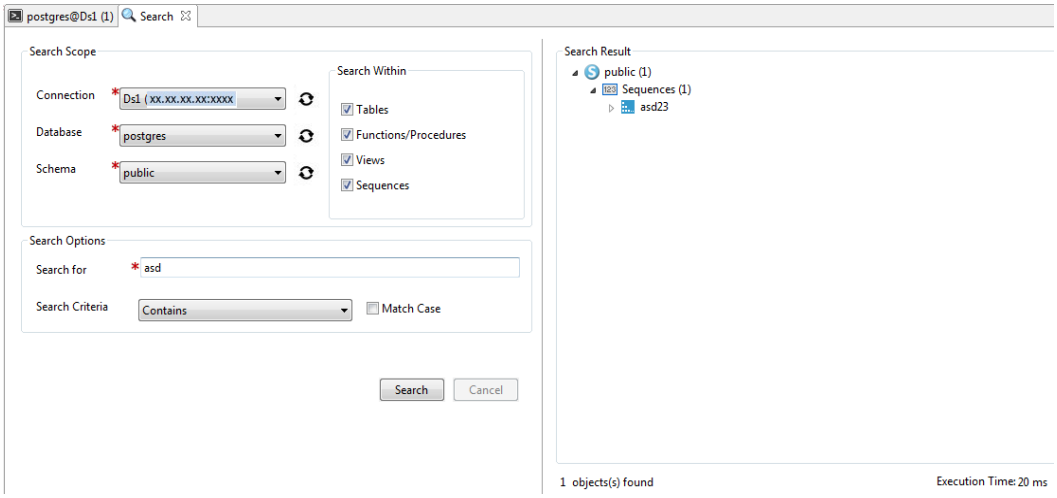
Object Type	Copied Format
Functions/Procedures	schema.object name(parameter name parameter type,...)
Databases	object name
Schemas	object name
Columns	object name
Constraints	object name
Partition names	object name
All other object types	schema.object name

Search Objects

The Searching Objects option allows you to search for object(s) from the Object Browser based on the search criteria entered. The search operation can be executed either from **Edit > Search Objects** menu or by clicking  from the Object Browser toolbar. The result of search displays tree structure similar to Object Browser. Right-click operations except for **Refresh** can be performed on the search result objects. Modified objects as a result of drop, set schema, rename, and so on can be viewed only from the main Object Browser after refresh. Right-click options on group names (tables, schema, views and so on) are not allowed on search result objects. Only objects to which you have access can be searched. Objects that you do not have access do not appear in the **Search Scope**.

NOTE

Newly added objects can be viewed in the **Search** window by clicking the refresh option at the end of the object type.



Supported Search Options:

Search Options	Search Behavior
Contains	A search text which contains the searched characters will be displayed.
Starts With	A search text which starts with the searched character will be displayed.
Exact Word	A search text which matches exactly with searched characters will be displayed.
Regular Expression	A search text with regular expression searches for similar pattern in Object Browser that fulfills the search condition. Select Regular Expression from **Search Criteria** drop-down to perform this search. For more information refer to POSIX Regular Expressions rules. **Example:** ● ^a, this displays all objects that start with the letter a. ● ^[^A-Za-z]+$, this displays all objects that do not have alphabets in them. ● ^[^0-9]+$, this displays all objects that do not have numbers in them. ● ^[a-t][^r-z]+$, this displays all objects whose name starts between a and t and excludes those that have characters between r and z in them. ● ^e.*a$, this displays all objects that start with the letter e and end with letter a. ● ^[a-z]+$ and select **Match Case**, this displays all objects that contain only alphabets in lowercase. ● ^[A-Z]+$ and select **Match Case**, this displays all objects that contain only alphabets in uppercase. ● ^[A-Za-z]+$ and select **Match Case**, this displays all objects that contain only alphabets in lowercase and uppercase. ● ^[A-Za-z0-9]+$ and select **Match Case**, this displays all objects that contain only alphabets in lowercase, uppercase and numbers. ● ^".*"$, this displays all objects that are within in quotes.

Underscore and % search:

Search Value	Search Behavior
_	A search text with _ (underscore) in it considers the underscore as a wildcard of single character. This does not apply to regular expression, starts with and exact word search. **Example:** • _ed, this displays all objects that starts with any single character followed by "ed". • D_t_e, this displays all objects that have character "d", followed by any single character, followed by character "t", followed by any single character, and followed by character "e".
%	A search text with % (percentage) in it considers the percentage as a wildcard of multiple characters. This does not apply to regular expression, starts with and exact word search. **Example:** • %ed, this displays all objects that have "ed" pattern in it. • D%t%e, this displays all objects that have character "d", followed by any number of characters, followed by character "t", followed by any number of characters, and followed by character "e".

Match case runs the search to match with the search text case.

4.5.3 Run

The **Run** menu contains options to execute a database object in the **PL/SQL Viewer** tab and to execute SQL statements in the **SQL Terminal** tab. Press **Alt**+**R** to open the **Run** menu.

Function	Button	Shortcut Key	Description
Execute DB Object		Ctrl+E	Starts execution (in normal mode) of the specified function/ procedure. Displays the result in **Result** tab. Displays the information on actions performed in **Messages** tab.

Function	Button	Shortcut Key	Description
Compile/Execute Statement		Ctrl+Enter	Compiles the function/ procedure. Starts execution of SQL statements in the **SQL Terminal** tab.
Compile/Execute in New Tab		Ctrl+Alt+Enter	Executes statement in new tab retaining old. Disabled, when **Retain Current** is selected.
Cancel		Shift+Esc	Cancels the executing query. Displays the result in **Result** tab. Displays the information on actions performed in **Messages** tab.

4.5.4 Debug

The **Debug** menu contains debug operations in the **PL/SQL Viewer** and **SQL Terminal** tab. Press **Alt**+**D** to open the **Debug** menu.

Function	Button	Shortcut Key	Description
Debug DB Object		Ctrl+D	Starts the debugging process.
Continue		F9	Continues with debugging.
Terminate Debugging		F10	Terminates debugging.
Step Into		F7	Steps through the code statement.
Step Over		F8	Steps over the function.
Step Out		Shift+F7	Steps out of the function.

4.5.5 Settings

The **Settings** menu contains the option to change the language. Press **Alt**+**G** to open the **Settings** menu.

Function	Shortcut Key	Description
Language	-	Sets the language for the Data Studio GUI.
Preferences	-	Sets the user preferences in Data Studio.

4.5.6 Help

The **Help** menu contains the user manual and version information of Data Studio. Press **Alt**+**H** to open the **Help** menu.

Function	Shortcut Key	Description
User Manual	F1	Opens the Data Studio User Manual.
About Data Studio	-	Displays the current version and copyright information of Data Studio.

NOTE

- Visit https://java.com/en/download/help/path.xml to set the Java Home path.

4.6 Data Studio Toolbar

The following image shows the toolbar:

The toolbar contains the following operations:

- **Adding a Connection**
- **Removing a Connection**
- **Connecting to a Database**
- **Disconnecting a Database**
- **Disconnecting All Databases**
- **Opening SQL Scripts**
- **Saving SQL Scripts**
- **Connection Profile Drop-down List**

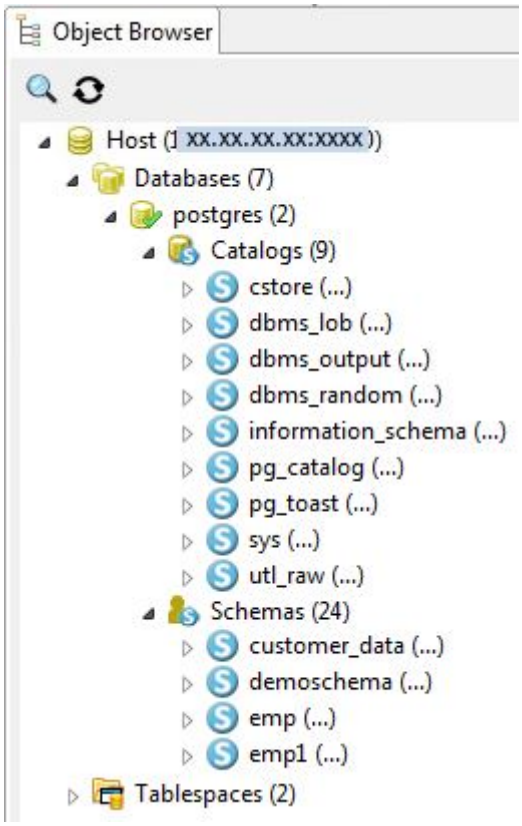
- Opening SQL Terminals
- Executing a DB Object
- Debugging a DB Object
- Editing a Function/Procedure
- Step Into
- Step Out
- Step Over
- Terminating Debugging
- Continuing Debugging
- Execution Plan and Cost
- Viewing the Query Execution Plan and Cost Graphically
- Formatting
- Upper Case
- Lower Case
- SQL Assistant

4.7 Data Studio Right-Click Menus

This section describes the right-click menus of Data Studio.

Object Browser Pane

The following figure shows the **Object Browser** pane.



Right-clicking the connection name allows you to select **Rename Connection**, **Edit Connection**, **Remove Connection**, and **Properties** along with **Refresh** options.

Menu Item	Shortcut Key	Description
Rename Connection	-	Renames a connection.
Edit Connection	-	Modifies connection details.
Remove Connection	-	Removes the existing database connection.
Properties	-	Shows the details of a connection.
Refresh	F5	Refreshes a connection.

Right-clicking the **Databases** tab allows you to select **Create Database**, **Disconnect All**, and **Refresh** options.

Menu Item	Shortcut Key	Description
Create Database	-	Creates a database of this connection.
Disconnect All	-	Disconnects all the databases of this connection.
Refresh	F5	Refreshes a database group.

Right-clicking the active database allows you to select **Disconnect from DB**, **Open Terminal**, **Properties**, and **Refresh** options.

Menu Item	Shortcut Key	Description
Disconnect from DB	Ctrl+Shift+D	Disconnects from a database.
Open Terminal	Ctrl+T	Opens a terminal of this connection.
Properties	-	Displays the properties of a database.
Refresh	F5	Refreshes a database.

Right-clicking the inactive database allows you to select **Connect to DB**, **Rename Database**, and **Drop Database** options.

Menu Item	Shortcut Key	Description
Connect to DB	-	Connects to a database.
Rename Database	-	Renames a database.
Drop Database	-	Drops a database.

Right-clicking the **Catalogs** tab allows you to select the **Refresh** option.

Menu Item	Shortcut Key	Description
Refresh	F5	Refreshes a function/procedure.

Right-clicking the **Schemas** tab allows you to select **Create Schema**, **Grant/Revoke**, and **Refresh** options.

Menu Item	Shortcut Key	Description
Create Schema	-	Creates a schema.
Grant/Revoke	-	Grants or revokes permissions for a schema group.
Refresh	F5	Refreshes a schema.

Right-clicking the schema allows you to select **Export DDL**, **Export DDL and Data**, **Rename Schema**, **Drop Schema**, **Grant/Revoke**, and **Refresh** options.

Menu Item	Shortcut Key	Description
Export DDL	-	Exports DDL of a schema.
Export DDL and Data	-	Exports DDL and data of a schema.
Rename Schema	-	Renames a schema.
Drop Schema	-	Drops a schema.
Grant/Revoke	-	Grants or revokes permissions for a schema.
Refresh	F5	Refreshes a schema.

Right-clicking **Functions/Procedures** allows you to select **Refresh** and **Create Function/Procedure** and **Grant/Revoke** options.

Menu Item	Shortcut Key	Description
Create PL/SQL Function	-	Creates a PL/SQL function.
Create PL/SQL Procedure	-	Creates a PL/SQL procedure.
Create SQL Function	-	Creates an SQL function.
Create C Function	-	Creates a C function.
Grant/Revoke	-	Grants or revokes permissions for a function/procedure.
Refresh	F5	Refreshes a function/procedure.

Right-clicking **Tables** allows you to select **Create table**, **Create partitioned table**, **Grant/Revoke**, and **Refresh** options.

Menu Item	Shortcut Key	Description
Create table	-	Creates a common table.
Create partitioned table	-	Creates a partitioned table.
Grant/Revoke	-	Grants or revokes permissions for a table.
Refresh	F5	Refreshes a table.

Right-clicking **Views** allows you to select **Create View**, **Grant/Revoke**, and **Refresh** options.

Menu Item	Shortcut Key	Description
Create View	-	Creates a view.
Grant/Revoke	-	Grant/revokes permissions for a view.
Refresh	F5	Refreshes a view.

Right-clicking the **PL/SQL Viewer** tab allows you to select **Cut**, **Copy**, **Paste**, **Select All**, **Comment/Uncomment Lines**, **Comment/Uncomment Block**, **Compile**, **Execute**, **Add Variable To Monitor**, **Debug with Rollback**, and **Debug** options.

Right-Click Option	Shortcut Key	Description
Cut, Copy, Paste	Ctrl+X, Ctrl +C, Ctrl+V	Specifies clipboard operations.
Select All	Ctrl+A	Selects options in the **PL/SQL Viewer** tab.
Comment/Uncomment Lines	-	Comments or uncomments all selected rows.
Comment/Uncomment Block	-	Comments or uncomments all selected rows or blocks.
Compile	-	Compiles a function/procedure.
Execute	-	Executes a function/procedure.
Add Variable To Monitor	-	Adds variables to the monitor window.
Debug with Rollback	-	Debugs a function/procedure and rolls back changes after the debugging is complete.
Debug	-	Debugs a function/procedure.

Right-clicking the **SQL Terminal** tab allows you to select **Cut, Copy, Paste**, **Select All**, **Execute Statement**, **Open**, **Save**, **Find and Replace**, **Execution Plan**, **Comment/Uncomment**, **Save As**, **Format** , and **Cancel** options.

Right-Click Option	Shortcut Key	Description
Cut, Copy, Paste	Ctrl+X, Ctrl +C, Ctrl+V	Specifies clipboard operations.
Select All	-	Selects all text.
Execute Statement	-	Executes a query.
Open	-	Opens a file.
Save	-	Saves a query.
Find and Replace	-	Finds and replaces text in the **SQL Terminal**tab.
Execution Plan	-	Executes a query.
Comment/Uncomment Lines	Ctrl+/	Comments or uncomments all selected rows.
Comment/Uncomment Block	Ctrl+Shift+/	Comments or uncomments all selected rows or blocks.
Cancel	-	Cancels the execution.

Right-Click Option	Shortcut Key	Description
Save As	CTRL+ALT+S	Saves the query to a new file.
Format	CTRL+SHIFT +F	Formats the selected SQL statements using the rules configured in the query.

Right-clicking the **Messages** tab allows you to select **Copy**, **Select All**, and **Clear** options.

Right-Click Option	Shortcut Key	Description
Copy	Ctrl+C	Copies the text.
Select All	Ctrl+A	Selects all text.
Clear	-	Clears the text.

4.8 Connection Profiles

4.8.1 Overview

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

Enter the connection parameters to create a new database connection between Data Studio and the database in the server. Hovering over the connection name will display the server information.

NOTE

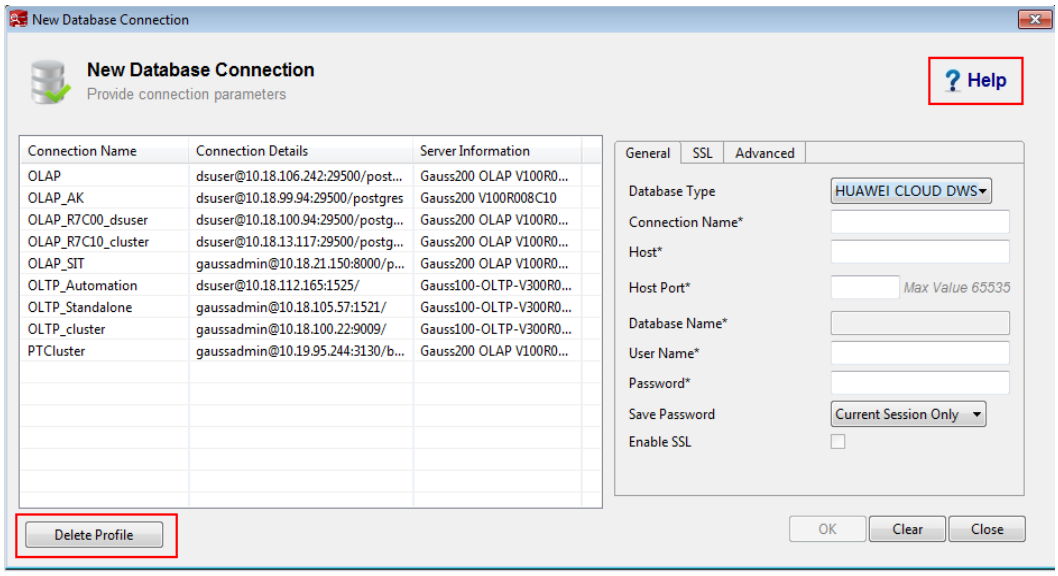
You need to fill all the mandatory parameters, that are marked with asterisk (*) to complete the operation successfully.

4.8.2 Adding a Connection

Performing the following steps to establish a new database connection.

Step 1 Choose **File > New Connection** from the main menu, or

click  on the toolbar, or press **Ctrl+N** to connect to the database. The **New Database Connection** dialog box is displayed.



NOTE

While establishing a connection, if the preference file is corrupted or the preferences values are invalid, then an error message is displayed informing you that preference values are invalid and default values are set for preferences. To complete establishing a new database connection operation, click **OK**.

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.

NOTE

The server information will be displayed only after one successful connection.

- Double clicking a connection name populates the connection parameters such as **Connection Name**, **Host**, and **Host Port**.

NOTE

If password is corrupted for any of the existing connection profile or the key is corrupted, then the password field needs to be filled in for all created connections.

- Clicking displays different pop-up messages based on the connection status of database.
  - If the database connection is active, then **Remove Connection Confirmation** pop-up is displayed. Click **Yes** to disconnect all databases.
  - If the database connection is not active, then **Remove Connection Confirmation** pop-up is displayed.
- Clicking without a connection name displays a pop-up stating to select at least one connection profile.

Step 3 Provide the following credentials to enter a new set of parameters to connect to the database:

NOTE

- You can click **Clear** to clear all fields in the **New Database Connection** dialog box.
- Use shortcut key (**Ctrl+V**) to paste data in Connection window. Data Studio does not support right-click options for all dialog boxes.

Field Name	Description	Example
Database Type	Select the database type.	-
Connection Name	Provide a connection name.	My_Connection_DB
Host	Provide the IP address (IPv4) or server domain name. **NOTE** • If domain name length is greater than 25 characters, then the complete domain name will not be displayed. **Example:** *test1(db.dws...com:25xxx)* • Hovering over the connection name once the connection is established will show the server IP and version. • Entry made in this field will be validated for IP address if it has format of digits with three separators (.). Any entry not meeting this validation will be considered as domain name. • A typical domain name: – Starts with a letter. – Allows letters, digits, hyphens (-), and period (.). All other special characters are not allowed. – Does not allow space/tabs. – Length cannot exceed 253 characters and a maximum of 63 characters is allowed in between periods.	db.dws.mycloud.com 10.xx.xx.xx
Host Port	Provide the port address.	8000
Database Name	Provide the database name.	gaussdb
User Name	Provide the user name to connect to the selected database.	-
Password	Provide the password to connect to the database. The password text is masked.	-

- Select an option from the **Save Password** drop-down list. The options available are:
  - **Permanently**: Saves the password even after exiting Data Studio. While establishing the connection for the first time this option will not be

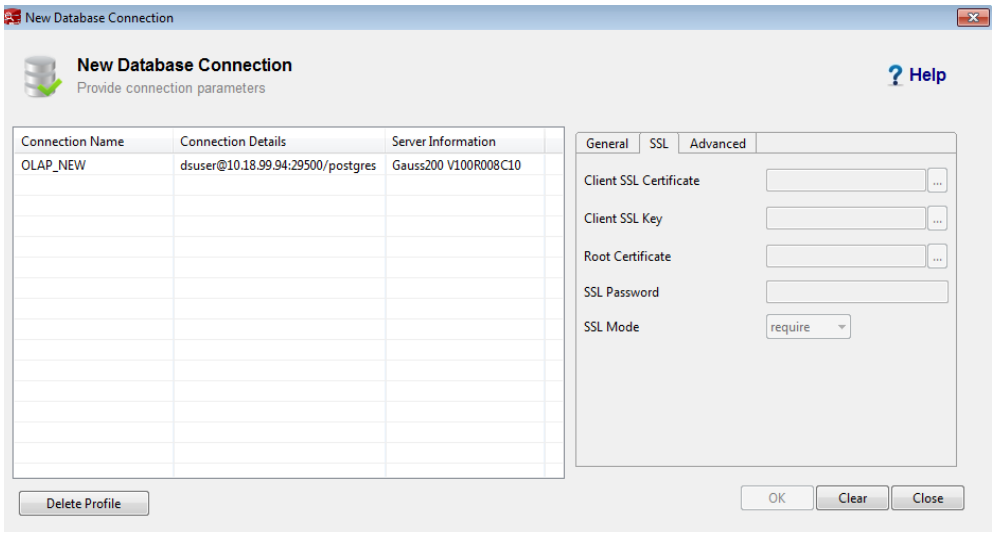
available. To hide or view this drop-down option, see the **Password** section.

- **Current Session Only**: Saves the password only for the current session.
- **Do Not Save**: Does not save the password. If this option is selected, Data Studio will prompt for the password for certain operations like:
  - **Creating a Database**
  - **Renaming a Database**
  - **Debugging a PL/SQL Function**
  - **Working with SQL Terminals**

- **Enable SSL** check box is selected by default.

Step 4 Follow the steps below to enable SSL:

1. Select the **Enable SSL** option.
2. Click the **SSL** tab.



3. Provide the following information. The following files are required for secured connection. Refer to **SSL Certificates** section.
   - To select the **Client SSL Certificate**, click ... and select the Client SSL Certificate.
   - To select the **Client SSL Key**, click ... and select the Client SSL key.
   - To select the **Root Certificate**, click ... and select the Root Certificate.
   - Select the SSL Mode from **SSL Mode** drop-down. Refer to table below for description of different SSL modes.

     NOTE

     - If **SSL Mode** is set to **verify-ca**, **Root Certificate** must be selected.
     - DS prompt for the Client key while accessing the gs_dump feature for the first time.

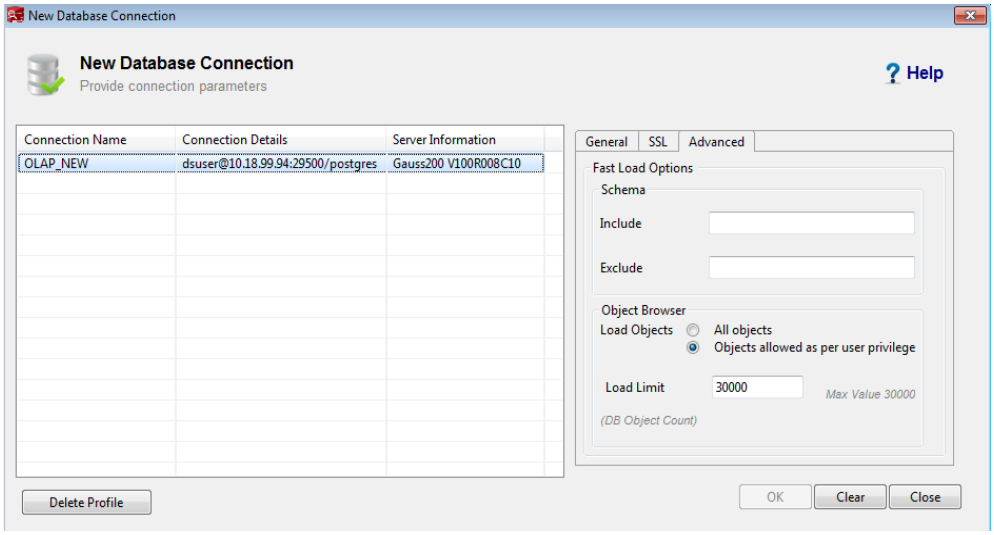
SSL Mode	Description
require	Selecting require will not check the validity of the certificates since a non-validating SSL factory is used.
verify-ca	Selecting verify-ca checks if the CA is correct using a validating SSL factory.

NOTE

- Selecting **Client SSL Certificate** and **Client SSL Key** ensures secured connection for export of DDL and data using Data Studio.
- Selecting invalid file for Client SSL Certificate and/or Client SSL Key will result in export failure. For details, see **Troubleshooting**.
- If you deselect **Enable SSL** check box and proceed, then **Connection Security Alert** dialog box is displayed. Refer to **Security Disclaimer** for information to display this security alert or not.
  - **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.
- Refer to server manuals for more details.

Step 5 Follow the steps below to set the **Fast Load Options**:

1. Click the **Advanced** tab.



2. Enter the schema names using comma separator to load on priority while establishing a connection in the **Include** field.
3. Enter the schema names using comma separator to avoid loading on priority while establishing a connection in the **Exclude** field.

4. Select an option from the **Load Objects** options. The options available are:
   - **All Objects**: Loads all objects.
   - **Objects allowed as per user privilege**: Loads only objects for which the user has permissions. For details about the minimum access permissions for objects listed in the object browser, see **Table 4-29**.

   NOTE

   The default value is **Objects allowed as per user privilege**.
5. Enter the load limit in **Load Limit** field. The maximum value allowed is 30000. This is the database object count.

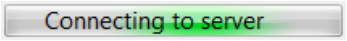
   NOTE

   - If the number of object type (tables, view..) of the schema mentioned in the **Include** field is greater than the value entered in the **Load Limit** field, then the only the parent objects for a schema will be loaded. This implies that child objects like columns, constraints, indexes, functions with more than three parameters, and so on will not be loaded.
   - Schema names provided in the Include and Exclude lists are validated.
   - If you do not have access to the schema name entered in the **Include** field, then an appropriate error message is displayed for that schema during connection.
   - If you do not have access to the schema name entered in the **Exclude** field, then the schema will not be loaded in **Object Browser** after connection is established.

Step 6 Click **OK** to establish the connection successfully.

The status bar displays the status of the completed operation.

While Data Studio is connecting to the database, the following status bar shows the status:



Once the connection is established, all schema objects will be displayed in the **Object Browser** pane.

NOTE

- Data Studio allows you to log in even if the password has expired with a message informing that some operations may not work as expected. For details, see **Password Expiry**.
- To cancel the connection, see **Canceling the Connection**.
- Postgres specific schemas are not displayed in the Object Browser.

----End

Canceling the Connection

Follow the steps below to cancel the connection:

Step 1 Click **Cancel**.

The **Cancel Connection** dialog box is displayed.

Step 2 Click **Yes**.

A message confirmation dialog box is displayed.

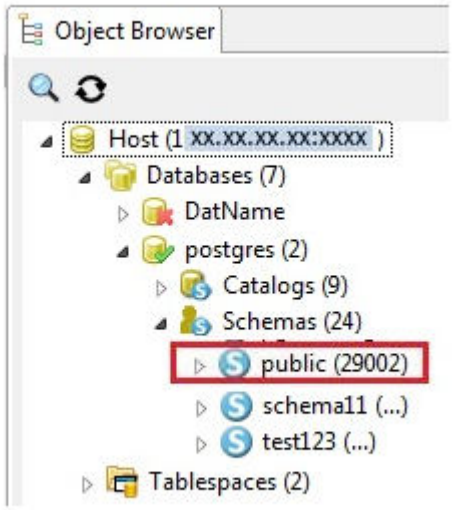
Step 3 Click **OK**.

----End

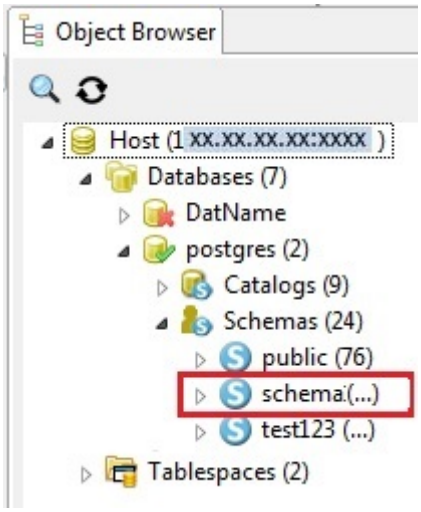
Lazy Loading

Lazy loading feature delays the loading of objects until required.

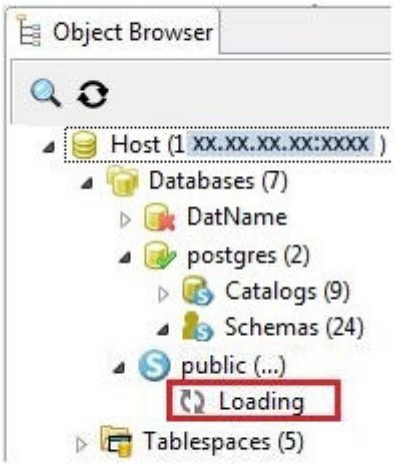
When you connect to a database only child objects of schema saved under **search_path** will be loaded as shown below:



Unloaded schemas are displayed as *schema name* **(...)**.



To load child objects, expand the schema. During expansion of schema, the objects under the schema will show as loading:



NOTE

If you try to load an unloaded object while loading is in progress for another object, a pop-up message is displayed informing you that another loading is in progress. The icon next to the unloaded object disappears. Refresh at the object or database level to display this icon again for loading.

Expand schema to load and view the child objects. The Object Browser can load child objects of only one schema at a time.

If **search_path** is modified after establishing connection, then the changes will be reflected only after reconnecting the database. Auto-suggest works on keywords, data types, schema names, table names, views, and table name aliases for all schema objects that you have access.

A maximum of 50,000 objects will be loaded in the **Object Browser** pane within 1 minute.

The database connection timeout interval defaults to 3 minutes (180 seconds). If the connection fails within this interval, a timeout error is displayed.

You can set the **loginTimeout** value in the **Data Studio.ini** file located in the **Data Studio** directory.

NOTE

When you log in to the Data Studio, **pg_catalog** is loaded automatically.

4.8.3 Renaming a Connection

Follow the steps below to rename a database 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 provide the new name for the connection.

Step 2 Enter the new connection name. Select **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

4.8.4 Editing a Connection

Follow the steps below to edit the database connection properties:

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

Editing an active connection will require closing the connection and then reopening the connection with the new properties. A warning message about connections being reset is shown.

The **Edit Connection** dialog box is displayed.

Step 2 Click **OK** to proceed or **Cancel** to exit the operation.

NOTE

The **Connection Name** field cannot be modified.

Step 3 Edit the connection parameters. Connection parameters are explained in **Adding a Connection**.

Step 4 Click **OK** to save the updated connection information.

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.
- 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, see **Password Expiry**.
- Refer to **Canceling the Connection** to cancel the connection.

If SSL is not enabled, a **Connection Security Alert** dialog box is displayed.

Step 5 Click **Continue** to proceed with insecure connections or click **Cancel** to return to the **Edit Connection** dialog box to enable SSL.

NOTE

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

The **Remove Server Confirmation** is displayed to confirm closing databases for the edited connection.

Step 6 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

4.8.5 Removing a Connection

Follow the steps below to remove an existing database connection:

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

A confirmation dialog box is displayed to remove the connection.

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

The status bar displays the status of the completed operation.

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

----End

4.8.6 Viewing Connection Properties

Follow the steps below to view the properties of a connection:

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 is displayed.

NOTE

If the property of a connection is modified for the connection that is already opened, then open the properties of the connection again to view the updated information on the same opened window.

----End

4.8.7 Refreshing a Database Connection

Follow the steps below to refresh the database connection:

Step 1 In the **Object Browser** pane, right-click the selected connection name and select **Refresh** or press **F5**.

The status bar displays the status of the completed operation.

----End

The time taken to refresh the database depends on the number of objects present in the database. For a large database, it is recommended to perform this operation only if required.

- If you right-click the connection name and select **Refresh**, the connection profile is refreshed. During refresh, the connection will be updated with the latest copy from the server.
- If you right-click the Schema and select **Refresh**, all functions/procedures and tables under the schema are refreshed. During refresh, all functions/procedures and tables are updated with the latest copy from the server.

If any stored function/procedure is deleted from the database before the refresh operation, then it will be removed from the **Object Browser** only after you perform the refresh operation.

- If you right-click a specific function/procedure and select **Refresh**, the specific function/procedure is refreshed. During refresh, the specific function/procedure is updated with the latest copy from the server.
- If you refresh at database level or connection profile level, then all the child objects of schema in **search_path** along with the schema already expanded by the user will be loaded.
- If you re-connect to the Database, then only schema objects saved under **search_path** will be loaded. Previously expanded objects will not be loaded.
- Database and multiple objects under a database cannot be refreshed simultaneously.

Exporting/Importing Connection Details

Data Studio provides the option to export/import connection details from the connection dialog for future reference.

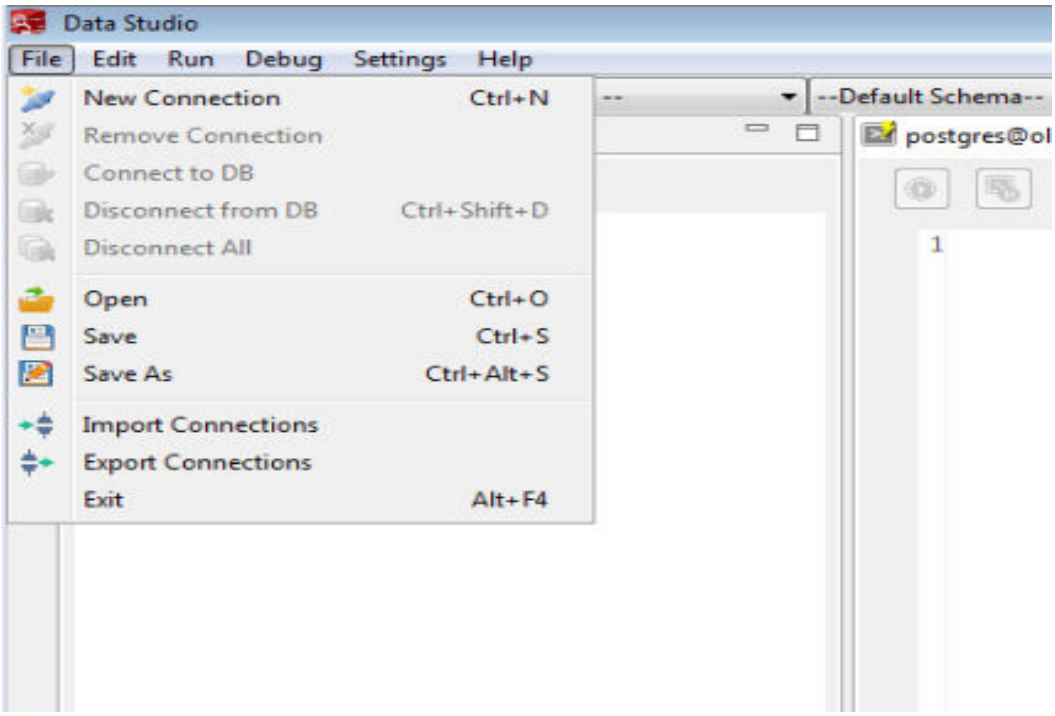
The following fields are exported:

- SSL Mode
- Connection name
- Server IP
- Server Port
- Database Name
- Username
- clSSLCertificatePath
- clSSLKeyPath
- profileId
- rootCertFilePathText
- connctionDriverName
- schemaExclusionList
- schemaInclusionList
- loadLimit
- privilegeBasedObAcess
- databaseVersion
- savePrdOption
- dbType
- version

Follow the steps to access the feature:

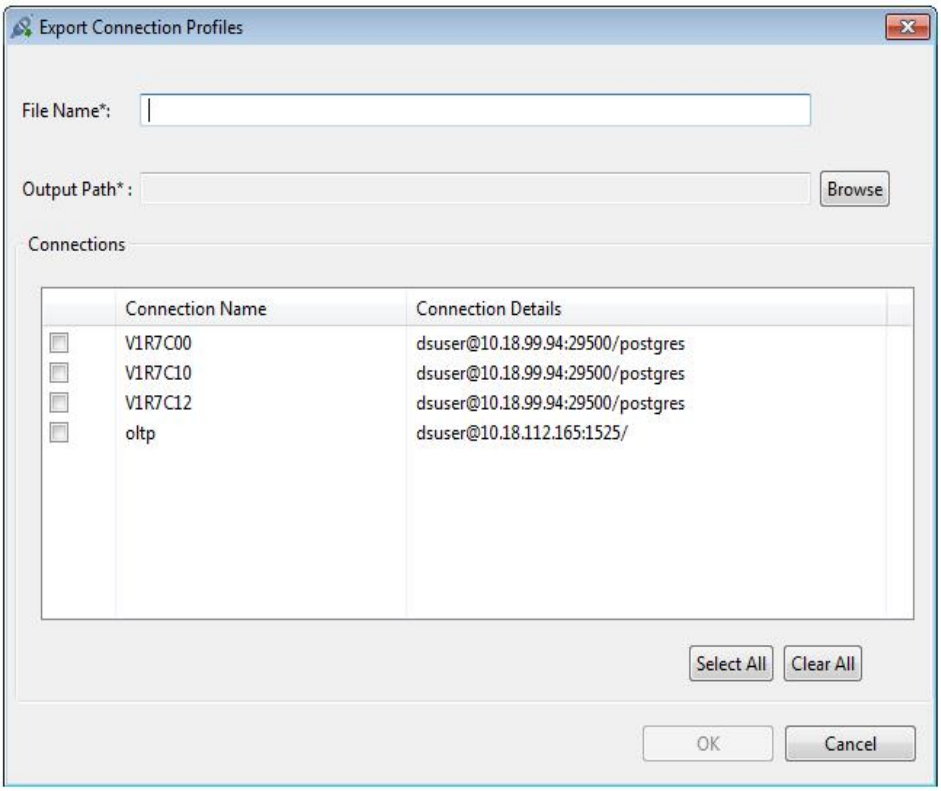
Step 1 Click **File** on Menu Bar.

The following window is displayed:



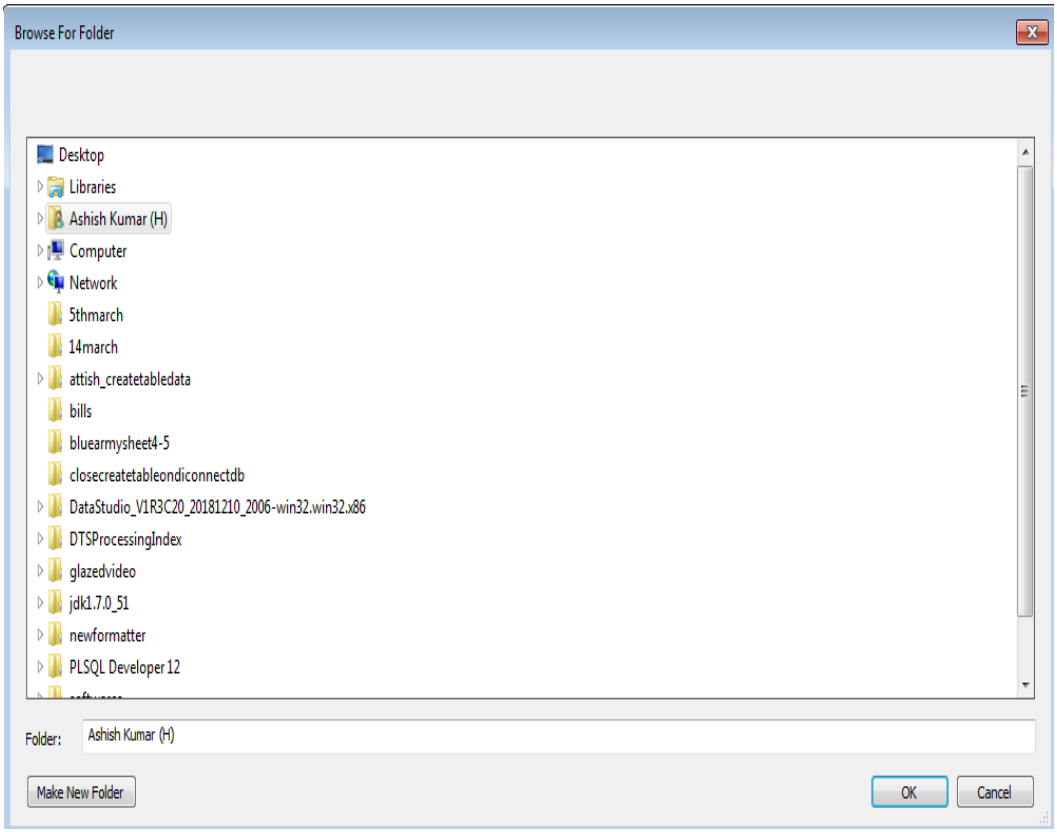
Step 2 Select **Export Connections** to export the configuration files.

The **Export Connection Profiles** dialog box is displayed. You can select the links to be exported in this window.

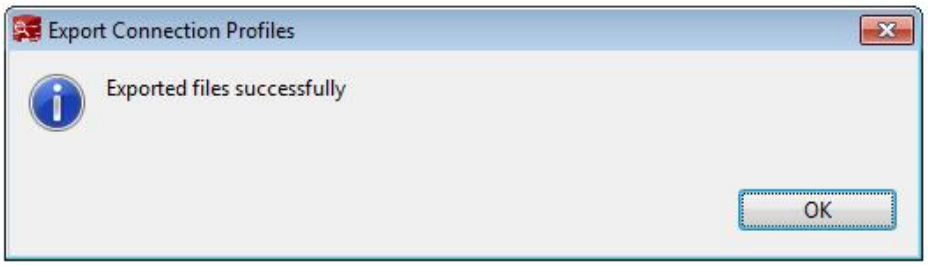


Select the connections you want to export and enter a file name where the exported connections will be saved. Click **OK**.

Select the location where you want to save the file and click **OK**.

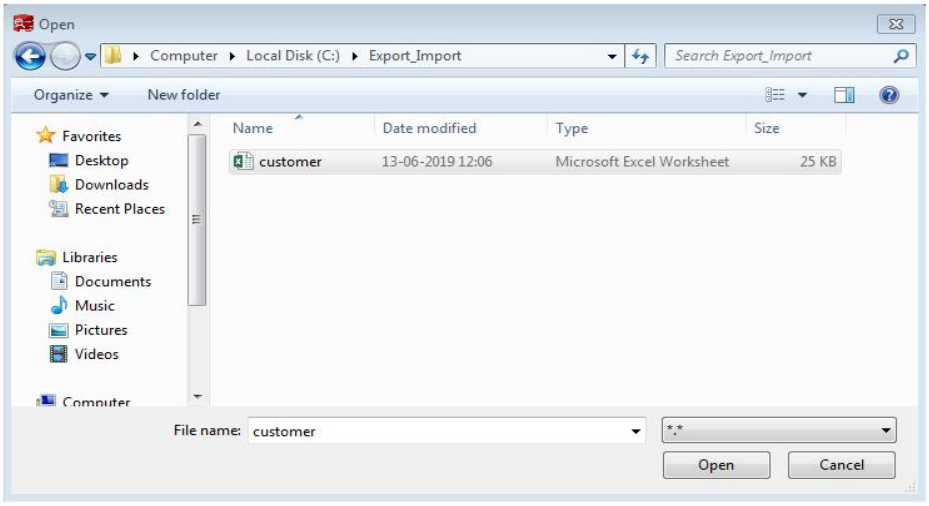


A dialog box is displayed once the connections are exported successfully.

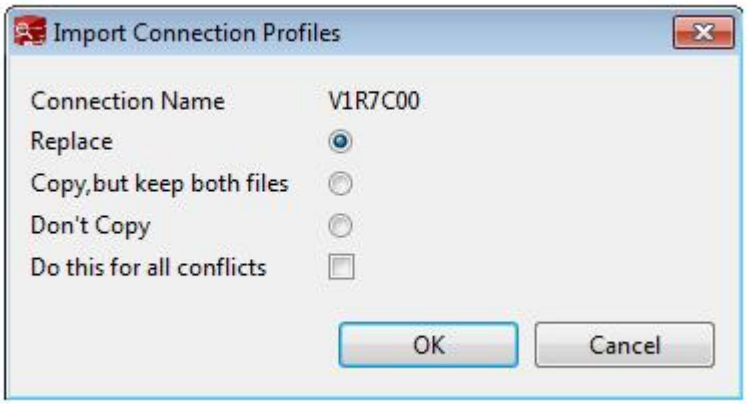


Step 3 Select **Import Connections** to import the configuration files.

Step 4 Select the file you want to import and click **Open**.



If the connection to be imported matches the existing connection, a dialog box is displayed as follows:



- **Replace**: The imported connection configuration file will be replaced with the existing one.
- **Copy, but keep both files**: The imported connection configuration file will be renamed.
- **Don't Copy**: The existing connection configuration file will remain unchanged.
- **Do this for all conflicts**: The same operation will be repeated for all the matches.

Click any of the given options as required and click **OK**.

----End

NOTE

Password and SSL password fields will not be exported.

4.9 Databases

4.9.1 Creating a Database

A relational database is a database that has a set of tables which is manipulated in accordance with the relational model of data. It contains a set of data objects used to store, manage, and access data. Examples of such data objects are tables, views, indexes, functions and so on.

Follow the steps below to create a database:

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

NOTE

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

A **Create Database** dialog box is displayed prompting you to provide the necessary information to create the database.

Step 2 Enter the database name. Refer to the server manual for database naming rules.

Step 3 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** types of encoding character sets. Creating the database with other encoding character sets may result in erroneous operations.

Step 4 Select the **Connect to the DB** check box 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 present in the server is automatically added to the new database.

NOTE

Data Studio allows you to login even if the password has expired with a message informing that some operations may not work as expected when no other database is connected in that connection profile. Refer to **Password Expiry** for information to change this behavior.

----End

Cancelling Connection

Follow the steps below to cancel the connection operation:

Step 1 Double-click the status bar to open the **Progress View** tab.

Step 2 In the **Progress View** tab, click  .

Step 3 In the **Cancel Operation** dialog box, click **Yes**.

The status bar displays the status of the cancelled operation.

----End

4.9.2 Disconnecting All Databases

You can disconnect all the databases from a connection.

Follow the steps below to disconnect all the databases from a connection:

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 that connection.

NOTE

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

A confirmation dialog box is displayed to disconnect all databases for the connection.

Step 2 Click **Yes** to disconnect.

The status bar displays the status of the completed operation.

Data Studio populates all the connection parameters (except password) that were provided during the last successful connection with the database. To reconnect, you need to enter only the password in the connection wizard.

----End

4.9.3 Connecting to a Database

You can connect to the database.

Follow the steps below to connect a database:

Step 1 In the **Object Browser** pane, right-click the selected database name and select **Connect to DB**.

NOTE

This operation can be performed only on an inactive database.

The database is connected.

The status bar displays the status of the completed operation.

NOTE

- Data Studio allows you to login even if the password has expired with a message informing that some operations may not work as expected when no other database is connected in that connection profile. Refer to **Password Expiry** for information to change this behavior.
- Refer to **Cancelling Connection** section to cancel the connection to database.

----End

4.9.4 Disconnecting a Database

You can disconnect the database.

Follow the steps below to disconnect a database:

Step 1 In the **Object Browser** pane, right-click the selected database name and select **Disconnect from DB**.

NOTE

This operation can be performed only on an active database.

A confirmation dialog box is displayed to disconnect database.

Step 2 Click **Yes** to disconnect.

The database is disconnected.

The status bar displays the status of the completed operation.

----End

4.9.5 Renaming a Database

Follow the steps below to rename a database:

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

NOTE

This operation can be performed only on an inactive database.

A **Rename Database** dialog box is displayed prompting you to provide the necessary information to rename the database.

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

A confirmation dialog box is displayed to rename the database.

Step 3 Click **OK** to rename the database.

The status bar displays the status of the completed operation.

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

NOTE

Refer to **Cancelling Connection** section to cancel the connection to database.

----End

4.9.6 Dropping a Database

Individual or batch drop can be performed on databases. Refer to **Dropping a Batch of Objects** section for batch drop.

Follow the steps below to drop a database:

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

NOTE

This operation can be performed only on an inactive database.

A confirmation dialog box is displayed to drop the database.

Step 2 Click **OK** to drop the database.

A popup message and the status bar display the status of the completed operation.

----End

4.9.7 Viewing Properties of a Database

Follow the steps below to view the properties of a database:

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

NOTE

This operation can be performed only on an active database.

The status bar displays the status of the completed operation.

The properties of the selected database are displayed.

NOTE

If the property of a database is modified for the database that is already opened, then refresh and open the properties of the database again to view the updated information on the same opened window.

----End

4.10 Schemas

4.10.1 Overview

This section describes working with database schemas. All system schemas are grouped under **Catalogs** and user schemas under **Schemas**.

4.10.2 Creating a Schema

In relational database technology, schemas provide a logical classification of objects in the database. Some of the objects that a schema may contain include functions/procedures, tables, sequences, views, and indexes.

Follow the steps below to define a 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 the schema name and click **OK**. You can create the schema only if the database connection is active.

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

The status bar displays the status of the completed operation.

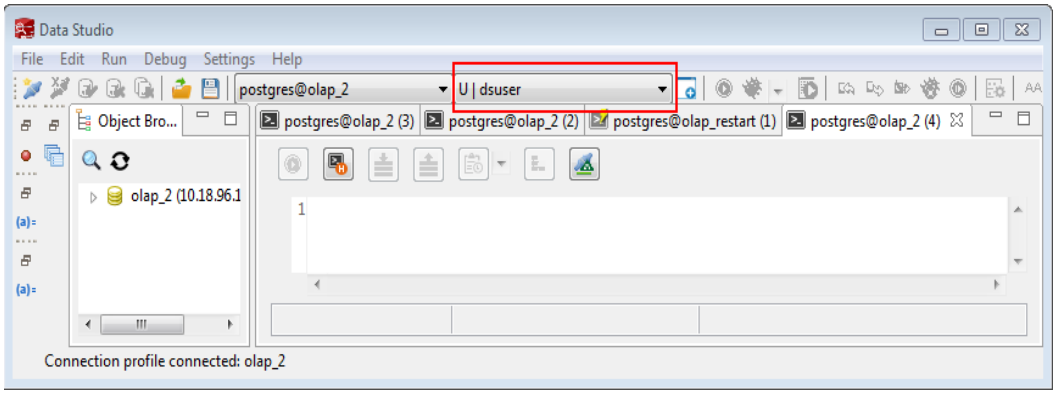
----End

You can perform the following actions on a schema:

- Refreshing a schema - To refresh a schema, right-click the selected **Schema Name** and select **Refresh Schema**. All the objects under that schema will be refreshed.
- Renaming a schema (Refer to **Renaming a Schema** for more details)
- Dropping a schema (Refer to **Dropping a Schema** for more details)
- Exporting DDL (Refer to **Exporting Schema DDL** for more details)
- Exporting DDL and data (Refer to **Exporting Schema DDL and Data** for more details)
- Grant/Revoke privilege (Refer to **Granting/Revoking a Privilege** for more details)

Displaying the Default Schema

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 default schemas are searched to find these objects.

When Data Studio starts, the default schemas are set to **<username>**, public schemas in 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).

NOTE

This feature is not available for OLTP database.

4.10.3 Exporting Schema DDL

Exporting the schema DDL exports the DDLs of functions/procedures, tables, sequences and views of the schema.

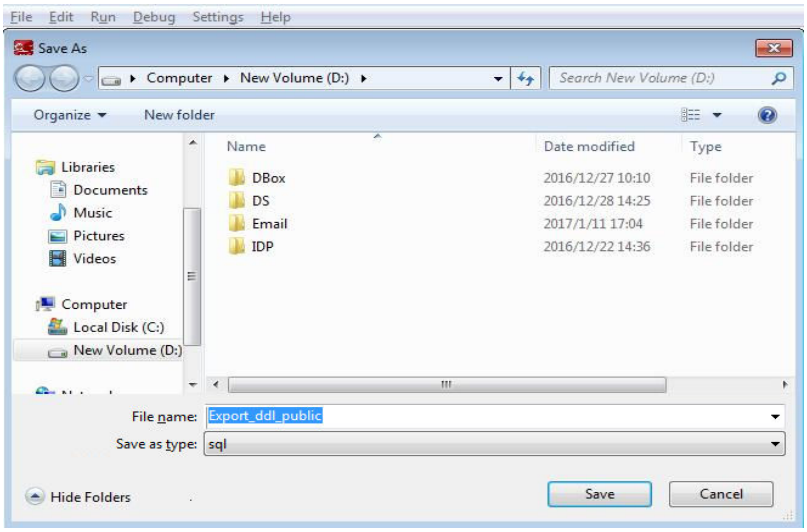
Follow the steps to export the schema DDL:

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

The **Data Studio Security Disclaimer** dialog box is displayed. You can close this security disclaimer message. For details, see **Security Disclaimer**.

Step 2 Click **OK**.

The **Save As** dialog box is displayed.



Step 3 In the **Save As** dialog box, select the location to save the DDL and click **Save**. The status bar displays the progress of the operation.

NOTE

- To cancel the export operation, double-click the status to open the **Progress View** tab and click . For details, see **Canceling the Table Data Export Operation**.
- If the file name contains characters that are not supported by Windows, the file name will be different from the schema name.
- MS Visual C runtime file (msvcrt100.dll) is required to complete this operation. For details, see **Troubleshooting**.

The **Export** message and the status bar display the status of the completed operation.

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

NOTE

You can select multiple objects and export their DDL. **Batch Export** lists the objects whose DDL cannot be exported.

----End

4.10.4 Exporting Schema DDL and Data

The exported schema DDL and data include the following:

- DDLs of functions/procedures of the schema.
- DDLs and data of tables of the schema.
- DDLs of views of the schema.
- DDLs of sequences of the schema.

Follow steps below to export the schema DDL and data:

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

The **Data Studio Security Disclaimer** dialog box is displayed.

You can close this security disclaimer message. For details, see **Security Disclaimer**.

Step 2 Click **OK**.

The **Save As** dialog box is displayed.

Step 3 In the **Save As** dialog box, select the location to save the DDL and data and click **Save**. The status bar displays the progress of the operation.

NOTE

- To cancel the export operation, double-click the status to open the **Progress View** tab and click  . For details, see **Canceling the Table Data Export Operation**.
- The exported file name will not be the same as schema name, if the schema name contains characters which are not supported by Windows.
- MS Visual C runtime file (msvcrt100.dll) is required to complete this operation. For details, see **Troubleshooting**.

The **Export** message and the status bar display the status of the completed operation.

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

NOTE

You can select multiple objects and export their DDL and data. **Batch Export** lists the objects whose DDL and data cannot be exported.

----End

4.10.5 Renaming a Schema

Follow the steps to rename a schema:

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

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

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

The status bar displays the status of the completed operation.

----End

4.10.6 Supporting Sequence DDL

Data Studio provides the option to show sequence DDL or allow users to export sequence DDL. It provides "Show DDL", "Export DDL", "Export DDL and Data"

Follow the steps to access the feature:

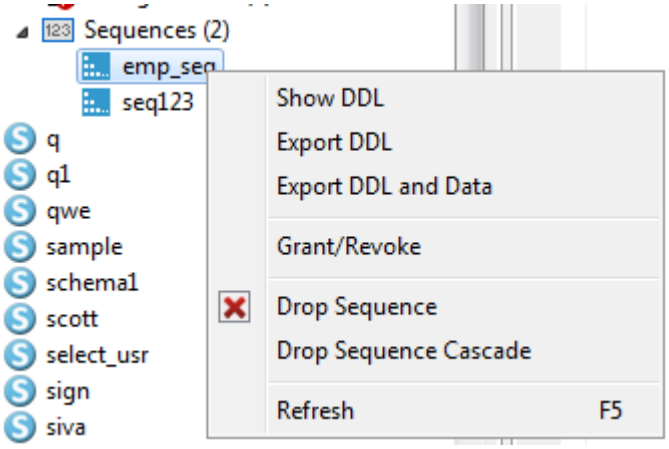
Step 1 In **Object Browser**, right click on any object under **Sequences**. A menu option will appear

Step 2 Select **Show DDL** option to see the DDL statements.

Or Select the **Export DDL** menu option to export DDL statements.

Or Select the **Export DDL and Data** menu option to export DDL statements and the select statement.

Refer to the following image:



NOTE

Only the sequence owner or sysadmin or has the **select** privilege of the sequence, then only the operation can be performed.

----End

4.10.7 Granting/Revoking a Privilege

Follow the steps below to grant/revoke a privilege:

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

The **Grant/Revoke** dialog 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 in the **Privilege Selection** tab.

Step 4 Select **Grant/Revoke** in the **Privilege Selection** tab.

Step 5 Select/unselect the required privileges in the **Privilege Selection** tab.

In **SQL Preview** tab, you can view the SQL query automatically generated for the inputs provided.

Step 6 Click **Finish**.

----End

4.10.8 Dropping a Schema

Individual or batch dropping can be performed on schemas. Refer to **Dropping a Batch of Objects** section for batch dropping.

Follow the steps below to drop a schema:

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

A confirmation dialog to drop the schema is displayed.

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

A popup message and the status bar display the status of the completed operation.

----End

4.11 Creating a Function/Procedure

Perform the following steps to create a function/procedure and SQL function:

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 PL/SQL Function**, **Create SQL Function**, **Create PL/SQL Procedure**, or **Create C Function** as required.

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

Step 2 Add the function/procedure by right-clicking the tab and selecting **Compile**, or choosing **Run > Compile/Execute Statement** from the main menu, or pressing **Ctrl+Enter** to compile the procedure.

The **Created function/procedure Successfully** dialog box is displayed, and the new function/procedure is displayed under the **Object Browser**. Click **OK** to close the **NewObject()** tab and add the debugging object to **Object Browser**.

Refer to **Execute SQL Queries** for re-obtaining connection options when connections are lost during execution.

Step 3 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.

NOTE

- C functions do not support debugging operations.
- A popup message displays the status of the completed operation, which is not displayed in the status bar.

----End

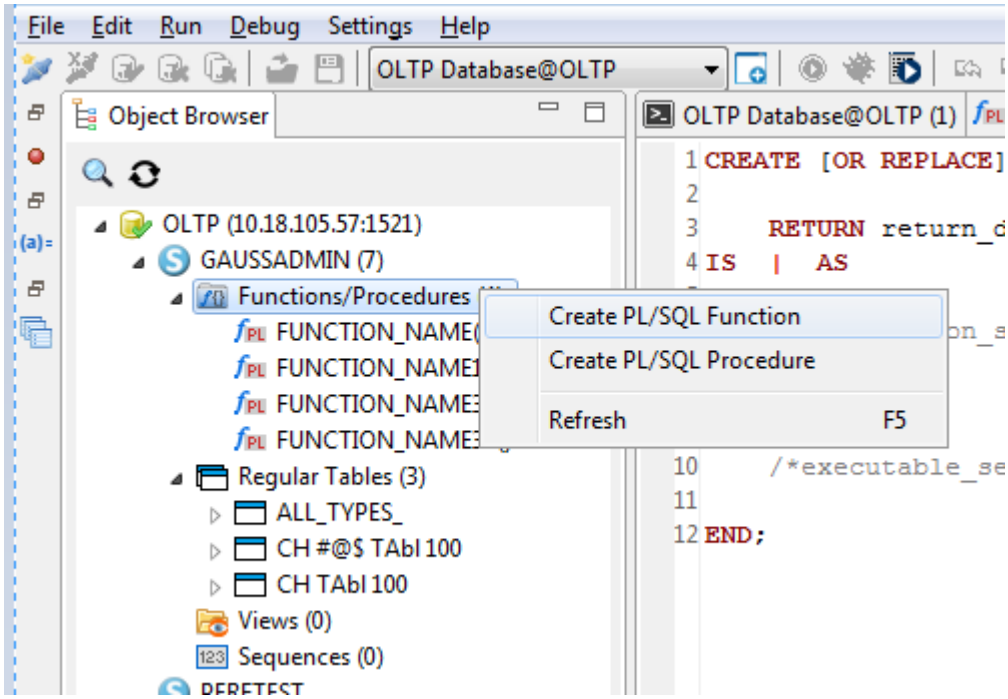
Compiling a Function

When a user creates a PL/SQL object from the template or by editing an existing PL/SQL object, the created PL/SQL object will be displayed in a new tab page.

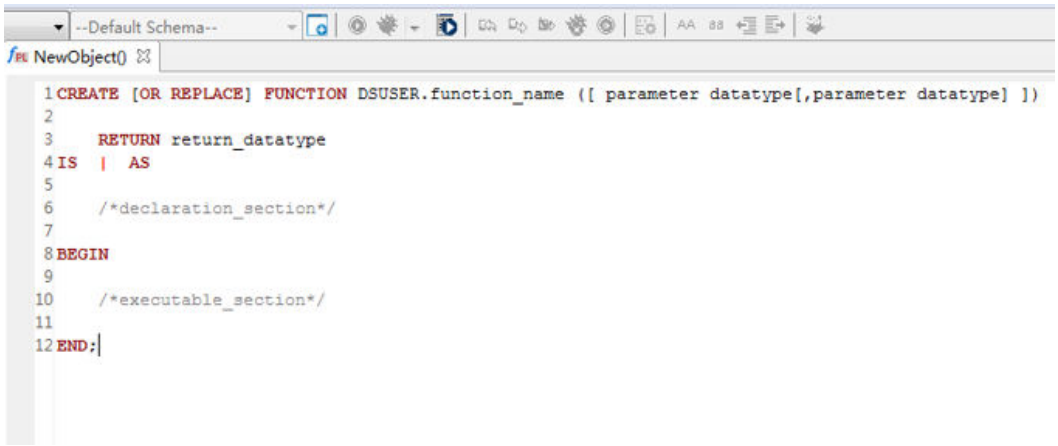
Perform the following steps to compile a created function:

Step 1 Select **Functions/Procedures** from the **Object Browser** tab page.

Step 2 Right-click **Functions/Procedures** and a menu is displayed.

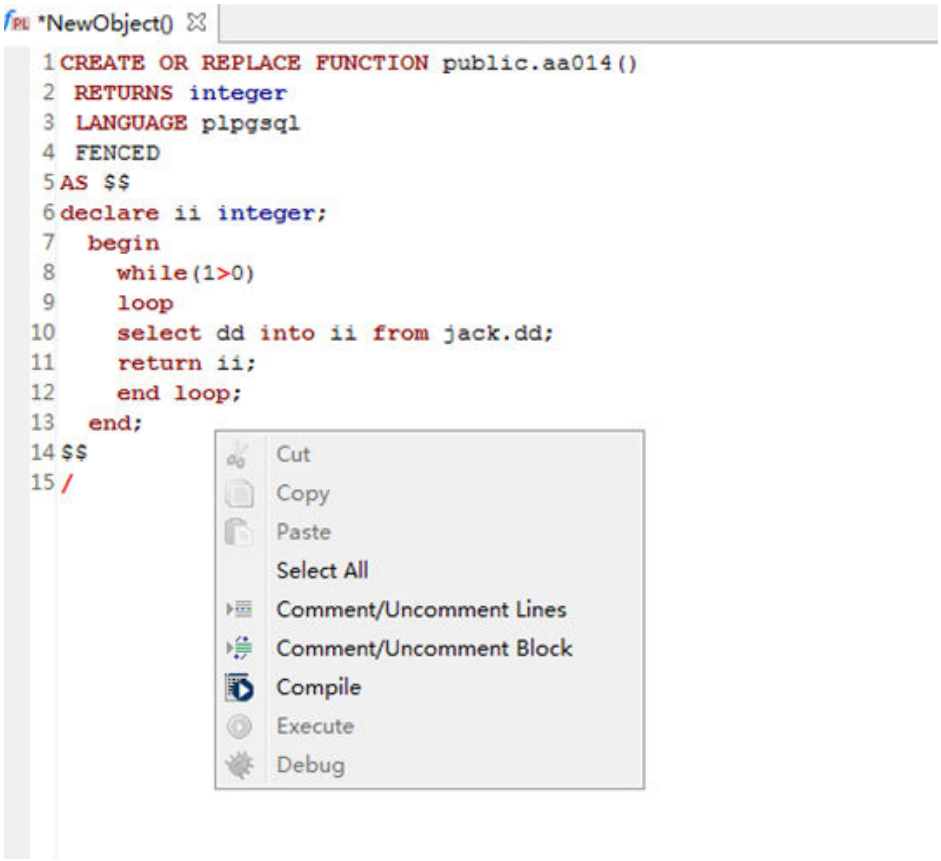


Step 3 Click **Create PL/SQL Function** and a new tab page is opened.

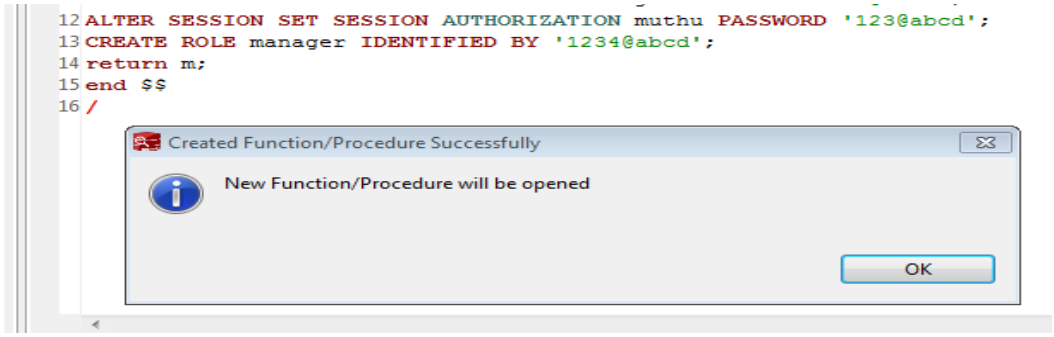


Step 4 Edit the code.

Step 5 Right-click the blank area of the tab page and a menu is displayed.



Step 6 Click **Compile**. A pop-up message is displayed as follows.



The function is displayed in a new tab page.

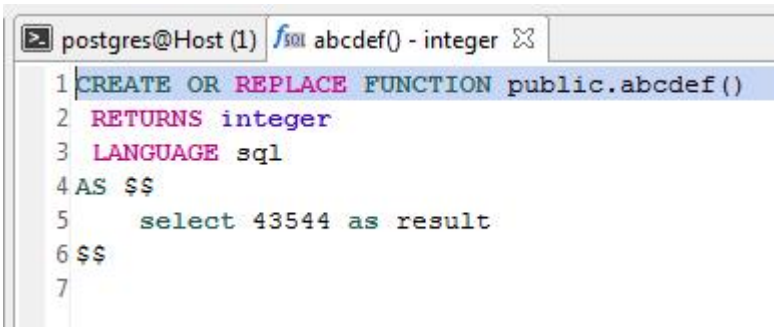
----End

4.12 Editing a Function/Procedure

Follow the steps below to open and edit the function/procedure or SQL function:

Step 1 In the **Object Browser** pane, double-click the required function/procedure or SQL function or right-click the function/procedure or SQL function and select **View Source**. You must refresh the **Object Browser** to view the latest DDL.

The function/procedure or SQL function based on your selection is displayed.



Only one function/procedure or SQL function with the same schema, name, and input parameters can be opened in Data Studio.

Step 2 After editing or updating, compile and execute the PL/SQL program or SQL function. For more details, refer to **Executing a Function/Procedure**.

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

Step 3 Click **Yes** to compile and execute the function/procedure.

The status of the completed operation is displayed on the **Message** tab page.

Refer to the **Execute SQL Queries** for information on reconnect option in case connection is lost during execution.

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

----End

4.13 Granting/Revoking a Permission (Function/ Procedure)

Perform the following steps to grant or revoke a permission:

Step 1 Right-click functions/procedures group and select the **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**.

The **Privilege Selection** tab is displayed.

Step 3 Select the role from the **Role** drop-down list.

Step 4 Select **Grant/Revoke**.

Step 5 Select or deselect the required permissions.

The **SQL Preview** tab displays the SQL query automatically generated for the inputs provided.

Step 6 Click **Finish**.

----End

NOTE

This feature is only supported in OLAP, not in OLTP.

4.14 Debugging a PL/SQL Function

4.14.1 Overview

During debugging, if the connection is lost but the database remains connected to Object Browser, the **Connection Error** dialog box is displayed with the following options:

- **Yes**: Establishes the connection again and restarts the debugging.
- **No**: Disconnects the database from Object Browser.

NOTE

SQL language function does not support debugging operations.

4.14.2 Using Breakpoints

This section contains the following topics:

- **Using the Breakpoints Pane**
- **Setting or Adding Breakpoints on a Row**
- **Enabling or Disabling a Breakpoint on a Row**
- **Removing a Breakpoint on a Row**
- **Changing Source Code**
- **Debugging a PL/SQL Program Using a Breakpoint**

A breakpoint is used to suspend the execution of a PL/SQL program at the row where the breakpoint is set. You can use breakpoints to control the execution and debug the function.

- An enabled breakpoint suspends the execution of the PL/SQL program whenever a breakpoint is encountered. When the execution hits the row of breakpoint, the execution will stop and you will be able to carry out other debug operations. Data Studio supports the following breakpoint operations:
  - Setting or adding breakpoint on a row
  - Enabling or disabling a breakpoint on a row
  - Removing a breakpoint on a row
- A disabled breakpoint will not suspend execution of PL/SQL program.

When you run a PL/SQL program, the execution pauses at every row where you set a breakpoint. When the program execution is paused, Data Studio retrieves information about the current program state, such as the values of the program variables.

Perform the following steps to debug a PL/SQL program:

Step 1 Set a breakpoint at the row where PL/SQL program execution should be paused.

Step 2 Start the debugging session.

When a row with a breakpoint is reached, monitor the state of the application in the debugger pane, and continue the execution.

Step 3 Close the debugging session.

----End

Data Studio provides debugging options in the toolbar that helps you step through the debug objects.

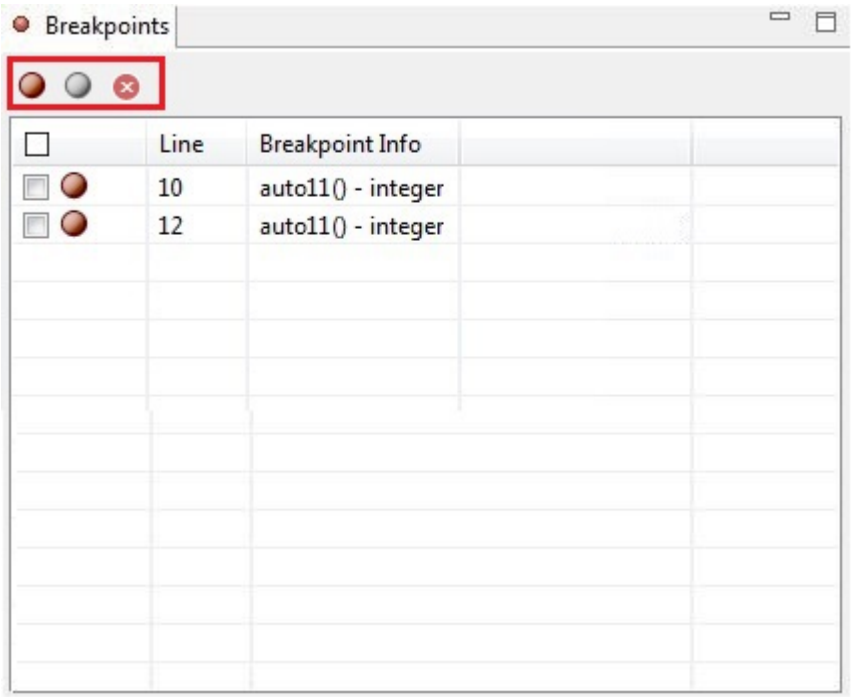
Using the Breakpoints Pane

You can use the **Breakpoints** pane to view and manage the currently set breakpoints. From the minimized window panel, click the breakpoint option to open the **Breakpoints** pane.

The **Breakpoints** pane lists each breakpoint with the row number and the debug object name.

You can enable or disable all the breakpoints by clicking ☐ in the **Breakpoints** pane. You can enable, disable or remove a specific breakpoint by selecting the breakpoint check box and clicking , or in the **Breakpoints** pane.

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



NOTE

- Disabling a breakpoint prevents the execution from pausing at the breakpoint, but leaves the definition in place (to enable the breakpoint later).
- Deleting a breakpoint removes it permanently.
- The content of the **Breakpoints** pane can be copied to the clipboard using **Alt**+**Y**.

Setting or Adding Breakpoints on a Row

Follow the steps to set or add breakpoints on a row:

Step 1 Open the PL/SQL function on a row where you want to add a breakpoint.

Step 2 In the **PL/SQL Viewer**, double-click the breakpoint ruler on the left side of the **Line** column. The added breakpoint is indicated by an enable breakpoint sign [] in the **PL/SQL Viewer**.

NOTE

If the execution of the function does not break or stop the breakpoint during debugging, the breakpoint that is already set will not be validated.

----End

Enabling or Disabling a Breakpoint on a Row

Once a breakpoint is set, you can temporarily disable it by selecting the corresponding check box in the left-side of the **Breakpoints** pane and clicking  at the top of the **Breakpoints** pane. Disabled breakpoints will be grayed out [] in the **PL/SQL Viewer** and **Breakpoints** pane. To enable a disabled breakpoint, select the corresponding breakpoint (using check box) and click  .

Removing a Breakpoint on a Row

You can remove an unused breakpoint using the same method as that for creating a breakpoint.

In the **PL/SQL Viewer** tab, open the function in which you want to remove the breakpoint. Double-click  in the **PL/SQL Viewer** to disable the breakpoint. The breakpoint is removed from the work area.

You can also enable or disable breakpoints using the preceding method.

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.

NOTE

If the source code is changed after it is fetched from the server, and if you perform the execution or debug operation with no breakpoint set, then the result of the source code at the server will be displayed on Data Studio. You are advised to refresh before performing debug or execute operation.

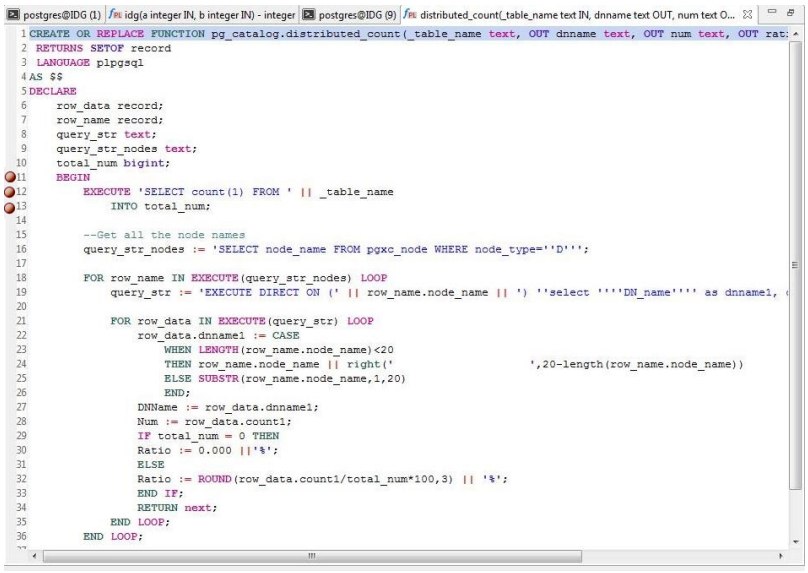
Debugging a PL/SQL Program Using a Breakpoint

Perform the following steps to debug a PL/SQL program using a breakpoint:

Step 1 Open the PL/SQL program and add a breakpoint on the row to be debugged.

An example is as follows:

Rows 11, 12, 13




```
1  CREATE OR REPLACE FUNCTION pg_catalog.distributed_count(_table_name text, OUT dnname text, OUT num text, OUT rat:
2  RETURNS SETOF record
3  LANGUAGE plpgsql
4  AS $$
5  DECLARE
6      row_data record;
7      row_name record;
8      query_str text;
9      query_str_nodes text;
10     total_num bigint;
11     BEGIN
12         EXECUTE 'SELECT count(1) FROM ' || _table_name
13             INTO total_num;
14
15         --Get all the node names
16         query_str_nodes := 'SELECT node_name FROM pgxc_node WHERE node_type=''D''';
17
18         FOR row_name IN EXECUTE(query_str_nodes) LOOP
19             query_str := 'EXECUTE DIRECT ON (' || row_name.node_name || ') ''select ''''DN_name'''' as dnname1, (
20
21             FOR row_data IN EXECUTE(query_str) LOOP
22                 row_data.dnname1 := CASE
23                     WHEN LENGTH(row_name.node_name)<20
24                     THEN row_name.node_name || right('                    ',20-length(row_name.node_name))
25                     ELSE SUBSTR(row_name.node_name,1,20)
26                     END;
27                 DNName := row_data.dnname1;
28                 Num := row_data.count1;
29                 IF total_num = 0 THEN
30                 Ratio := 0.000 ||'%';
31                 ELSE
32                 Ratio := ROUND(row_data.count1/total_num*100,3) || '%';
33                 END IF;
34                 RETURN next;
35             END LOOP;
36         END LOOP;
```

Step 2 To start debugging, click or press **Ctrl**+**D**, or right-click the selected PL/SQL program in the **Object Browser** and select **Debug**. In the **Debug Function/Procedure** dialog box, enter the parameter information.

NOTE

If there is no input parameter, the **Debug Function/Procedure** dialog box will not be displayed.

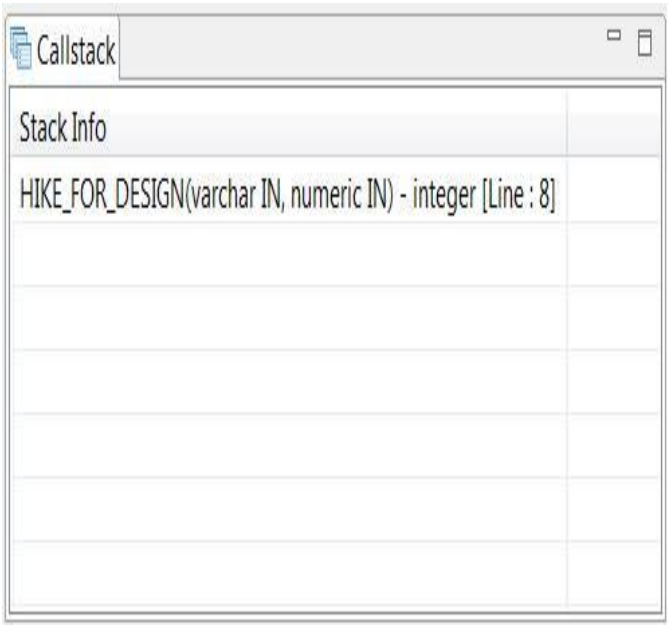
Step 3 Enter the information and click **OK**. Single quotation marks (') are mandatory for the parameters of **varchar** and **date** data types, but not mandatory for the parameters of **numeric** data type.

To set NULL as the parameter value, enter **NULL** or **null**.

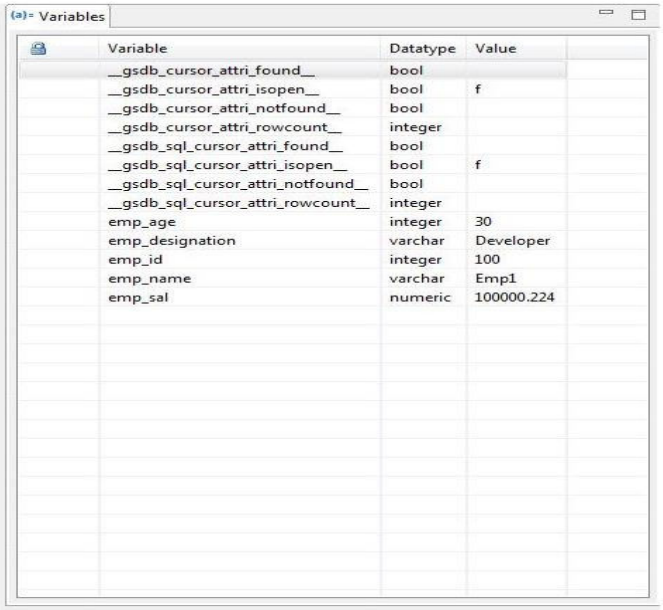
On clicking the **Debug** button, you will see an arrow pointing to the row where the breakpoint is set. The arrow indicates the row number at which execution will resume from.

You can terminate debugging by clicking from the toolbar, or pressing **F10**, or select **Terminate Debugging** from the **Debug** menu. After the debugging is complete, the function execution proceeds and will not be terminated at any breakpoint.

The **Callstack** and **Variables** panes are populated.



The **Variables** pane shows the current value of variables. Mouse over the variable in the function/procedure also shows the current value of variables.



You can step through the code using **Step Into**, **Step Out** or **Step Over**. For details, see **Controlling Execution**.

Step 4 Click **Continue** to continue the execution till the next breakpoint (if any). The result of the executed PL/SQL program is displayed in the **Result** tab and the **Callstack** and **Variables** panes are cleared. You can copy the content of the **Result** tab, by clicking .

To remove the breakpoint, do the following:

- Double-click again on the breakpoint to remove it from the **PL/SQL Viewer**.
- Select the breakpoint in the breakpoint check box and click in the **Breakpoints** pane.

----End

Rearranging the Variable Window

This feature enables the Variable Window and columns to be rearranged. You are able to arrange Variable Window to the following places:

- Next to the **SQL Assistant** tab
- Next to the **SQL Terminal** tab
- Next to the **Object Browser** tab
- Next to the **Resultset** tab
- Next to the **Breakpoints** tab
- Next to the **Callstack** tab
- Next to the **Object Browser** tab

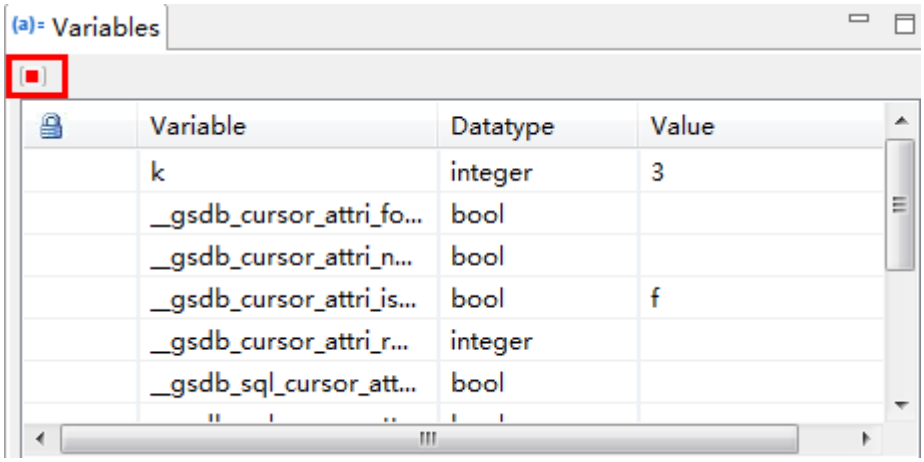
NOTE

When debugging is finished, the variable window will be minimized even if the variable window is rearranged while debugging. If variable window is rearranged as the **Terminal** tab or the **Result** tab, on completion of debugging, the tab should be minimized manually. The position of variable window is maintained after it is rearranged.

Enabling/Disabling System Variables

System Variables are displayed by default. You can disable the system variables whenever required.

Step 1 Click the red button under **Variables** to disable system variables.



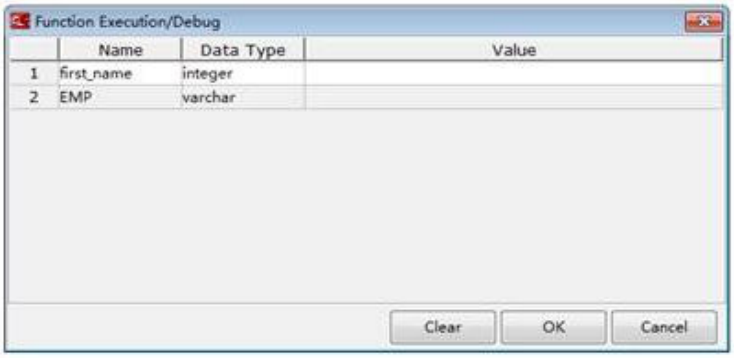
The button is in **ON** state by default.

----End

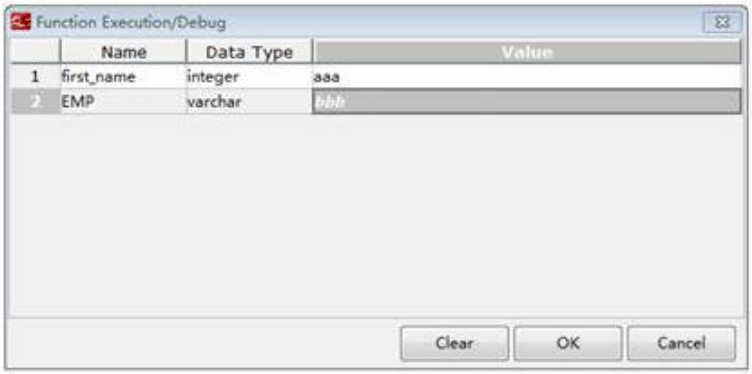
Displaying Cached Parameters

When a PL/SQL function or procedure is debugged or executed, the same parameter values are used for the next debugging or execution.

While executing a PL/SQL object, following window is displayed:



For the first time, parameter values are empty. Enter the value as required.



Click **OK**. The parameter values will be cached. Next time during the query execution/debug same parameter values will be displayed.

NOTE

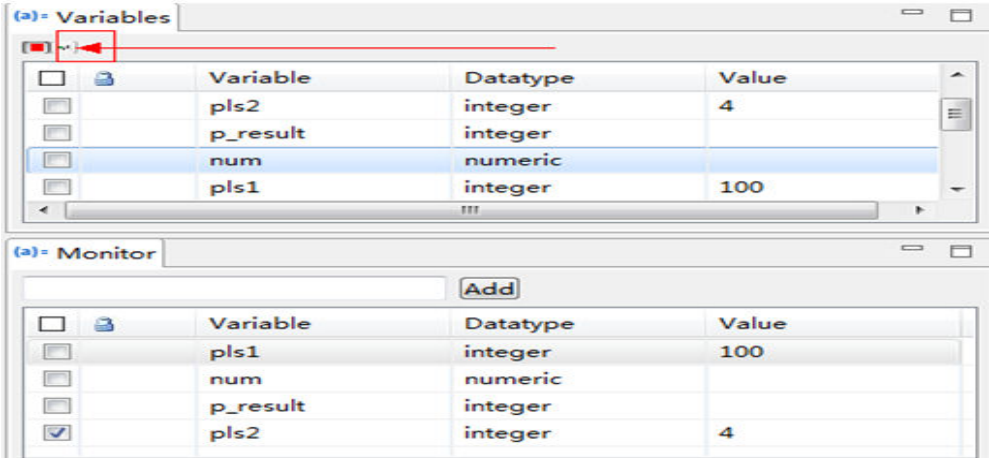
Once the specific connection is removed, all the parameter values in cache are cleared.

Displaying Variable in Monitor Window

Data Studio displays the variables which are being monitored in the Monitor Window while debugging.

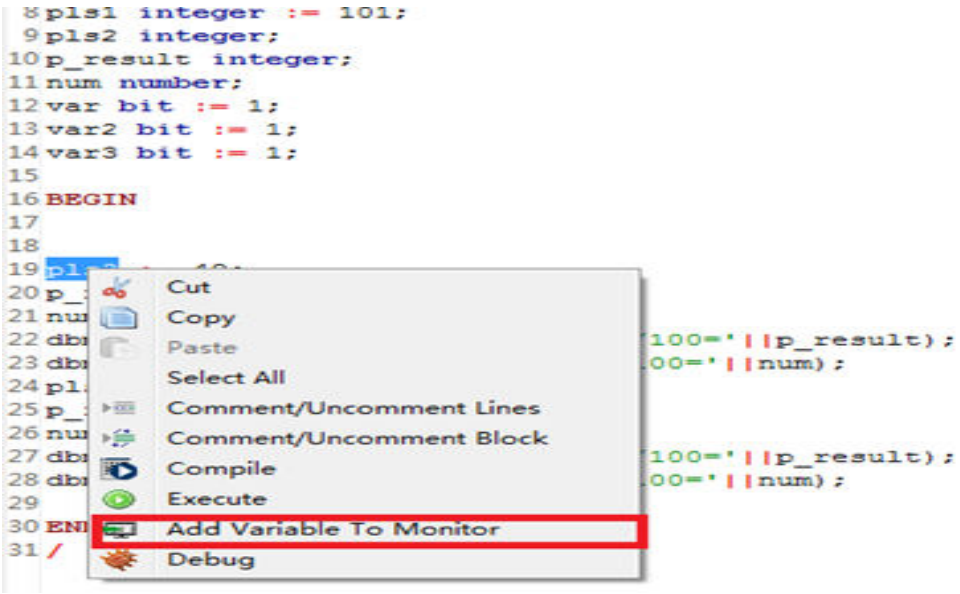
In the Monitor Window, variables must be added in following ways:

- Add selected variables from the **Variable** window and right-click the variable.
- Select variables from the **Variable** window and add variables by clicking the button in the **Variable** window toolbar.



If the value is changed in the variable window, the same would reflect in the monitor window if the variable is monitored and vice versa.

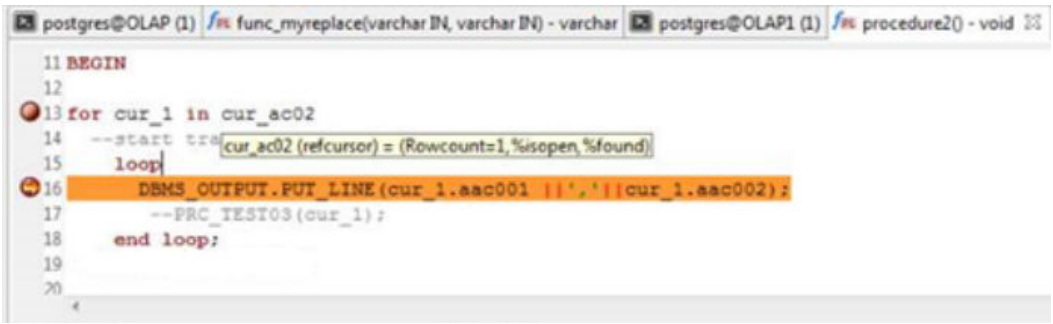
- During function/procedure debugging, right-click the variable to be added in the editor and add the variable to the **Monitor** window.



The **Monitor** window can be dragged to anywhere in the **Data Studio** window.

Displaying Cursor Information for Variables During Debugging

In Data Studio, variable information is displayed if the cursor is hovered over that variable during the debugging of PL/SQL functions.



Supporting Rollback/Commit During Debugging

Data Studio provides the option to commit/rollback the PL/SQL query execution result after debugging is finished.

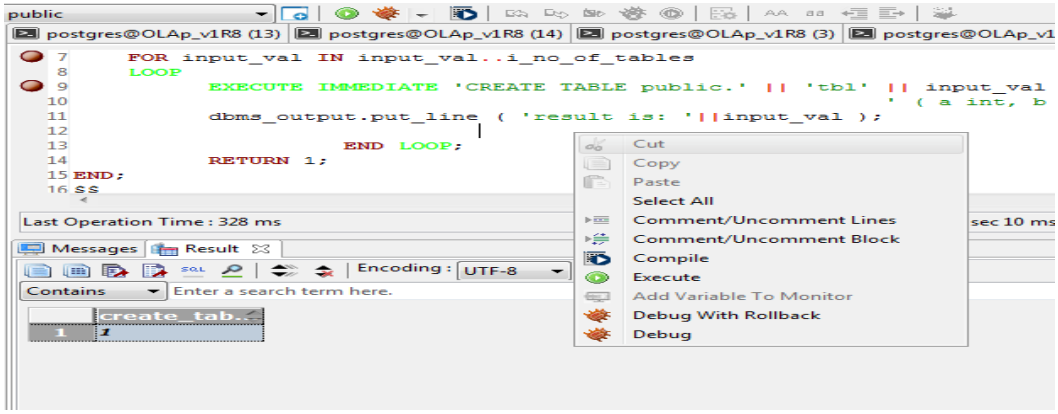
- If **Debug With Rollback** option is enabled, the PL/SQL execution result after debugging is not saved to the database.
- If **Debug With Rollback** option is disabled, the PL/SQL execution result after debugging is submitted to the database.

Perform the following steps to enable the rollback function:

Step 1 Check the **Debug With Rollback** box to enable the rollback function during PL/SQL debugging.

Or

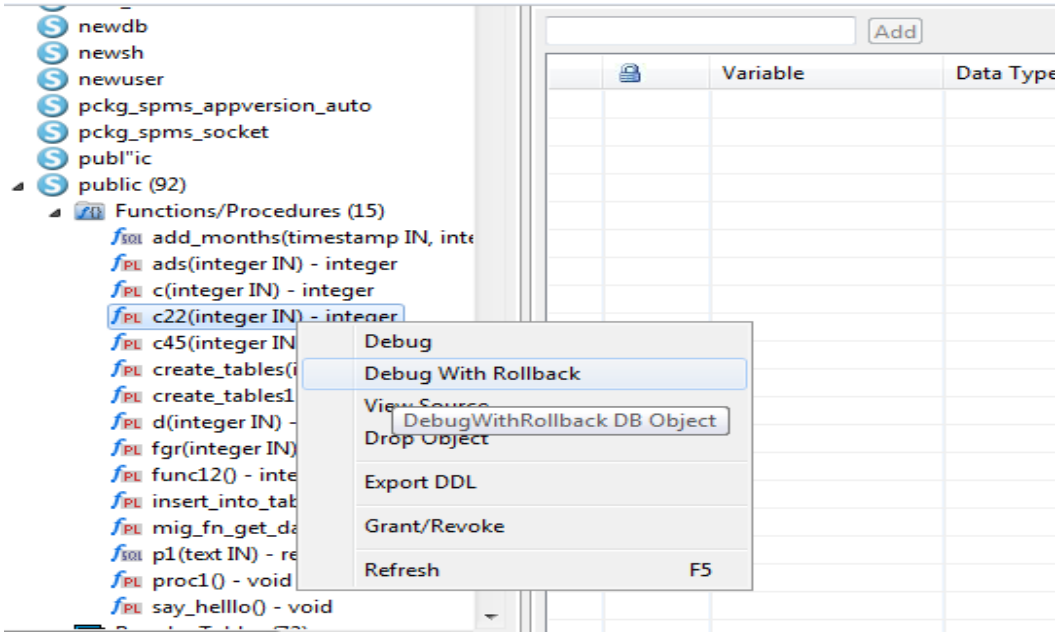
Right-click the **SQL Terminal** window where the PL/SQL function is executed.



Select **Debug With Rollback** to enable the rollback function after the debugging is complete.

Or

Right-click any PL/SQL function under **Functions/Procedure** in **Object Browser**.



----End

4.14.3 Controlling Execution

This section contains the following topics:

- **Starting Debugging**
- **Stepping Through a PL/SQL Function**
- **Continuing the Debug Execution**
- **Viewing Callstack**

Starting Debugging

Select the function that you want to debug in the **Object Browser** pane. Start debugging by clicking on the toolbar (or any other method as mentioned in the earlier sections). If no breakpoint is set, or the set breakpoint is invalid, the debugging operation will not halt at any statement and Data Studio will simply execute the object and display the results (if any).

Stepping Through a PL/SQL Function

You can step through the debugging execution using the debug step commands from the toolbar. Step controls are used to step through the execution of the program line by line. If a breakpoint is encountered while performing a step operation, the execution will suspend at the breakpoint and the step operation is ended.

Stepping is the process of running one statement at a time. After stepping through a statement, 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 a new tab beyond 100 is opened, the tab of the calling function is closed. For example, if 100 tabs are already opened and if one of the debug objects calls a new debug object (other than already opened 100 tabs), then Data Studio will close the calling function, and open the new debug object.

Step Into

To step through code one statement at a time, select **Step Into** from the **Debug** menu, or click , or press **F7**.

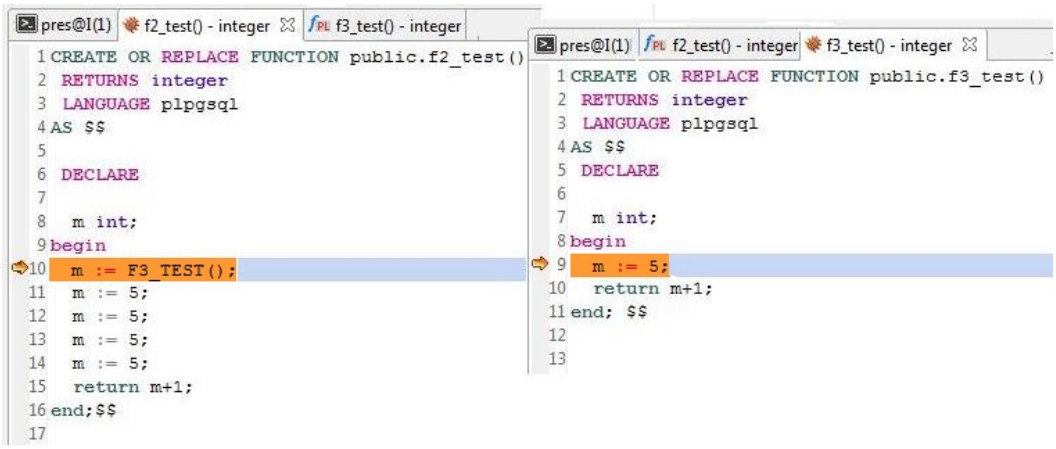
When stepping into a function, Data Studio executes the current statement and then enters the break mode. The debug position will be indicated by an arrow on the left ruler pane. If the executed statement calls another function, Data Studio will step into that function. Once you have stepped through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

To go into the next statement, click the **Step Into** button or press **F7** again. If you click the **Continue** button, PL/SQL code execution will continue.

An example is as follows:

When entering line 8, enter **m := F3_TEST();**. That is, go to the line 9 in **f3_test()**. You can step through all the statements in **f3_test()** by stepping into each line by clicking the **Step Into** button or pressing **F7** repeatedly. Once you have stepped through all the statements in that function, Data Studio jumps to *Line 10* in *f2_test()*.

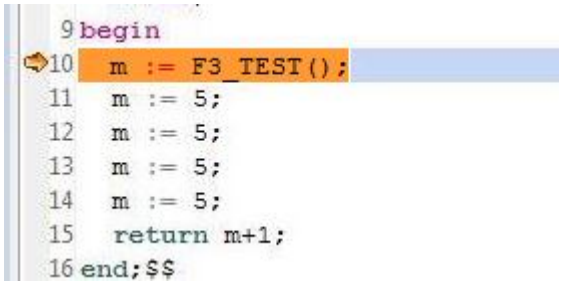
The currently debugging object is marked with the symbol in the tab title with the function name.



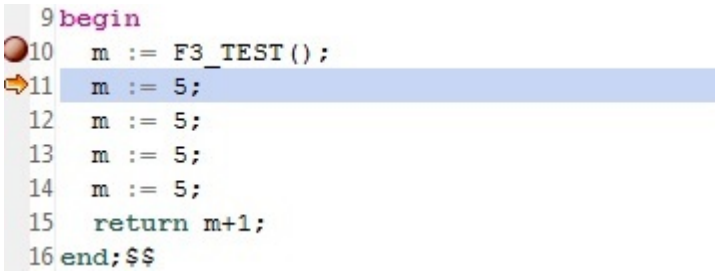
Step Over

Stepping over is the same as Stepping into, except that when it reaches a call for another function, it will not step into the function. The function will run, and you will be brought to the next statement in the current function. **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.

In the below example, when you click **Step Over** in *Line 10*, Data Studio runs the *f3_test()* function.



The cursor will be moved to the next statement in *f2_test()*, that is, *Line 11* in *f2_test()*.



You can step over a function when you are familiar with the way the function works and are sure that its execution will not affect the issue that you are investigating.

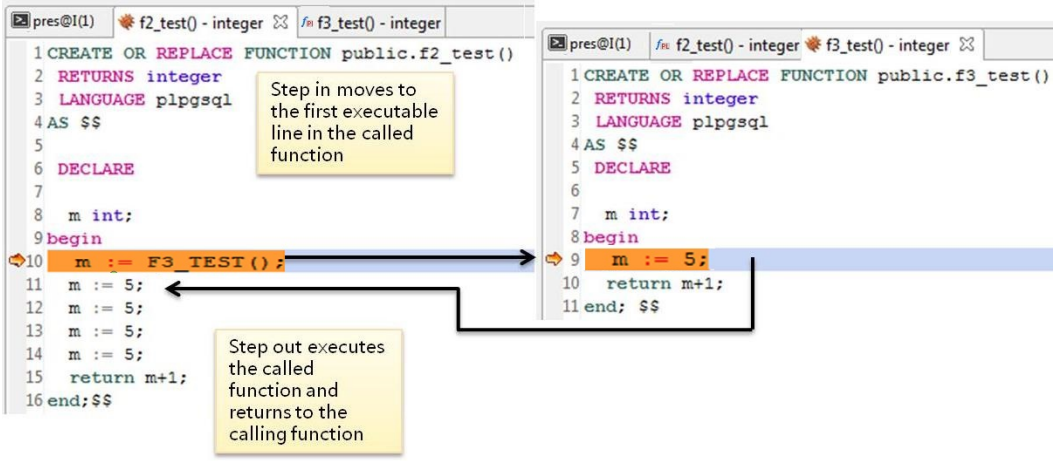
NOTE

Stepping over a line of code that does not contain a function call executes the line just like stepping into the line.

Step Out

Stepping out of a sub-program continues execution of the function and then suspends execution after the function returns to its calling function. You can step out of a long function when you have determined that the rest of the function is not significant 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 calling function.

Both stepping over and stepping out of a function will execute a function. The shortcut key for the step out operation is **Shift**+**F7**.



In the preceding example,

- Choose **Debug > Step Into** to step into *f3_test()*.
- Choose **Debug > Step Out** to step out of *f3_test()*

Continuing the Debug Execution

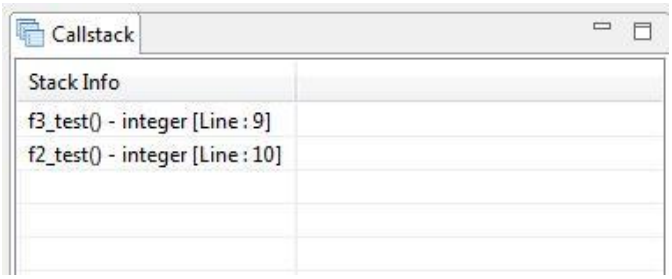
When the debugging process stops at a specific location, you can select **Continue** (**F9**) from the **Debug Menu** or click button from the toolbar to continue the PL/SQL function execution.

Viewing Callstack

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

You can navigate among multiple functions through the **Callstack** pane by double-clicking the function name in the **Callstack** pane. For example, when *f2_test()* calls *f3_test()* at *Line 10*, the debug pointer will point to the first valid executable line (which is *Line 9*, in the above example) in the called function.

In this case, the **Callstack** pane will be as shown below:



NOTE

The content of the **Callstack** pane can be copied to the clipboard using **Alt+J**.

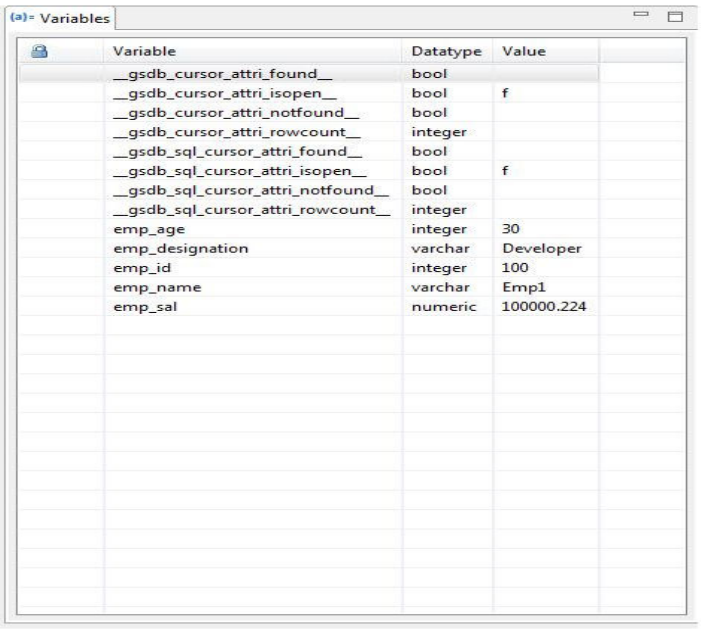
4.14.4 Checking Debugging Information

When you use Data Studio, you can examine debugging information through several debugging tabs. This section describes how to check the debugging information:

- **Operating on Variables**
- **Viewing Results**

Operating on Variables

The **Variables** pane is used to monitor information or evaluate values. The **Variables** pane can be opened from the minimized window panel. Using this pane, you can evaluate or modify variables or arguments in a PL/SQL function. As you step through the code, the values of some local variables may change.



NOTE

The content of the **Variables** pane can be copied to the clipboard using **Alt**+**K**.

You can double-click the corresponding row of the variable and manually change variable values during run-time.

Click the **Variable**, **Datatype**, or **Value** column in the **Variables** pane to sort the values. For example, to change the value of the percentage variable from 5 to 15, double-click the corresponding row in the **Variable** pane. The **Set Variable Value** dialog box will be displayed, which prompts you to input the variable value. Input the variable value and click **OK**.

To set **NULL** as a variable value, enter **NULL** or **null** in the **Value** column.

If the variable is read-only, it will be indicated by beside the corresponding variable.

Users cannot update these variables. A variable declared as a constant will not be shown as read-only in the **Variables** pane. However, while updating it, an error will occur.

NOTE

- In the **Variables** pane, the parameter value will be displayed as **NULL**, if the input to the parameter value is string literal 'NULL'.
- When the value is set to a variable using Data Studio, then the value of the variable is the same as the value returned by the statement "select expression" executed from **gsql**.

Setting/Displaying Variables	Description
Setting NULL Values	1. Double-click a variable value in the **Variables** pane. A dialog box is displayed. 2. Set the variable to an empty value.
Configuring String Values	Configure the string values as follows: • To configure **abc**, enter **abc**. • To configure **Master's Degree**, enter **Master's Degree**. • To set variable as text (**NULL**), configure **NULL** in the **Variables** pane.
Setting Boolean Values	Enclose the boolean values *t* or *f* within single quotes. To set *t* to a boolean variable, enter *'t'* in the **Variables** pane.
Displaying Variable Value	If the variable value is **NULL** text, it will be displayed as **NULL**. If the variable value is **NULL**, it will be displayed as empty. If the variable value is a string, for example, *abc*, it will be displayed as *abc*.

Viewing Results

The **Result** tab displays the output for the PL/SQL debugging session, with the corresponding function/procedure name at the top of the tab. The **Result** tab will appear automatically, only if there is a result for the executed PL/SQL program.

You can copy the content of the **Result** tab, by clicking  . For details, see **Working with SQL Terminals**.

NOTE

- The tool tip in the **Result** tab displays a maximum of 10 lines, where each line contains maximum of 80 characters.
- If the result of the executed query is NULL, it will be displayed as **<NULL>**.
- Tab characters (ASCII 009) in table data will not be displayed in the **Results/View Table Data/Properties** window. Tab characters will be included correctly when copying/exporting the data. Tool tip will also display the tab characters correctly.

4.15 Working with Functions/Procedures

4.15.1 Overview

This section provides you with details on working with functions/procedures and SQL functions in Data Studio.

NOTE

Data Studio supports PL/pgSQL and SQL languages for the operations are listed as follows:

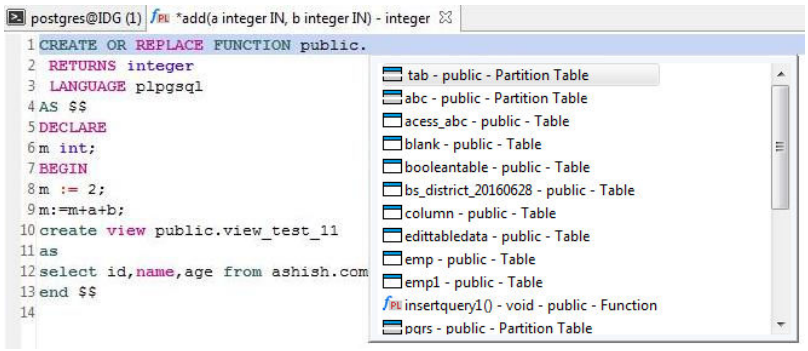
- **Creating a Function/Procedure**
- **Editing a Function/Procedure**
- **Exporting a Function/Procedure DDL**
- **Dropping a Function/Procedure**

4.15.2 Selecting a DB Object in the PL/SQL Viewer

Data Studio suggests a list of possible schema names, table names, column names, views, sequences, and functions in the **PL/SQL Viewer**.

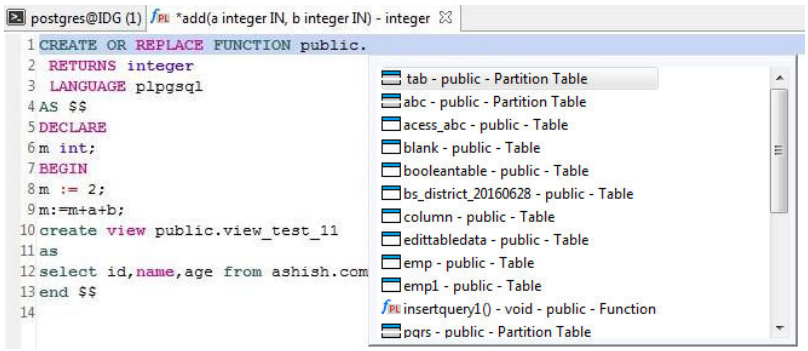
Follow the steps below to select a DB object:

Step 1 Press **Ctrl**+**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 **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 Enter **.** (period) 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 accessed.

  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>
```

- Auto-suggest may show "Loading" in Terminal for following scenarios:
  - The object is not loaded due to the value mentioned in the **Load Limit** field. Refer to **Adding a Connection** for more information.
  - The object is not loaded since it is added in the **Exclude** list option.
  - 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 that are named **public** and **PUBLIC**, then all child objects for both these schemas will be displayed.

----End

4.15.3 Exporting a Function/Procedure DDL

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

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

The **Data Studio Security Disclaimer** dialog box is displayed.

Step 2 Click **OK**.

The **Save As** dialog box is displayed.

Step 3 In the **Save As** dialog box, select the location to save the DDL and click **Save**. The status bar displays the progress of the operation.

NOTE

- To cancel the export operation, double-click the status to open the **Progress View** tab and click . For details, see **Canceling the Table Data Export Operation**.
- The exported file name will not be the same as function/procedure name, if the function/procedure name contains characters which are not supported by Windows.
- MS Visual C runtime file (msvcrt100.dll) is required to complete this operation. For details, see **Troubleshooting**.
- Multiple objects can be selected to export DDL. Refer to **Batch Export** section for list of objects not supported for export DDL operation.

The **Export** message and status bar displays the status of the completed operation.

Database Encoding	File Encoding	Support for DDL Export
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

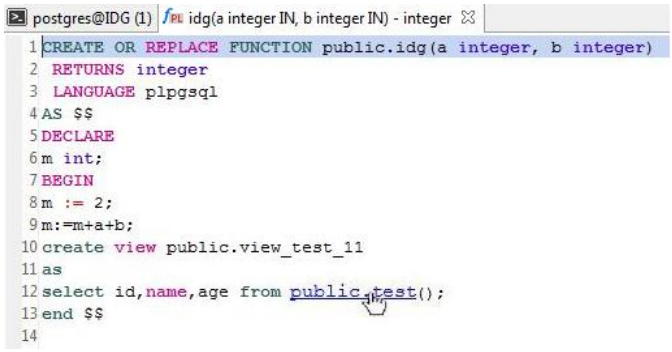
----End

4.15.4 Viewing Object Properties in the PL/SQL Viewer

Data Studio allows you to view table properties, procedures/functions and SQL functions.

Follow the steps below to view table properties:

Step 1 Press **Ctrl** and point to the table name.



Step 2 Click the highlighted table name.

The properties of the selected table is displayed.

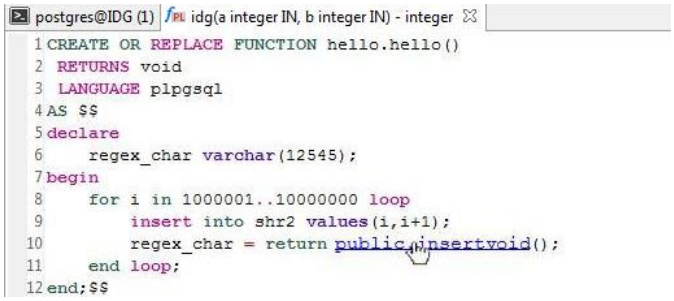
NOTE

The table properties are read-only.

----End

Follow the steps below to view functions/procedures or SQL functions:

Step 1 Press **Ctrl** and point to the procedure/function name or SQL function name.

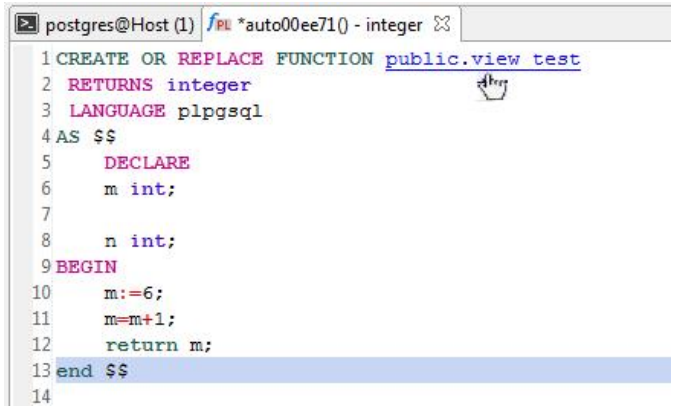


Step 2 Click the highlighted function/procedure name or SQL function name. The function/procedure or SQL function is displayed in a new **PL/SQL Viewer** tab based on your selection.

----End

Follow the steps below to view object DDL:

Step 1 Press **Ctrl** and point to the name of an object DDL to be viewed.



Step 2 Click the highlighted Object DDL name. A new tab page for viewing the object DDL is displayed based on your selection.

----End

4.15.5 Dropping a Function/Procedure

Individual or batch drop can be performed on functions/procedures. Refer to **Dropping a Batch of Objects** section for batch drop.

Follow the steps below to drop a function/procedure or SQL function object:

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

Step 2 To drop objects in batches, right-click two or more selected function/procedure objects and choose **Drop Objects**.

Step 3 In the confirmation dialog box, click **Yes** to complete the operation successfully.

The status bar displays the status of the completed operation.

----End

4.15.6 Executing a Function/Procedure

After you connect to the database, all the stored functions/procedures and tables will be automatically populated in the **Object Browser** pane. You can use Data Studio to execute PL/SQL programs or SQL functions.

NOTE

- Blank lines occurring above or below in a function/procedure will be trimmed by Data Studio before being sent to the server. Blank lines will also be trimmed when displaying the source received from the server.
- To execute a function/procedure, enter the same values in Data Studio and the gsql client. If you do not enter any value in Data Studio, then NULL is considered as the input.

  For example:

  - To execute the function/procedure with string, enter the value as data.

  - To execute the function/procedure with date, enter the value as **to_date('2012-10-10', 'YYYY-MM-DD')**.
- A function/procedure with OUT and INOUT parameter types cannot be executed directly.
- Data Studio will not execute any function/procedure with unknown data type parameters.

You can right-click the function/procedure in the **Object Browser** to perform the following operations:

- Refresh the program to get the latest program from the server.
- Execute the function/procedure or SQL function.
- Debug a PL/SQL function.
- Drop the debug object.

Executing a PL/SQL Program or SQL Function

Follow the steps below to execute a PL/SQL program or SQL function:

Step 1 Double-click and open the PL/SQL program or SQL function. Each debug object will be opened in a new tab. You can open a maximum of 100 tabs in Data Studio.

Step 2 Click on the toolbar or choose **Run** > **Execute** from the main menu,

or right-click the program name in the **Object Browser** and select **Execute**.

Step 3 The **Execute Function/Procedure** dialog box is displayed prompting for your input.

NOTE

If there is no input parameter, then the **Execute Function/Procedure** dialog box will not appear. Instead, the PL/SQL program will execute and the result (if any) will be displayed in the **Result** tab.

Step 4 Provide your input for the function/procedure in the **Execute PL/pgSQL** dialog box and click **OK**.

To set NULL as the parameter value, enter **NULL** or **null**.

- If you do not provide a value that starts with a single quote, then a single quote (') will be added by Data Studio before and after the value and typecasting is done.
- If you provide a value that starts with a single quote, an additional single quote will not be added by Data Studio, but data type typecasting is done. For example:

  For supported data types, the execution queries are as follows:

  ```
  select func('1'::INTEGER);
  select func('1'::FLOAT);
  select func('xyz'::VARCHAR);
  ```

- If quotes are already provided, you need to take care of escaping the quotes.

  Example: If the input value is **ab'c**, then you need to enter **ab''c**.

  The PL/SQL program is executed in the **SQL Terminal** tab and the result is displayed in the **Result** tab. You can copy the contents of the Result tab by clicking . Refer to **Working with SQL Terminals** for more information on toolbar options..

  Refer to the **Execute SQL Queries** section for information on reconnect option in case connection is lost during execution.

----End

4.15.7 Granting/Revoking a Privilege

Follow the steps below to grant/revoke a privilege:

Step 1 Right-click selected function/procedure and select the **Grant/Revoke**.

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

Step 2 Refer to **Granting/Revoking a Permission (Function/Procedure)** section to grant/revoke a privilege.

----End

4.16 GaussDB(DWS) Tables

4.16.1 Table Management Overview

This section describes how to manage tables efficiently.

NOTE

- You need to configure all mandatory parameters to complete the operation. Mandatory parameters are marked with an asterisk (*).

4.16.2 Creating Regular Table

4.16.2.1 Overview

This section describes how to create a common 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 to create a common table:

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.

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 4-8 Parameters

Parameter	Row-store Table	Column-store Table	ORC Table
Table Type	✓	✓	✗
If Not Exists	✓	✓	✓

Parameter	Row-store Table	Column-store Table	ORC Table
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.

You can perform the following operations only in a common table:

- **Deleting a Column**
- **Editing a Column**
- **Moving a Column**

To define a column, perform the following steps:

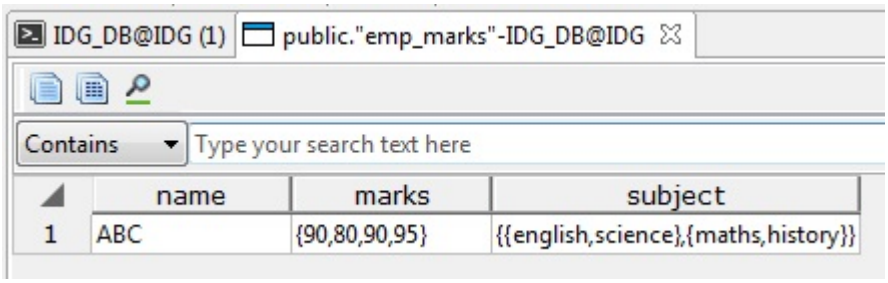
Step 1 Enter the column name in **Column Name** field. It specifies the name of a column to be created in the new table. This must be a unique name in the table.

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**.

----End

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

Table 4-9 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

Deleting a Column

To delete a column, perform the following steps:

Step 1 Select a column.

Step 2 Click **Delete**.

----End

Editing a Column

Follow the steps to edit a column:

Step 1 Select a column.

Step 2 Click **Edit**.

Step 3 Edit the column details as required and click **Update** to save changes.

NOTE

You must complete the edit operation and save the changes to continue with other operations.

----End

Moving a Column

You can move a column in a table. To move a column, select the column and click **Up** or **Down**.

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 4-10 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.

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

- **Primary Key**
- **UNIQUE**
- **CHECK**

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.

Fill Factor: For details, see **Primary Key**.

DEFERRABLE: For details, see **Primary Key**.

INITIALLY DEFERRED: For details, see **Primary Key**.

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

Mandatory parameters are marked with asterisks (*).

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 (*). After defining all constraints, click **Next**.

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

Table 4-11 Constraint types

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

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 4-12 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.

4.16.2.2 Working with Columns

After creating a table, you can add new columns in that table. You can also perform the following operations on the existing column only for a Regular table:

- **Creating a New Column**
- **Renaming a Column**
- **Toggle Not Null**
- **Dropping a Column**
- **Setting the Default Value of a Column**
- **Changing the Data Type**

Creating a New Column

Follow the steps below to add a new column to the existing table:

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.

Step 2 Enter the details and click **Add**. You can view the added column in the corresponding table.

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

----End

Renaming a Column

Follow the steps below to rename a column:

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

A **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

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**.

A **Toggle Not Null Property** dialog box is displayed prompting you to set or reset the Not Null option.

Step 2 In the confirmation dialog box, click **OK** to complete the operation successfully. Data Studio displays the status of the operation in the status bar.

----End

Dropping a Column

Follow the steps below to drop 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

Setting the Default Value of a Column

Follow the steps below to set the default value for 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

Follow the steps below to change the data type of a column:

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 show as Unknown while modifying 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

4.16.2.3 Working with Constraints

You can perform the following operations after a table is created only for a Regular table:

- **Creating a Constraint**
- **Renaming a Constraint**
- **Dropping a Constraint**

Creating a Constraint

Follow the steps below to add a new constraint to the existing table:

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.

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

4.16.2.4 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. You can perform the following operations only in a common table:

- **Creating an Index**
- **Renaming an Index**
- **Changing a Fill Factor**
- **Deleting an Index**

Creating an Index

Perform the following steps to add an index to a table:

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

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

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

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

4.16.3 Creating Foreign Table

Foreign tables created using query execution in SQL Terminal or any other tool can be viewed in the Object browser after refresh.

Step 1 To view the newly created foreign table, right-click and select **Refresh** either at database, schema and foreign table group level.

NOTE

- GDS Foreign table is denoted with  icon before the table name.
- HDFS Foreign table is denoted with  icon before the table name.
- HDFS Foreign table with partition is denoted with  icon before the table name.

----End

4.16.4 Creating Partition Table

4.16.4.1 Overview

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 **Change Order of Partition**.

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 **Checking the 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

Provide the following information to create a table:

For details, see **Providing Basic Information**.

- Table Name
- Schema
- Options
- Description of Table

Perform the following steps to configure other parameters:

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.

Step 2 Enter the ORC version number in the **ORC Version** field. This is applicable only for HDFS Partitioned table.

Step 3 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 4-13 Parameters

Parameter	Row Partition	Column Partition	ORC Partition
Table Type	x	x	x
If Not Exists	√	√	√
With OIDS	x	x	x

Parameter	Row Partition	Column Partition	ORC Partition
Fill Factor	√	x	x

----End

Defining a Column

For details, see **Defining a Column**.

The following table describes the parameters of partitioned tables.

Table 4-14 Parameters

Field	Row Partition	Column Partition	ORC Partition
Array Dimensions	√	x	x
Data Type	√	x	x
NOT NULL	√	√	√
Default	√	√	√
UNIQUE	√	x	x
CHECK	√	x	x

Change Order of Partition

You can change the order of partition as required in the table. To change the order, select the required partition and click **Up** or **Down**.

Checking the SQL Preview

For details, see **SQL Preview**.

Editing a Partition

Perform the following steps to edit a partition:

Step 1 Select a partition.

Step 2 Click **Edit**.

Step 3 Edit partition configurations as needed and click **Update** to save the changes.

NOTE

You must complete the edit operation and save the changes to continue with other operations.

----End

Deleting a Partition

Perform the following steps to delete a partition:

Step 1 Select a partition.

Step 2 Click **Delete**.

----End

Defining a Partition

The following table describes the parameters of partitioned tables.

Table 4-15 Parameters

Parameter	Row Partition	Column Partition	ORC Partition
Partition Type	By Range	By Range	By Value
Partition Name	√	√	x
Partition Value	√	√	x

Perform the following steps to define a table partition:

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 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  next to **Partition Value**.

1. Enter the value by which you want to partition the table in **Value** column.
2. Click **OK**.

Step 5 After you enter all information for partition, click **Add**.

Step 6 After defining all partitions, click **Next**.

----End

You can perform the following operations on the partitions of a row-or column-partitioned table, but not on ORC partitioned tables:

Deleting a Partition

Editing a Partition

Defining an Index

For details about index definitions, see **Defining an Index**.

Table 4-16 Parameters

Parameter	Row Partition	Column Partition	ORC Partition
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

Defining Table Constraints

For details about how to define table constraints, see **Defining Table Constraints**.

Table 4-17 Parameters

Parameter	Partition	Column Partition	ORC Partition
Check	√	x	x
Unique	√	x	x
Primary Key	√	x	x

Configuring Data Distribution

For details about how to select a distribution type, see **Selecting Data Distribution**.

Table 4-18 Parameters

Parameter	Row Partition	Column Partition	ORC Partition
DEFAULT DISTRIBUTION	√	√	x
Hash	√	√	√
Replication	√	√	x

4.16.4.2 Working with Partitions

After creating a table, you can add/modify partitions. You can also perform the following operations on an existing partition:

Renaming a Partition

Dropping a Partition

Renaming a Partition

Follow the steps below to rename 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

Dropping a Partition

Follow the steps below to drop 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 dropped from the table. Data Studio displays the status of the operation in the status bar.

----End

4.16.5 Grant/Revoke Privilege - Regular/Partition Table

Follow the steps below to grant/revoke a privilege:

Step 1 Right-click regular tables group and select the **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.

Step 4 Select **Grant/Revoke** in the **Privilege Selection** tab.

Step 5 Select/unselect the required privileges in the **Privilege Selection** tab.

In the **SQL Preview** tab, you can view the SQL query automatically generated for the inputs provided.

Step 6 Click **Finish**.

----End

4.16.6 Managing Table

4.16.6.1 Overview

This section describes how to manage tables efficiently.

NOTE

- You need to configure all mandatory parameters to complete the operation. Mandatory parameters are marked with asterisks (*).
- Refreshing is the only operation supported for foreign table.

After creating the table, you can perform operations on the existing table. Right-click the selected table and select the required operation.

Context Menu

Additional options for table operations are available in the table context menu. The context menu options available for table operations are:

Table 4-19 Table context menu options

Menu Item	Description
View Table Data	Opens the table data information. For details, see **Viewing Table Data**.
Edit Table Data	Opens the window for editing table data. For details, see **Editing Table Data**.
Reindex Table	Re-creates the table index. For details, see **Reindexing a Table**.
Analyze Table	Analyzes a table. For details, see **Analyzing a Table**.
Truncate Table	Truncates table data. For details, see **Truncating a Table**.
Vacuum Table	Vacuums table data. For details, see **Vacuuming a Table**.

Menu Item	Description
Set Table Description	Sets table description. For details, see **Setting the Table Description**.
Set Schema	Sets the schema of a table. For details, see **Setting the Schema**.
Export Table Data	Exports table data. For details, see **Exporting Table Data**.
Import Table Data	Imports table data. For details, see **Importing Table Data**.
Show DDL	Shows the DDL of a table. For details, see **Show DDL**.
Export DDL	Exports Table DDL. For details, see **Exporting Table DDL**.
Export DDL and Data	Exports DDL and table data. For details, see **Exporting Table DDL and Data**.
Rename Database	Renames a table. For details, see **Renaming a Table**.
Drop Table	Drops (deletes) a table. For details, see **Dropping a Table**.
Properties	Shows table properties. For details, see **Viewing Table Properties**.
Grant/Revoke	Grants or revokes permissions. For details, see **Grant/Revoke Privilege**.
Refresh	Refreshes a table.

4.16.6.2 Renaming a Table

Follow the steps below to rename a table:

Step 1 Right-click the selected table and select **Rename Table**.

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

Step 2 Enter the table name and click **OK**. You can view the updated table name in **Object Browser**.

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

NOTE

This operation is not supported for Partition ORC tables.

----End

4.16.6.3 Truncating a Table

Follow the steps below to truncate a table:

Step 1 Right-click the selected table and select **Truncate Table**. This operation deletes the data from an existing table.

Data Studio prompts you to confirm this operation.

Step 2 In the confirmation dialog box, click **OK** to complete the operation successfully.

A popup message and status bar display the status of the completed operation.

----End

4.16.6.4 Reindexing a Table

Index facilitate 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 must never happen, in practice, indexes can become corrupted due to software bugs or hardware failures. Reindexing provides a recovery method.
- The index has become "bloated". That is, it contains many empty or nearly-empty pages. This can occur with B-tree indexes in PostgreSQL under certain uncommon access patterns. Reindexing provides a way to reduce the space consumption of the index by writing a new version of the index without the dead pages.
- 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:

Step 1 Right-click the selected table and select **Reindex Table**.

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

NOTE

This operation is not supported for Partition ORC tables.

----End

4.16.6.5 Analyzing a Table

The analyzing table operation collects statistics about tables and table indicies 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.

Follow the steps below to analyze a table:

Step 1 Right-click the selected table and select **Analyze Table**.

The **Analyze Table** message and status bar displays the status of the completed operation.

----End

4.16.6.6 Vacuuming a Table

Vacuuming table operation reclaims space and makes it available for re-use.

Follow the steps below to vacuum the table:

Step 1 Right-click the selected table and select **Vacuum Table**.

The **Vacuum Table** message and status bar display the status of the completed operation.

----End

4.16.6.7 Setting the Table Description

Follow the steps below to set the description of a table:

Step 1 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.

Step 2 Enter the description and click **OK**.

The status bar displays the status of the completed operation.

Step 3 To view the table description, right-click the selected table and select **Properties**.

----End

4.16.6.8 Setting the Schema

Follow the steps below to set a schema:

Step 1 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.

Step 2 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.

----End

4.16.6.9 Dropping a Table

Individual or batch dropping can be performed on tables. Refer to **Dropping a Batch of Objects** section for batch dropping.

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.

Follow the steps below to drop the table:

Step 1 Right-click the selected table and select **Drop Table**.

Data Studio prompts you to confirm this operation.

Step 2 In the confirmation dialog box, click **OK** to complete the operation successfully.

The status bar displays the status of the completed operation.

----End

4.16.6.10 Viewing Table Properties

Follow the steps below to view the properties of a table:

Step 1 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.

Tab Name	Operations Allowed
General	Save, Cancel, and Copy **NOTE** Only Table Description field can be modified.
Columns	Add, Delete, Save, Cancel, and Copy
Constraints	Add, Delete, Save, Cancel, and Copy
Index	Add, Delete, Save, Cancel, and Copy

Refer to **Editing Table Data** section for more information on edit, save, cancel, copy, paste, refresh operations.

NOTICE

When viewing table data, Data Studio automatically adjusts the column widths for table view. Users 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.

NOTE

- Individual property window is displayed for each table.
- If the property of a table is modified for the table that is already opened, then 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 a column is determined by the column with the maximum content length.
- Any change made to the table properties from Object Browser will be reflected after refreshing () the **Properties** tab.
- Pasting operation is not allowed in **Data Type** column.

----End

4.16.6.11 Grant/Revoke Privilege

Follow the steps below to grant/revoke a privilege:

Step 1 Right-click the selected regular/partitioned table and select **Grant/Revoke**.

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

Step 2 Refer to **Grant/Revoke Privilege - Regular/Partition Table** to grant/revoke privilege.

----End

4.16.7 Managing Table Data

4.16.7.1 Overview

This section describes how to manage table data.

4.16.7.2 Exporting Table DDL

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**.

The **Data Studio Security Disclaimer** dialog box is displayed.

Step 2 Click **OK**.

The **Save As** dialog box is displayed.

Step 3 In the **Save As** dialog box, select the location to save the DDL and click **Save**. The status bar displays the progress of the operation.

NOTE

- To cancel the export operation, double-click the status to open the **Progress View** tab and click . For details, see **Canceling the Table Data Export Operation**.
- 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 the status bar display the status of the completed operation.

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

NOTE

You can select multiple objects and export their DDL. **Batch Export** lists the objects whose DDL cannot be exported.

----End

4.16.7.3 Exporting Table DDL and Data

The exported table DDL and data include the following:

- DDL of the table.
- Columns and rows of the table.

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**.

The **Data Studio Security Disclaimer** dialog box is displayed.

Step 2 Click **OK**.

The **Save As** dialog box is displayed.

Step 3 In the **Save As** dialog box, select the location to save the DDL and click **Save**. The status bar displays the progress of the operation.

NOTE

- To cancel the export operation, double-click the status to open the **Progress View** tab and click . For details, see **Canceling the Table Data Export Operation**.
- 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.

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

NOTE

You can select multiple objects from regular and partitioned tables to export DDL and data, including columns, rows, indexes, constraints, and partitions. **Batch Export** lists the objects whose DDL and data cannot be exported.

----End

4.16.7.4 Exporting Table Data

Perform the following steps to export 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

    Column/Row Size in **FAQ** describes the row and column size supported by xlsx and xls.
- **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

Canceling the Table Data Export Operation

Perform the following steps to cancel the table data export operation:

Step 1 Double-click the status bar to open the **Progress View** tab.

Step 2 In the **Progress View** tab, click  .

Step 3 In the **Cancel Operation** dialog box, click **Yes**.

The **Messages** tab and status bar display the status of the canceled operation.

----End

4.16.7.5 Show DDL

To display a DDL query for a table, perform the following steps:

Step 1 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**.

Database Encoding	File Encoding	Show 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

4.16.7.6 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

Canceling the Table Data Import Operation

Perform the following steps to cancel the table data import operation:

Step 1 Double-click the status bar to open the **Progress View** tab.

Step 2 In the **Progress View** tab, click  .

Step 3 In the **Cancel Operation** dialog box, click **Yes**.

The **Messages** tab and status bar display the status of the canceled operation.

----End

4.16.7.7 Viewing Table Data

Follow the steps to view table data:

Step 1 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	Toolbar Icon	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 **Query Results** 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 Search field:

Icon Name	Icon	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. Users 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.
  - 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.

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** 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 **Query Results**.

----End

4.16.7.8 Editing Table Data

Follow the steps below to edit table data:

Step 1 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.

----End

Data Studio validates only the following data types entered into cells:

Bigint, bit, boolean, char, date, decimal, double, float, integer, numeric, real, smallint, time, time with time zone, time stamp, time stamp with time zone, tinyint, and varchar. Editing of array type data type is not supported.

Any related errors during this operation reported by database will be displayed in Data Studio. Time with time zone and timestamp with time zone columns are non-editable columns.

You can perform the following operations in the **Edit Table Data** tab:

- **Insert**
- **Delete**
- **Update**
- **Copy**
- **Paste**

Insert

Follow the steps to insert a row:

Step 1 Click  to insert a row.

Step 2 Double-click the cell to modify and enter the required details in the row.

Step 3 Click  to save changes.

The **Edit Table Data** tab status bar shows the **Query Submit Time**, **Number of Rows Fetched**, **Execution Time** and **Status** of the operation.

NOTICE

Data Studio updates rows identified by the unique key. If a unique key is not identified for a table and there are identical rows, then an update operation made on one of the rows will affect all identical rows. Refresh the **Edit Table Data** tab to view the updated rows.

NOTE

- Changes to cells in a row that are not saved are highlighted in green. Once saved the color resets to default color.
- Unsaved records are highlighted in red. The number of successful and failed records are displayed in the status bar of the **Edit Table Data** tab.
- Clicking **Save** either saves all the valid changes or does not save anything if there are invalid changes. Refer to **Editing Table Data** to set the behavior of save operation.

Step 4 Click  to roll back the changes that are not saved.

Step 5 Set the preference to define:

- Number of records to be fetched
- Column width
- Copy option from result set

  For details, see **Query Results**.

----End

Data Studio allows you to edit the distribution key column only for a new row.

Delete

Follow the steps to delete a row:

Step 1 Click the row header of the row to be deleted.

Step 2 Click  to delete a row.

Step 3 Click  to save changes. The **Define Unique Key** dialog box is displayed.

Step 4 Click the required option:

- **Use All Columns**

  Click **Use All Columns** to define all columns as unique key.

- **Custom Unique Key**

  a. Click **Custom Unique Key** to define selected columns as unique key.

  b. The **Define Unique Key** dialogue box is displayed.

  c. Select the required columns and click **OK**.

- Cancel

  Click **Cancel** to modify the information in **Edit Table Data** tab.

  The **Edit Table Data** tab status bar shows the **Query Submit Time**, **Number of Rows Fetched**, **Execution Time** and **Status** of the operation.

  Select **Remember the selection for this window** option to hide the unique definition window from displaying while continuing with the edit table data operation. Click  from Edit Table Data toolbar to clear previously selected unique key definition and display unique definition wind

NOTE

- Changes to cells in a row that are not saved are highlighted in green. Once saved the color resets to default color.
- Unsaved records are highlighted in red. The number of successful and failed records are displayed in the status bar of the **Edit Table Data** tab.
- Clicking **Save** either saves all the valid changes or does not save anything if there are invalid changes. For details, see **Editing Table Data**.

Step 5 Click  to roll back the changes that are not saved.

Step 6 Refresh the table data to view deleted duplicate rows.

----End

Update

Follow the steps to update cell data:

Step 1 Double-click the cell to update the contents of the cell.

Step 2 Click  to save changes.

The **Define Unique Key** dialog box is displayed.

Step 3 Click the required option:

- **Use All Columns**

  Click **Use All Columns** to define all columns as unique key.

- **Custom Unique Key**

  a. Click **Custom Unique Key** to define selected columns as unique key.

  b. The **Define Unique Key** dialogue box is displayed.

  c. Select the required columns and click **OK**.

- **Cancel**

  Click **Cancel** to modify the information in **Edit Table Data** tab.

  The status bar shows the **Execution Time** and **Status** of the operation.

  Select **Remember the selection for this window** option to hide the unique definition window from displaying while continuing with the edit table data operation. Click  from Edit Table Data toolbar to clear previously selected unique key definition and display unique definition wind

  NOTE

  - Changes to cells in a row that are not saved are highlighted in green. Once saved the color resets to default color.
  - Unsaved records are highlighted in red. The number of successful and failed records are displayed in the status bar of the **Edit Table Data** tab.
  - Clicking **Save** either saves all the valid changes or does not save anything if there are invalid changes. For details, see **Editing Table Data**.

Step 4 Click  to roll back the changes that are not saved.

Step 5 Refresh the table data to view deleted duplicate rows.

----End

During the edit operation, Data Studio does not allow you to edit the distribution key column as it is used by the DB to locate data in the database cluster.

Copy

You can copy data from the **Edit Table Data** tab.

Follow the steps to copy data:

Step 1 Select the cell(s) and click  (Copy) or  (Advanced Copy).

For more information about the differences between **Copy** and **Advanced Copy**, see **Execute SQL Queries**.

NOTE

- Data can be copied to include the row number and/or column header. Refer to **Query Results** to set this preference.
- Select part of cell content and press **Ctrl+C** or click  to copy selected text from a cell.

----End

Paste

You can copy data from a CSV file and paste it into cells in the **Edit Table Data** tab to insert and update records. If you paste onto existing cell data, the data is overwritten with the new data from the CSV file. Follow the steps to paste data into a cell:

Step 1 Copy data from the CSV file.

Step 2 Select the cell(s) and click  .

Step 3 Click  to save changes. The **Define Unique Key** dialog box is displayed.

Step 4 Click the required option:

- **Use All Columns**

  Click **Use All Columns** to define all columns as unique key.

- **Custom Unique Key**

  a. Click **Custom Unique Key** to define selected columns as unique key.

  b. The **Define Unique Key** dialogue box is displayed.

  c. Select the required columns and click **OK**.

- **Cancel**

  Click **Cancel** to modify the information in **Edit Table Data** tab.

  The status bar shows the **Execution Time** and **Status** of the operation.

  Select **Remember the selection for this window** option to hide the unique definition window from displaying while continuing with the edit table data operation. Click  from Edit Table Data toolbar to clear previously selected unique key definition and display unique definition wind

  NOTE

  - The number of copied cells from CSV must match the number of cells selected in the **Edit Table Data** tab to paste the data.
  - Click  to roll back the changes that are not saved.
  - Changes to cells in a row that are not saved are highlighted in green. Once saved the color resets to default color.
  - Unsaved records are highlighted in red. The number of successful and failed records are displayed in the status bar of the **Edit Table Data** tab.
  - Clicking **Save** either saves all the valid changes or does not save anything if there are invalid changes. For details, see **Editing Table Data**.

----End

During the pasting operation, Data Studio does not allow you to edit the distribution key column as it is used by the DB to locate data in the database cluster.

NOTE

Empty cells are shown as [NULL]. Empty cell in **Edit Table Data** tab can be searched using the **Null Values** search drop-down.

Refer to **Execute SQL Queries** for information on show/hide search bar, sort, column reorder, and encoding options..

4.16.8 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 the SQL Terminal to edit temporary tables. Refer to **Managing SQL Terminal Connections** for information about enabling SQL Terminal Connection reuse.

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 from the **Result** tab. Refer to the **Execute SQL Queries** for information on editing the resultset.

----End

4.17 Sequences

4.17.1 Creating Sequence

Follow the steps below to create 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 sequence name entered 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 minimum and maximum value should be between -9223372036854775808 and 9223372036854775807.
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

In the **SQL Preview** tab, you can view the SQL query automatically generated for the inputs provided.

----End

4.17.2 Grant/Revoke Privilege

Follow the steps below to grant/revoke a privilege:

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

The **Grant/Revoke** dialog 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 **Role** drop-down list in the **Privilege Selection** tab.

Step 4 Select **Grant/Revoke** in the **Privilege Selection** tab.

Step 5 Select/unselect the required privileges in the **Privilege Selection** tab.

In the **SQL Preview** tab, you can view the SQL query automatically generated for the inputs provided.

Step 6 Click **Finish**.

----End

4.17.3 Working with Sequences

You can perform the following operations on an existing sequence:

- Granting/Revoking a Privilege
- Dropping a Sequence
- Dropping a Sequence Cascade

Dropping a Sequence

Individual or batch dropping can be performed on sequences. Refer to **Dropping a Batch of Objects** section for batch drop.

Follow the steps to dropping a sequence:

Step 1 Right-click the selected sequence and select **Drop Sequence**.

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

Step 2 Click **Yes** to drop the sequence.

The status bar displays the status of the completed operation.

----End

Dropping a Sequence Cascade

Follow the steps to drop a sequence cascade:

Step 1 Right-click the selected sequence and select **Drop Sequence Cascade**.

The **Drop Sequence Cascade** dialog box is displayed.

Step 2 Click **Yes** to drop the sequence cascade.

The status bar displays the status of the completed operation.

----End

NOTE

This is only available for OLAP, not for OLTP.

Granting/Revoking a Privilege

Follow the steps to grant/revoke a privilege:

Step 1 Right-click selected sequence and select **Grant/Revoke**.

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

Step 2 Refer to **Grant/Revoke Privilege** to grant/revoke a privilege.

----End

4.18 Views

4.18.1 Creating a View

Follow the steps below to create a new view:

Step 1 Right-click the **Views** and select **Create View**.

The DDL template for the view is displayed in the SQL Terminal tab.

Step 2 Edit the DDL as required.

Step 3 Click  to execute the DDL.

Step 4 Press **F5** to refresh the **Object Browser**.

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

NOTE

The status bar will not display message on completion of this operation.

----End

4.18.2 Granting/Revoking a Privilege

Follow the steps below to grant/revoke 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 the role from the **Role** drop-down list in the **Privilege Selection** tab.

Step 4 Select **Grant/Revoke** in the **Privilege Selection** tab.

Step 5 Select/unselect the required privileges in the **Privilege Selection** tab.

In the **SQL Preview** tab, you can view the SQL query automatically generated for the inputs provided.

Step 6 Click **Finish**.

----End

4.18.3 Working with Views

Views can be created to restrict access to specific rows or columns of a table. A view can be created from one or more tables and is determined by the query used to create the view.

You can perform the following operations on an existing view:

- **Exporting the View DDL**
- **Dropping a View**
- **Dropping a View Cascade**
- **Renaming a View**
- **Setting the Schema for a View**
- **Viewing the Show DDL**
- **Setting the Default Value for the View Column**

- **Viewing the Properties of a View**
- **Granting/Revoking a Privilege**

Exporting the View DDL

Follow the steps below to export view the DDL:

Step 1 Right-click the selected view and select **Export DDL**.

The **Data Studio Security Disclaimer** dialog box is displayed.

Step 2 Click **OK**.

The **Save As** dialog box is displayed.

Step 3 In the **Save As** dialog box, select the location to save the DDL and click **Save**. The status bar displays the progress of the operation.

NOTE

- To cancel the export operation, double-click the status to open the **Progress View** tab and click  .
- The exported file name will not be the same as the view name, if the view name contains characters which are not supported by Windows.
- Multiple objects can be selected to export the view DDL. Refer to **Batch Export** for the list of the objects that are not supported for exporting view DDL.

The **Export** message and status bar display the status of the completed operation.

Database Encoding	File Encoding	Supports Exporting 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

Dropping a View

Individual or batch dropping can be performed on views. Refer to **Dropping a Batch of Objects** for batch dropping.

Follow the steps below to drop the view:

Step 1 Right-click the selected view and select **Drop View**.

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

Step 2 Click **Yes** to drop the view.

The status bar displays the status of the completed operation.

----End

Dropping a View Cascade

Follow the steps below to drop a view and its dependent database objects:

Step 1 Right-click the selected view and select **Drop View Cascade**.

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

Step 2 Click **Yes** to drop the view and its dependent database objects.

The status bar displays the status of the completed operation.

----End

Renaming a View

Follow the steps below to rename 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

Setting the Schema for a View

Follow the steps below to set 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 DDL

Follow the steps below to view the DDL of the view:

Step 1 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.

----End

Setting the Default Value for the View Column

Follow the steps below to set the default value for a column in the view:

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

A dialog box with the current default value (if it is set) is displayed which prompts 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

Viewing the Properties of a View

Follow the steps below to view the properties of the View:

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

The properties (General and Columns) of the selected View is displayed in different tabs.

NOTE

If the property of a View is modified that is already opened, then refresh and open the properties of the View again to view the updated information on the same opened window.

----End

Granting/Revoking a Privilege

Follow the steps below to grant/revoke a privilege:

Step 1 Right-click the selected view and select **Grant/Revoke**.

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

Step 2 Refer to **Granting/Revoking a Privilege** to grant/revoke privilege.

----End

4.19 Users/Roles

4.19.1 Creating a User/Role

A database is used by many users, and the users are grouped for management convenience. A database role can be one 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 privileges 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.

4.19.2 Working with Users/Roles

You can perform the following operations on an existing user/role:

- **Dropping a User/Role**
- **Viewing/Editing User/Role Properties**
- **Viewing the User/Role DDL**

Dropping a User/Role

Follow the steps below to drop a user/role:

Step 1 Right-click the selected user/role and select **Drop User/Role**.

The **Drop User/Role** dialog box is displayed.

Step 2 Click **Yes** to drop the user/role.

The status bar displays the status of the completed operation.

----End

Viewing/Editing User/Role Properties

Follow the steps below to view the properties of a user/role:

Step 1 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.

Refer to **Editing Table Data** for information on edit, save, cancel, copy, and refresh operations.

----End

Viewing the User/Role DDL

Follow the steps below to view the DDL of a user/role:

Step 1 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.

----End

4.20 SQL Terminal

4.20.1 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  in the toolbar or press **Ctrl+T** to open a new **SQL Terminal** tab.

The **SQL Terminal** tab is displayed.

----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.

Perform the following steps to open a new **SQL Terminal** tab in another connection:

Step 1 Select the required connection from the connection list in the toolbar.

Step 2 In the **Object Browser** pane, right-click the desired connected database and choose **Open Terminal**, or click in the toolbar. The **SQL Terminal** tab is displayed.

The name format of the new **SQL Terminal** tab is as follows:

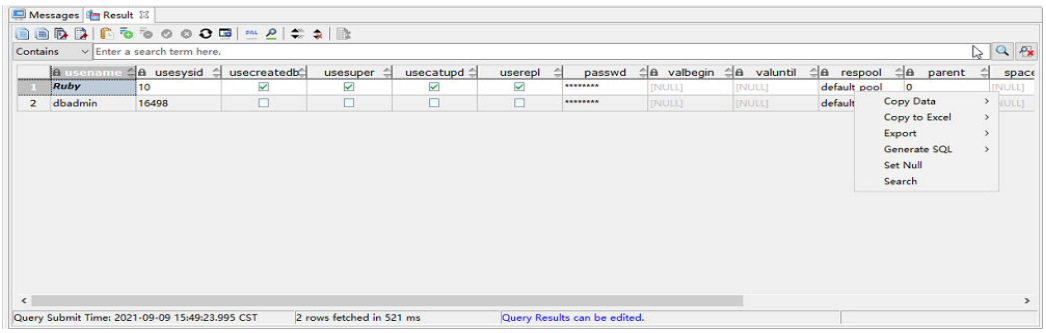
Database name@*Connection information*(*Tab number*), for example, *postgres*@*IDG_1*(*2*). The number of each **SQL Terminal** tab in the same connection information is unique.

----End

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**.

Step 3 Expand the **Result Management** tab and choose **Query Results**.

Step 4 In the **Result Advanced Copy** area, select **Include column header** and **Include row number**.

----End

The following table describes the right-click options.

Table 4-20 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.
Export	Current Page	Exports the table data on the current page.

Option	Sub-Item	Description
	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

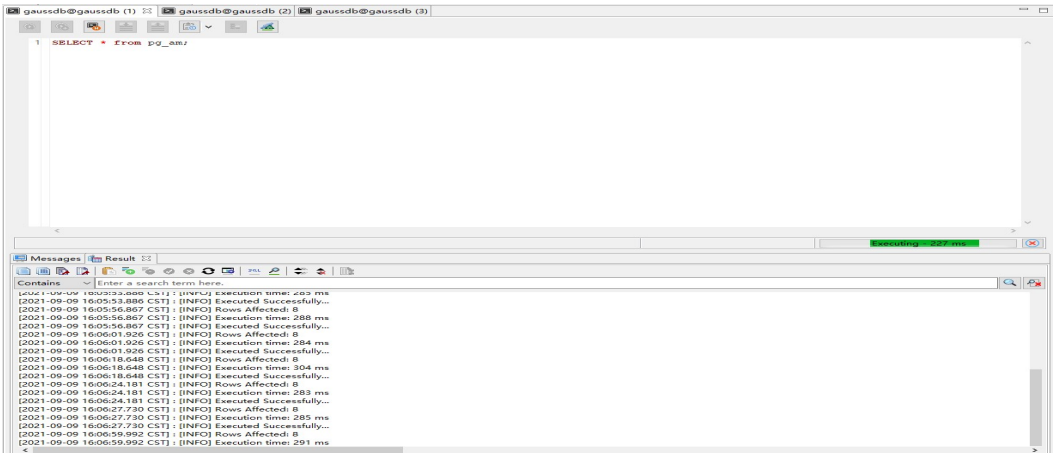
Searching for data in multiple cells may cause the system to display incorrect text results, because all information needs to be copied to the **Search** pane in plain text.

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.

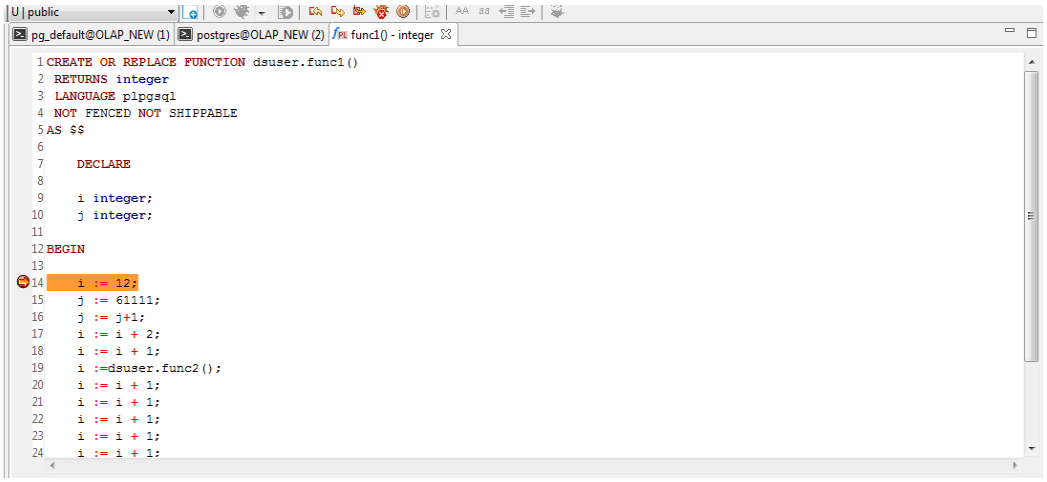
Debugging Duration

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.
- Debugging applies only to OLAP.

4.20.2 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**.

Follow the steps to view the SQL history:

Step 1 Click  in the **SQL Terminal** tab.

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 **SQL History** 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 **SQL History** 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 is listed in the progress bar. 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 **SQL History** 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

4.20.3 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 is 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 **preferences**.
- 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.
- 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** from the main menu.
- Alternatively click "Ctrl + Alt + S key to save SQL terminal or File terminal content in new file.

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

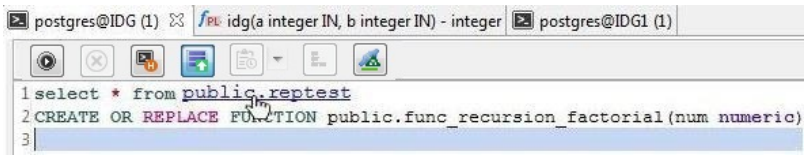
When there are unsaved changes in File Terminals, user will be given an option to save or cancel on graceful exit of data studio.

4.20.4 Viewing Object Properties in the SQL Terminal

Data Studio allows you to view table properties and functions/procedures.

Follow the steps to view table properties:

Step 1 Press **Ctrl** and point to the table name.



Step 2 Click the highlighted table name. The properties of the selected table are displayed.

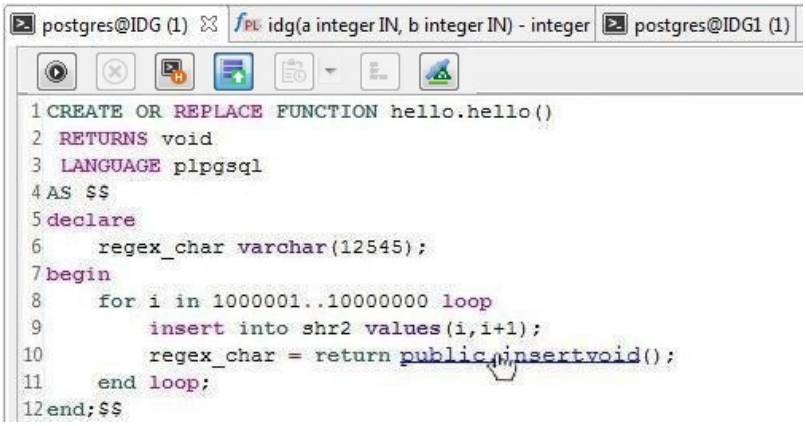
NOTE

The table properties are read-only.

----End

Follow the steps to view functions/procedures:

Step 1 Press **Ctrl** and point to the function/procedure name.



Step 2 Click the highlighted function/procedure name. The function/procedure is displayed in a new **PL/SQL Viewer** tab.

----End

Follow the steps to view the properties of a view:

Step 1 Press **Ctrl** and point to the view name.

Step 2 Click the highlighted view name. The properties of the selected view are displayed.

----End

Saving a Terminal Content Before Exiting Data Studio

Data Studio allows you to save the unsaved content of the terminal before exiting the application.

Follow the steps to save the content of the terminal:

Step 1 Click on the close button of the application. **Exit Application** dialog box will appear.

Step 2 Click **Graceful Exit**.

1. The **Saving File Terminal** dialog box appears. Unsaved dirty file terminal is displayed.

Step 3 Select the terminal to save.

Step 4 Click **OK**.

----End

NOTE

The **Saving File Terminal** dialog box will not appear in case of force exit.

4.20.5 Canceling the Execution of SQL Queries

Data Studio allows you to cancel the execution of an SQL query being executed in the **SQL Terminal**.

Follow the steps to cancel execution of an SQL query:

Step 1 Execute the SQL query in the **SQL Terminal**.

Step 2 Click  in the **SQL Terminal** or press **Shift**+**Esc**.

Alternatively, you can choose **Run > Cancel** from the main menu or right-click **SQL Terminal** and select **Cancel**, or select **Cancel** from **Progress View** tab.

----End

When you cancel the query, the execution stops at the currently executing SQL statement.

Database changes made by the canceled query are rolled back and the queries following the canceled query are not executed.

A query cannot be canceled and the **Result** tab shows the result when:

1. The server has finished execution of the query and is preparing the result.
2. The result of the executed query is being transferred from the server to the Data Studio client.

A query cannot be canceled while viewing the query **Execution Plan**. For more details, refer to **Viewing the Query Execution Plan and Cost**.

The **Messages** tab shows the query cancelation message.

NOTE

The **Cancel** button is enabled only during query execution.

4.20.6 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
   - Group by
   - Having
   - Order by
   - 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
   - RETURNING
4. The DELETE statement is made of the following clauses only:
   - Delete From Table name
   - USING clause (including JOIN)
   - Where
   - 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 4-21 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
		Place WHERE condition on single line
		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
		Line break after HAVING
		Indent HAVING condition
	ORDER BY	Line break before ORDER
		Line break after BY
		Indent column list
		Stack column list

Statement	Clause	Formatting Rule
	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 brace
		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
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

Statement	Clause	Formatting Rule
		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
		Place WHERE condition on single line
		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
		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

Statement	Clause	Formatting Rule
		Place WHERE condition on single line
		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  or 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	

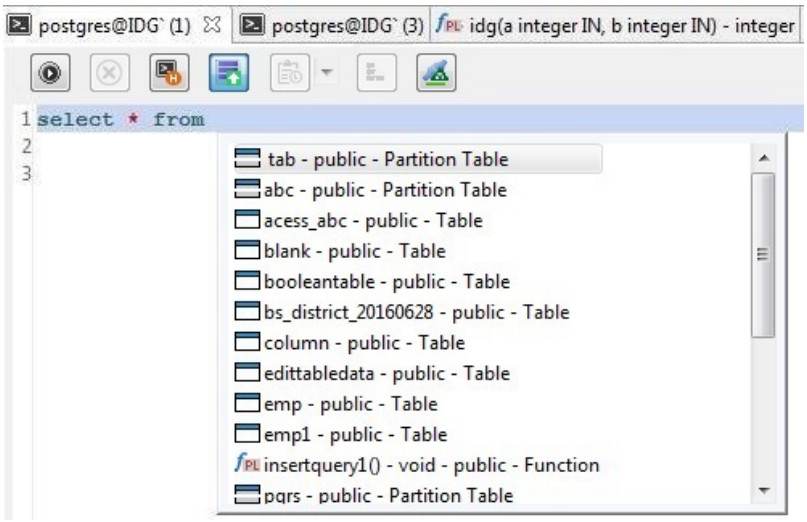
You can also customize SQL highlighting schemes for specific types of syntax. For details, see **Syntax Coloring**.

4.20.7 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**.

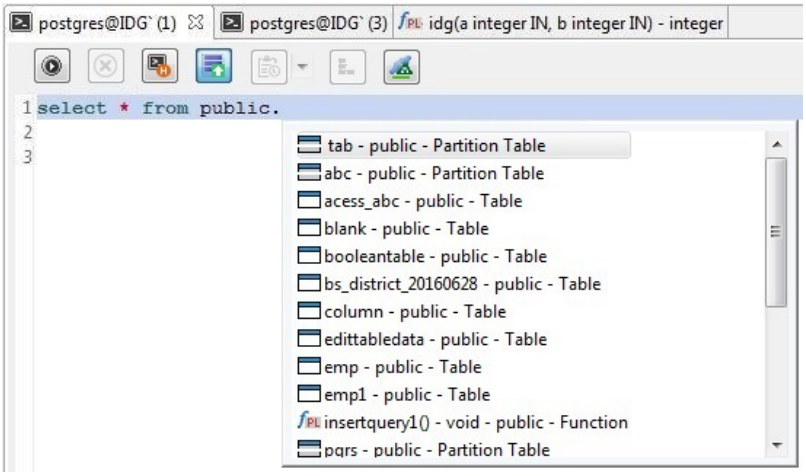
Follow the steps below to select a DB object:

Step 1 Press **Ctrl**+**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 Enter **.** (period) 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>
```

- Auto-suggest may show "Loading" in Terminal for following scenarios:
  - The object is not loaded due to the value mentioned in the **Load Limit** field. Refer to **Adding a Connection** for more information.
  - The object is not loaded since it is added in the **Exclude** list option. Refer to **Adding a Connection** for more information.
  - 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

4.20.8 Viewing the Query Execution Plan and Cost

The 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).

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 4-22 Explain plan options

Explain Plan Type	Parameters Selected	Columns
Include Analyze unchecked (default setting)	Verbose, Costs	Node type, startup cost, total cost, rows, width, and additional Info
Include Analyze checked	Analyze, Verbose, Costs, Buffers, Timing	Node type, startup cost, total cost, rows, width, Actual startup time, Actual total time, Actual Rows, Actual loops, and Additional Info

Additional Info column includes, predicate information (filter predicate, hash condition), distribution key and output information along with the node type information.

The tree view of plan categorizes nodes into 16 types. In tree view, each node will be preceded with corresponding type of icon. Following is the list of node categories with icons:

Table 4-23 Node categories with Icon

Node Category	Icon
Aggregate	
Group Aggregate	
Function	
Hash	
Hash Join	
Nested Loop	
Nested Loop Join	
Modify Table	
Partition Iterator	
Row Adapter	

Node Category	Icon
Seq Scan on	
Set Operator	
Sort	
Stream	
Union	
Unknown	

Hover over the highlighted cells to identify the heaviest, costliest, and slowest node. Cells will be highlighted only for tree view.

If multiple queries are selected, explain plan with/without analyze will be displayed only for last query selected.

Each time execution plan is executed, the plan opens in a new tab.

If the connection is lost and the database is still connected in Object Browser, then **Connection Error** dialog box is displayed:

- **Yes** - The connection is reestablished and retrieves explain plan and cost.
- **No** - Disconnects database in Object Browser.

Toolbar menu in the **Execution Plan** window:

Toolbar Name	Toolbar Icon	Description
Tree Format		This icon is used view explain plan in tree format.
Text Format		This icon is used view explain plan in text format.
Copy		This icon is used to copy selected content from result window to clipboard. Shortcut key - **Ctrl+C**.
Save		This icon is used to save the explain plan in text format.

Refer to **Execute SQL Queries** for information refresh, SQL preview, and search bar.

Refresh operation re-executes the explain/analyze query and refreshes the plan in the existing tab.

The result is displayed in the **Messages** tab.

----End

4.20.9 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 helps to refine query to enhance query and server performance. It helps to analyze the query path taken by the database and identifies 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.

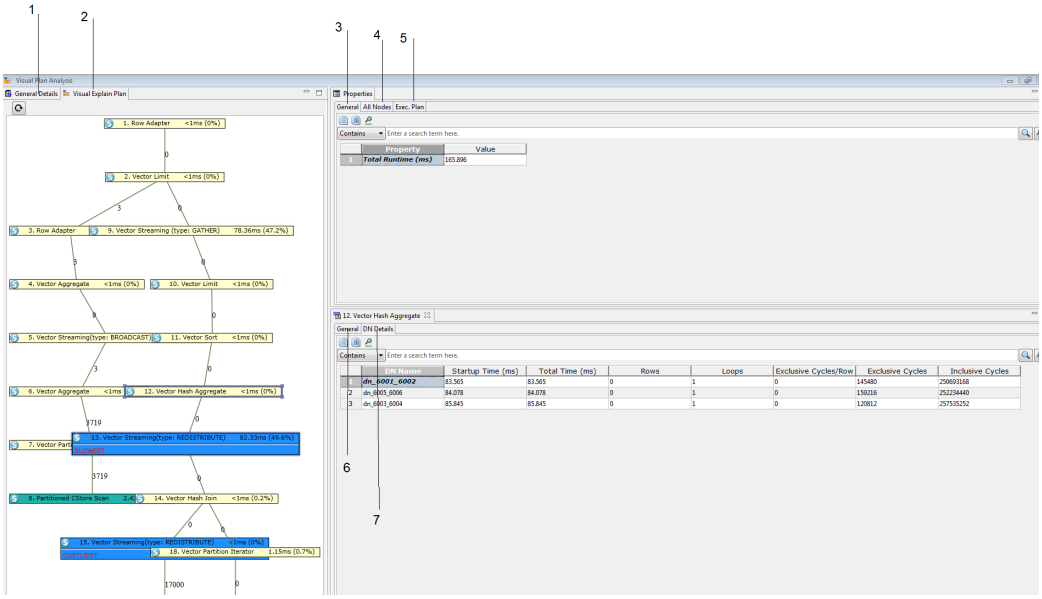
Slowest: Highest execution time by a plan node.

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. Alternatively, press **ALT+CTRL+X** together.

Visual Plan Analysis window is displayed.

Refer to **Viewing the Query Execution Plan and Cost** 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	Name of the 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** section for description on copy and search toolbar options.

----End

4.20.10 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 **Transaction** 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 4-28** 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 **Query Results** 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.

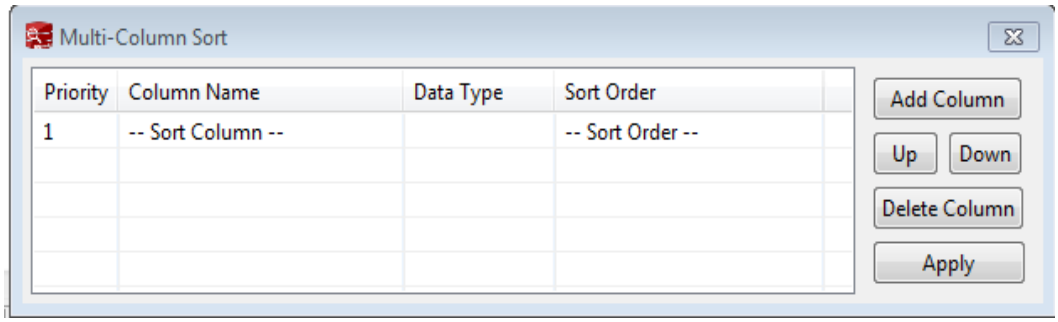
The feature is available for the following pages:

- Result Set Tab
- Edit Table Data Window
- View Table Data Window
- Batch Drop Result Window

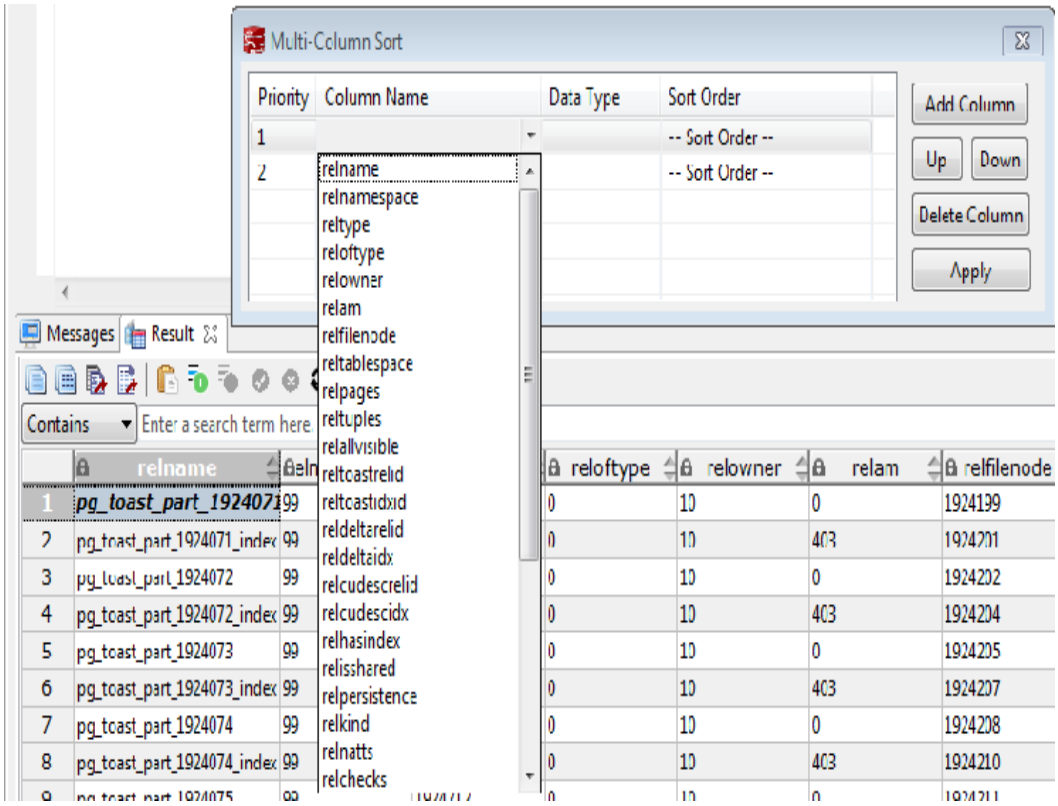
Follow the steps below to access Multi-column sort:

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 4-24 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 4-25 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 on 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 4-26 Toolbar Menus

Toolbar Name	Toolbar Icon	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 **Query Results** to set this preference. The shortcut key is **Ctrl+Shift+C**.

Toolbar Name	Toolbar Icon	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. For details, see **Paste**.
Add		This button is used to add a row to the result set. For details, see **Insert**.
Delete		This button is used to delete a row from the result set. For details, see **Delete**.
Save		This button is used to save the changes made in the result set. For details, see **Editing Table Data**.
Rollback		This button is used to roll back the changes made to the result set. For details, see **Editing Table Data**.
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. For details, see **Editing Table Data**.

Toolbar Name	Toolbar Icon	Description
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.
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 **Result Data Encoding** 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.

Option	Description
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 **Installing and 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 **Query Results**.
- 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 **Query/Function/Procedure Backup** 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 are 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

Below are invalid inputs to this 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 **Formatter**.

----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 **Formatter**.

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 **Formatter**.

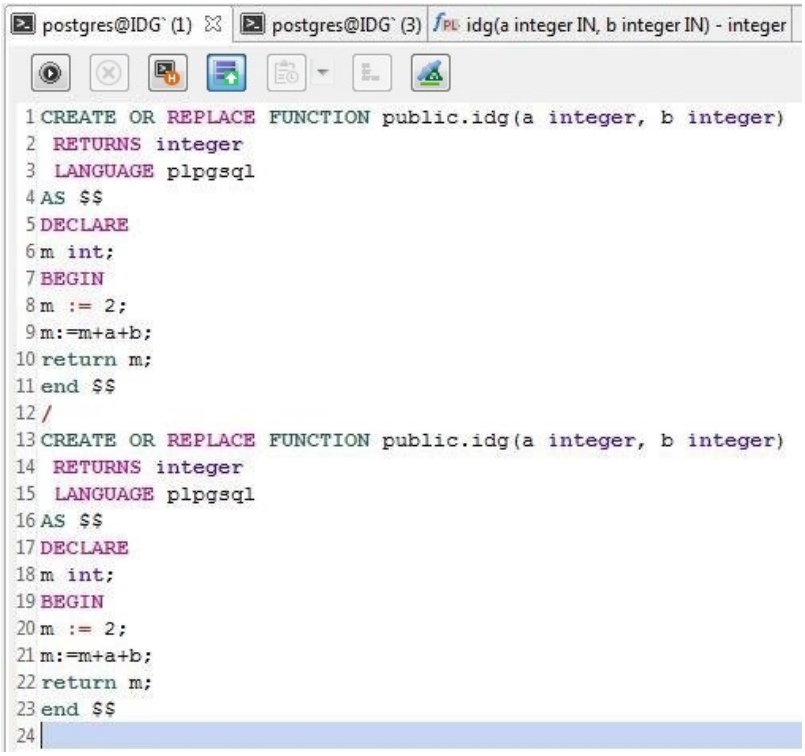
----End

Execute Multiple Functions/Procedures or Queries

Follow the steps below to execute multiple functions/procedures:

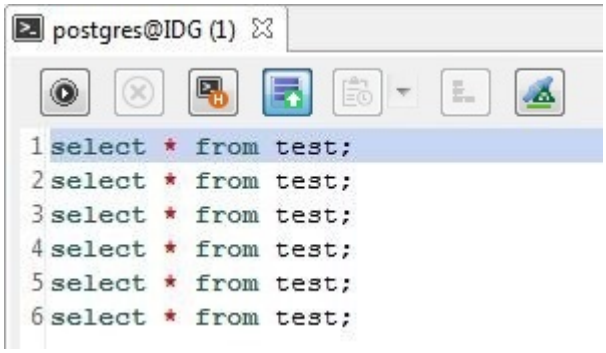
Insert a forward slash (/) in a new line after the function/procedure in the **SQL Terminal**.

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. Refer to **Adding/Modifying Templates** section for information on adding, removing, and creating new templates.

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".

Exact Match	Display List
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

4.20.11 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 is displayed.

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.

----End

The Messages tab shows the **Execution Time**, **Total Result Records Fetched**, and the path where the file is saved.

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

4.20.12 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

4.21 Batch Operation

4.21.1 Overview

You can view accessible database objects in the navigation tree in **Object Browser**. Schema are displayed under databases, and tables are displayed under schemas.

Object Browser displays only the objects that meet the following minimum permission requirements of the current user.

Object Type	Permissions displayed in Object Browser
Database	Connect
Schema	Usage
Table	Select
Column	Select
Sequences	Usage
Function/Procedure	Execute

The child objects of the objects accessible to you do not need to be displayed in **Object Browser**. For example, if you have the permission to access a table but does not have the permission to access a column in the table, **Object Browser** only displays the columns you can access. If access to an object is revoked during an operation on the object, an error message will be displayed, indicating that you do not have permissions to perform the operation. After you refresh **Object Browser**, the object will not be displayed.

The following objects can be displayed in the navigation tree:

- Schemas
- Functions/Procedures
- Tables
- Sequences
- Views
- Columns, constraints, and indexes

All default created schemas, except for the **public** schema, are grouped under **Catalogs**. User schemas are displayed under their databases in **Schemas**.

NOTE

The filter option in **Object Browser** opens a new tab, where you can specify the search scope. Press **Enter** to start the search. **Object Browser** also provides a search bar. You can search for an object by name. In an expanded navigation tree, only the objects that match the filter criteria are displayed.

In a collapsed navigation tree, the filtering rule takes effect when a node is expanded.

4.21.2 Dropping a Batch of Objects

The batch drop operation allows you select multiple objects to drop. You can also perform batch drop operation on searched objects.

NOTE

- Batch drop is allowed only within a database.
- Batch drop of system objects will result in error, since system objects cannot be dropped.

Follow the steps below to drop objects in a batch:

Step 1 Press **Ctrl+left-click** (select objects one by one) or **Shift+left-click** (select objects in a bunch) to select the objects to be dropped.

Step 2 Right-click and select **Drop Objects**.

Drop Objects tab displays with the list of objects to be dropped.

Column Name	Description	Example
Type	Displays information on the object type.	table, views
Name	Displays the name of the object.	public.bs_operation_201804
Query	Displays the query that will be executed to drop the object.	DROP TABLE IF EXISTS public.a123

Column Name	Description	Example
Status	Displays the status of the drop operation. • - To start: The drop operation yet to be initiated. • - In progress: The object is currently being dropped. • - Completed: The drop operation has been completed. • - Error: The object has not been dropped due to an error.	• To start • In progress • Completed • Error
Error Message	Displays the failure reason of the drop operation.	Table "abc" does not exist. Skip it.

Step 3 Select the required drop option:

Option	Description
Cascade	Cascade drop operation drops their dependent objects and attributes. The dependent objects that are dropped will be removed from the Object Browser only after refresh operation is performed.
Atomic	Atomic drop operation drops all objects in case of success or drops none in case of a failure.
No selection	Un-selection of Atomic or Cascade does not drop dependent objects.

Step 4 Click **Start**.

Runs - Displays the number of objects that are dropped from the total list of objects.

Errors - Displays the number of object that was not dropped due to errors.

Step 5 Click **Stop** or close the **Drop Objects** dialog to stop the drop operation.

Refer to **Execute SQL Queries** section for information on copy, advanced copy, show/hide search bar, sort, and column reorder options.

NOTE

- Select part of cell content and press **Ctrl+C** or click  to copy selected text from a cell.
- When you select multiple objects in object browser to drop, a batch drop window is opened and its menu icons are enabled in the menu bar. If you disconnect the database, the icons will remain disabled and will not be enabled even after reconnecting. You need to reselect the objects to drop and the selected objects will be available in the new terminal window.

----End

4.21.3 Granting/Revoking Privileges

The batch grant/revoke operation allows you select multiple objects to grant/revoke privileges. You can also perform batch grant/revoke operation on searched objects.

This feature is only available for OLAP, not for OLTP.

NOTE

Batch grant/revoke is allowed only with the same object type within that schema.

Follow the steps to grant/revoke privileges in a batch:

Step 1 Press **Ctrl+left-click** (select objects one by one) or **Shift+left-click** (select objects in a bunch) to select the objects to grant/revoke privileges.

Step 2 Right-click and select **Grant/Revoke**.

Grant/Revoke dialog box is displayed.

Step 3 Refer to **Granting/Revoking a Privilege** section to grant/revoke privilege.

----End

4.22 Personalizing Data Studio

4.22.1 General

This section provides details on how to customize shortcut keys.

Setting the Shortcut Keys

You can customize the Data Studio shortcut keys as required.

Follow the steps below to set or modify the shortcut keys:

Step 1 Choose **Settings** > **Preferences** from the main menu.

The **Preferences** dialog box is displayed.

Step 2 Choose **General > Shortcut Mapper**.

The **Shortcut Mapper** pane is displayed.

Step 3 Select the required shortcut key and click **Modify**.

Step 4 Enter the required shortcut key in the **Binding** text box.

For example, to change the shortcut key for **Step Into** from **F7** to **F6**, enter **F6** in the **Binding** text box.

Step 5 Click **OK**. The **Restart Data Studio** dialog box is displayed.

NOTE

Multiple shortcut keys can be modified before restarting Data Studio.

Step 6 Click **Yes** to restart Data Studio. If any export, import or execution operations are in progress, then the **Restart Confirmation** dialog box is displayed.

Step 7 Click **OK** to close running jobs and restart or click **Cancel** to abort restart operation.

----End

Follow the steps below to remove the shortcut keys:

Step 1 Choose **Settings** > **Preferences** from the main menu.

The **Preferences** dialog box is displayed.

Step 2 Choose **General** > **Shortcut Mapper**.

The **Shortcut Mapper** pane is displayed.

Step 3 Select the required shortcut key and click **Unbind Key**.

Step 4 Click **Ok**. The **Restart Data Studio** window is displayed.

NOTE

Multiple shortcut keys can be removed before restarting Data Studio.

Step 5 Click **Yes** to restart Data Studio. If any export, import or execution operations are in progress, then the **Process is running** dialog box is displayed.

Step 6 Click **OK** to wait for operations to complete or click **Force Restart** to discard operations.

----End

Follow the steps below to restore the default shortcut keys:

Step 1 Choose **Settings** > **Preferences** from the main menu.

The **Preferences** dialog box is displayed.

Step 2 Choose **General** > **Shortcut Mapper**.

The **Shortcut Mapper** pane is displayed.

Step 3 Click **Restore Defaults**. For more information on default shortcut keys, refer to **Data Studio Right-Click Menus**.

Step 4 Click **Ok**.

The **Restart Data Studio** window is displayed.

Step 5 Click **Yes** to restart Data Studio. If any export, import or execution operations are in progress, the **Process is running** dialog box displays.

Step 6 Click **OK** to wait for operations to complete or click **Force Restart** to discard operations.

----End

Shortcut Keys

Data Studio supports keyboard short cut keys similar to other windows based application. The following table lists some of the shortcut keys for effective usage of the functionalities provided by Data Studio. For details about how to customize shortcut keys, see **Setting the Shortcut Keys**.

Table 4-27 Default shortcut keys of Data Studio

Function	Shortcut Key
Sorts the result sets of visual charts, edit tables, and queries in ascending, descending, or server receiving order	Alt+Click
Open the Help menu	Alt+H
Save the SQL script	Ctrl+S
Edit menu	Alt+E
Compile/Execute SQL terminal statements	Ctrl+Enter
Search and Replace	Ctrl+F
Search for the previous one	Shift+F3
Search for the next one	F3
Redoing	Ctrl+Y
On the Edit Table Data tab page, copy Execution Time and Status	Ctrl+Shift+K
Copy the database object from the automatic recommendation list	Alt+U
Open the Call Stack, the Breakpoints pane, and the Variables pane	Alt+V
Open the SQL script	Ctrl+O
Step Skip	F8
Step into	F7
Single step exit	Shift+F7

Function	Shortcut Key
Comment out or cancel the comment line	Ctrl+/
Locate the first element in the Object Browser	Alt+Page Up or Alt+Home
Locate the last element in the Object Browser	Alt+Page Down or Alt+End
Locate to row	Ctrl+G
Disconnect the connection	Ctrl+Shift+D
Formatting (SQL and PL/SQL)	Ctrl+Shift+F
Change the value to uppercase	Ctrl+Shift+U
Change the value to lowercase	Ctrl+Shift+L
Updates the cells or columns in the Edit Table Data, Properties, and Results windows. Click the cell or column header to enable this option	F2
Close the PL/SQL Viewer tab page, Table Data View tab page, Execute Query tab page, or Properties tab page	Shift+F4
Continue the PL/SQL debugging	F9
Shearing	Ctrl+X
Copy Object Browser or the name of the object modified in the terminal. Copy the selected data from the Terminal, Result, Table Data, or Edit Table Data tab page.	Ctrl+C
Copy the data on the Result, Table Data, or Edit Table Data tab page. The data contains/does not contain the column title and row number	Ctrl+Shift+C
Copy the query result on the Edit Table Data tab page	Ctrl+Alt+C
Copy the content on the Variable tab page	Alt+K
Copy the content on the Call Stack tab page	Alt+J
Copy the content on the Breakpoint tab page	Alt+Y
Visualized interpretation plan	Alt+Ctrl+X

Function	Shortcut Key
Online help (displaying the user manual)	F1
Template	Alt+Ctrl+Space
Switch to the first SQL Terminal tab page	Alt+S
Select All	Ctrl+A
Setting menu	Alt+G
Refresh (in the Object Browser area)	F5
Search Object	Ctrl+Shift+S
Debugging menu	Alt+D
Debugging template	F10
Debugging the Database Object	Ctrl+D
Highlight Object Browser	Alt+X
File menu	Alt+F
Creating a connection	Ctrl+N
Running menu	Alt+R
Switch between the SQL Terminal tab page	Ctrl+Page Up or Ctrl+Page Down
Expand/Collapse All Objects	Ctrl+M
Pastes	Ctrl+V
Collapsible object browsing navigation tree	Alt+Q
Execute	Ctrl+E
Execution plan and expense	Ctrl+Shift+X
Stop the query in the running state	Shift+Esc
Comment/Cancel the comment line or the entire segment	Ctrl+Shift+/
List of automatically recommended database objects	Ctrl+Space

4.22.2 Editor

This section provides details on how to personalize syntax coloring, SQL history information, templates, and formatter.

Syntax Coloring

Follow the steps to customize the SQL highlight color:

Step 1 Choose **Settings** > **Preferences** from the main menu.

The **Preferences** dialog box is displayed.

Step 2 Choose **Editor > Syntax Coloring**.

The **Syntax Coloring** pane is displayed.

Step 3 Click the color button to customize the color for the type of syntax.

For example, click  to customize the color for **Strings**. The color picker dialog box is displayed.

Use the color picker to set the required color for a specific syntax category. You can choose basic colors or define custom colors in the color picker.

NOTE

Click **Restore Defaults** from **Syntax Coloring** pane to reset to default color scheme.

Step 4 Click **OK**. The **Restart Data Studio** dialog box is displayed.

Step 5 Click **Yes** to restart Data Studio. If any export, import or execution operations are in progress, then Data Studio displays the **Process is running** dialog box**.**

Step 6 Click **Force Restart** to discard operations and restart Data Studio. Click **OK** to continue performing operations.

NOTE

The *Preferences.prefs* file contains the custom color settings. If the file is corrupted, Data Studio will display the default values.

The custom color(s) will be set after you restart Data Studio.

----End

SQL History

You can customize Data Studio to set the number of SQL history count that can be made available and also the number of characters for the query for each of the query saved in SQL history.

Follow the steps to customize the number of executed queries and number of characters in the query to be saved in SQL History:

Step 1 Choose **Settings** > **Preferences** from the main menu.

The **Preferences** dialog box is displayed.

Step 2 Choose **Editor > SQL History**.

The **SQL History** pane is displayed.

Step 3 Set the number of queries to be saved in **SQL History Count** field.

NOTE

Minimum value is 1 and maximum is 1000. The current value set for this preference will be displayed.

Step 4 Set the number of characters to be allowed in each query that is saved in the SQL History in the **SQL Query Characters** field.

NOTE

Minimum value is 1 and maximum is 1000. Enter "0" in this field to set no character limit. The current value set for this preference will be displayed.

Step 5 Click **Apply**.

Step 6 Click **OK**.

NOTE

- Click **Restore Defaults** from **SQL History** pane to reset to default value.
- The default value for **SQL History Count** is 50 and **SQL Query Characters** it is 1000.
- If the input value is less than the original one, data may be lost. In this case, a message is displayed to notify you of the data loss risk and ask you whether to proceed with the operation.
- If there are unsaved changes and you navigate away from this pane, then a message displays to state that there are unsaved changes.
- Pinned queries are not affected by the changes made to the **SQL History Count** field. **Example:** If the number of pinned queries is 50 and the **SQL History Count** is set to 25, then SQL History will show 50 pinned queries.
- The **SQL Query Characters** changes affects only queries added post the configuration change.

----End

Adding Templates

You can customize Data Studio to create new, edit existing, and remove templates. Refer to the **Using Templates** section for detailed information on templates.

NOTE

Restoring the settings to default removes all user defined templates from the list.

Follow the steps below to create templates:

Step 1 Choose **Settings** > **Preferences** from the main menu.

The **Preferences** dialog box is displayed.

Step 2 Choose **Editor** > **Templates**.

The **Templates** pane is displayed.

Step 3 Click **New**.

Step 4 Enter a name for the template in the **Name** field.

Step 5 Enter description in the **Description** field.

Step 6 Enter the SQL statement pattern in the **Pattern** field.

NOTE

The text entered in **Pattern** field will be syntax highlighted.

Step 7 Click **OK**.

----End

Modifying Templates

Follow the steps below to edit templates:

Step 1 Choose **Settings** > **Preferences** from the main menu.

The **Preferences** dialog box is displayed.

Step 2 Choose **Editor > Templates**.

The **Templates** pane is displayed.

Step 3 Click **Edit**.

Step 4 Edit the name in the **Name** field, if required.

Step 5 Edit the description in the **Description** field, if required.

Step 6 Edit the SQL statement pattern in the **Pattern** field, if required.

NOTE

The text entered in **Pattern** field will be syntax highlighted.

Step 7 Click **OK**.

----End

Removing Templates

Follow the steps below to remove templates:

Step 1 Choose **Settings** > **Preferences** from the main menu.

The **Preferences** dialog box is displayed.

Step 2 Choose **Editor > Templates**.

The **Templates** pane is displayed.

Step 3 Select the template to be removed, and click **Remove**.

The template is removed from the **Templates** list.

NOTE

Default templates that are removed can be added back using **Restore Removed** option. It will restore the template to the last updated change. **Restore Removed** option is not applicable to user defined templates.

----End

Reverting to Default Templates

Follow the steps below to revert to default templates:

Step 1 Choose **Settings** > **Preferences** from the main menu.

The **Preferences** dialog box is displayed.

Step 2 Choose **Editor > Templates**.

The **Templates** pane is displayed.

Step 3 Select at least one default template that is modified to revert to default template settings.

Step 4 Click **Revert to Default**.

----End

Formatter

You can customize Data Studio to set the tab width and convert tab to spaces while performing indent and unindent operation. Refer to **Indent/Un-indent Lines** section to perform indent/unindent operation and replace tab with spaces.

Follow the steps to customize the indent size and convert tab to spaces:

Step 1 Choose **Settings** > **Preferences** from the main menu.

The **Preferences** dialog box is displayed.

Step 2 Choose **Editor > Formatter**.

The **Formatter** pane is displayed.

Step 3 Select the **Insert Space** option to replace tab with spaces or **Insert Tab** to add/remove tabs while indenting/unindenting lines.

Step 4 Enter the indent size in **Indent Size**. Based on the number specified in this field, the indent/unindent/space length is defined.

----End

Transaction

Follow the steps to edit Transaction settings:

Step 1 Choose **Settings** > **Preferences** from the main menu.

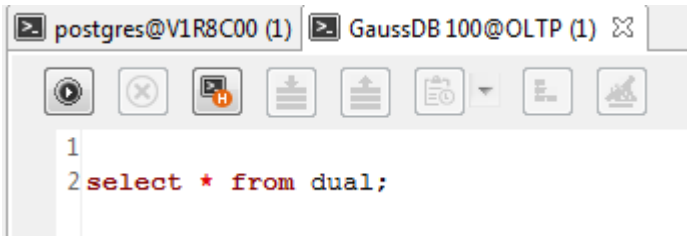
The **Preferences** dialog box is displayed.

Step 2 Choose **Editor > Transaction**

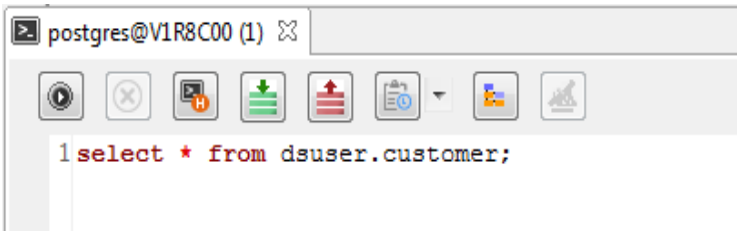
The **Transaction** pane is displayed.

Step 3 In the **Auto Commit** window, you can perform the following operations:

- Select **Enable** to switch on the auto commit feature. In this case commit and rollback button will be disabled. Transaction will be committed automatically.



- Select **Disable** to switch off the auto commit feature. Commit and Rollback button can be used manually for commiting or reverting changes.



NOTE

Default behavior for Auto-Commit is **ON**.

----End

Folding

Follow the steps for Folding:

Step 1 Choose **Settings** > **Preferences** from the main menu.

The **Preferences** dialog box is displayed.

Step 2 Choose **Editor** > **Folding**.

The Folding pane is displayed.

Step 3 Select **Enable** or **Disable**. By default, **Enable** is selected.

- **Enable:** This indicates enable SQL folding feature. Supported SQL statements can be folded or unfolded.
- **Disable:** This indicates disable SQL folding feature.

NOTE

Modification in settings reflects in newly opened editor. The editor which is already opened will remain with previous settings until restart.

----End

Font

Follow the steps to set Font:

Step 1 Choose **Settings** > **Preferences** from the main menu.

The **Preferences** dialog box is displayed.

Step 2 Choose **Editor > Font**.

The Font pane is displayed.

Step 3 Provide required font size within range from 1 to 50. By default, font size is 10.

----End

Auto Suggest

Follow the steps for Auto Suggest:

Step 1 Choose **Settings** > **Preferences** from the main menu.

The **Preferences** dialog box is displayed.

Step 2 Choose **Editor > Auto Suggest**.

The **Auto Suggest** pane is displayed.

Step 3 In **Auto Suggest** pane, provide required number of character in **Auto Suggest Min Character**. Default value is 2. Range of number of Auto Suggest minimum characters are within 2 to 10.

For auto suggest, sorting can be as follows:

1. Keywords
2. Data types
3. Loaded Database Objects

NOTE

- Each group should be in sorted order.
- Databases are classified by keyword and data type.
- If database is not connected, then default keywords must be displayed.
- When you press dot (.) then only respective database objects should be displayed. Keywords/Data types should not be displayed.
- Auto suggest should be triggered by shortcuts.

----End

4.22.3 Environment

Session Setting

Follow the steps to set Data Studio and file encoding:

Step 1 Choose **Settings** > **Preferences** from the main menu.

The **Preferences** dialog box is displayed.

Step 2 Choose **Environment > Session Setting**.

The **Session Setting** pane is displayed.

Step 3 Select the required Data Studio encoding from **Data Studio Encoding** drop-down.

Step 4 Select the file encoding from **File Encoding** field.

NOTE

Data Studio supports only UTF-8 and GBK file encoding types.

Step 5 Click **OK**. The **Restart Data Studio** dialog box is displayed.

Step 6 Click **Yes** to restart Data Studio. If any export, import or execution operations are in progress, then Data Studio displays the **Process is running** dialog box.

Step 7 Click **Force Restart** to discard operations and restart Data Studio. Click **OK** to continue performing operations.

NOTE

Click **Restore Defaults** from **Session Setting** pane to reset to default values. The default value for Data Studio Encoding and File Encoding is **UTF-8**.

----End

SQL Assistant

Follow the steps to enable/disable SQL Assistant tool:

Step 1 Choose **Settings** > **Preferences** from the main menu.

The **Preferences** dialog box is displayed.

Step 2 Choose **Environment > Session Setting**.

The **Session Setting** pane is displayed.

Step 3 Select **Enable/Disable** from SQL Assistant section.

Step 4 Click **OK**.

NOTE

Click **Restore Defaults** from **Session Setting** pane to reset to default value. The default value for SQL Assistant is **Enable**.

----End

Query/Function/Procedure Backup

Refer to the **Backuping Unsaved Queries/Functions/Procedures** section for information on backup feature provided by Data Studio.

Follow the steps to enable/disable backup of unsaved data in SQL Terminal/PL/SQL Viewer:

Step 1 Choose **Settings** > **Preferences** from the main menu.

The **Preferences** dialog box is displayed.

Step 2 Choose **Environment > Session Setting**.

The **Session Setting** pane is displayed.

Step 3 Select/unselect **Auto Save** from **Auto Save** section.

Step 4 Set the time interval to backup the data in **Interval** field.

Step 5 Click **OK**.

NOTE

Click **Restore Defaults** from **Session Setting** pane to reset to default value. Backup of data will be enabled by default with 5 minutes as the default time interval.

----End

Follow the steps to enable/disable data encryption of saved data:

Step 1 Choose **Settings** > **Preferences** from the main menu.

The **Preferences** dialog box is displayed.

Step 2 Choose **Environment > Session Setting**.

The **Session Setting** pane is displayed.

Step 3 Select/unselect **Encryption** from Auto Save section.

Step 4 Click **OK**.

NOTE

Click **Restore Defaults** from **Session Setting** pane to reset to default value. Encryption will be enabled by default.

----End

Follow the steps to set the size of **Import Table Data Limit**/**Import File Data Limit**:

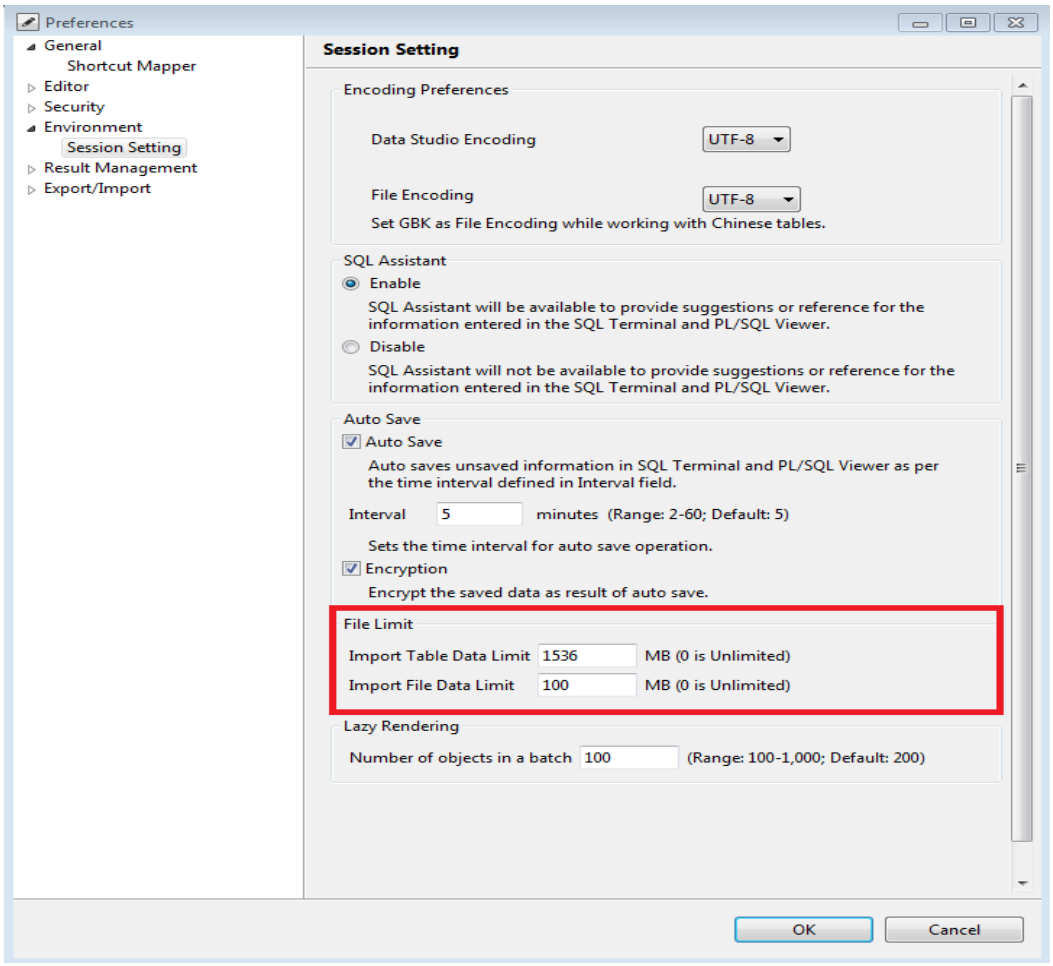
Step 1 Choose **Settings** > **Preferences** from the main menu.

The **Preferences** dialog box is displayed.

Step 2 Choose **Environment > Session Setting**.

The **Session Setting** pane is displayed.

In File Limit section **Import Table Data Limit** and **Import File Data Limit** parameters are displayed.



Import Table Data Limit value defines the maximum size of the table data to be imported.

Import File Data Limit value defines the maximum size of the file to be imported.

Step 3 Click **OK**.

NOTE

Mentioned values in the above screenshot are the default values.

----End

Follow the steps to perform rendering:

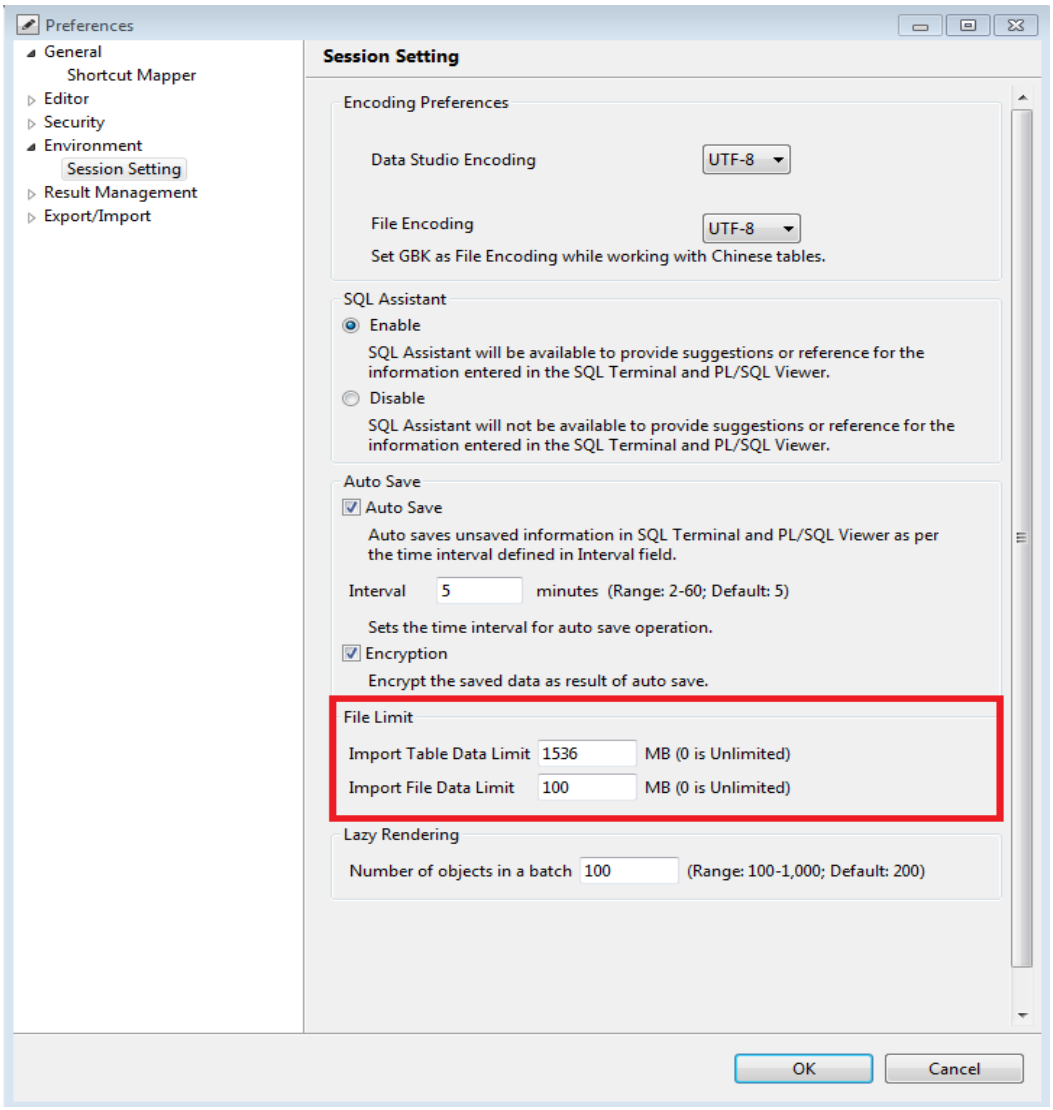
Step 1 Choose **Settings** > **Preferences** from the main menu.

The **Preferences** dialog box is displayed.

Step 2 Choose **Environment > Session Setting**.

The **Session Setting** pane is displayed.

In Lazy Rendering section, **Number of objects in a batch** parameter is displayed.



Step 3 Provide required number of objects in a batch, want to be rendered. Range is from 100 to 1000. Default value is **200**.

If you provide any value which is less than 100 or more than 1000, then **Invalid Range, (100 -1000)** error message is displayed.

Step 4 Click **OK**.

----End

4.22.4 Result Management

This section provides details on how to personalize the column width, number of records to be fetched in the query results, and result copy of column header or row number using the **Query Results** setting.

Query Results

Set column width of query results:

Step 1 Choose **Settings** > **Preferences** from the main menu.

The **Preferences** dialog box is displayed.

Step 2 Choose **Result Management > Query Results**.

The **Query Results** pane is displayed.

Step 3 Select the required option.

Column Width customization options:

Option	Outcome
Content Length	Selecting this option enables you to set the column width based on the content length of the column.
Custom Length	Selecting this option enables you to set the column width based on the value entered in this field. **NOTE** This column accepts value between 100 and 500.

Step 4 Click **OK**.

NOTE

Click **Restore Defaults** from **Query Results** pane to reset to default values. The default value is **Content Length**.

----End

Set the number of records to be fetched in the query results:

Step 1 Choose **Settings** > **Preferences** from the main menu.

The **Preferences** dialog box is displayed.

Step 2 Choose **Result Management > Query Results**.

The **Query Results** pane is displayed.

Step 3 Select the required option.

Option	Outcome
Fetch All records	Selecting this option enables you to fetch all the records in the query results.
Fetch custom number of records	Selecting this option enables you to set the number of records that needs to be fetched in the query results. **NOTE** This column accepts value between 100 and 5000.

Step 4 Click **OK**.

NOTE

Click **Restore Defaults** from **Query Results** pane to reset to default values. The default value is **Fetch custom number of records (1000)**.

----End

Set preference to copy column name and row number from query results:

Step 1 Choose **Settings** > **Preferences** from the main menu.

The **Preferences** dialog box is displayed.

Step 2 Choose **Result Management > Query Results**.

The **Query Results** pane is displayed.

Step 3 Select the required option.

Option	Outcome
Include column header	Selecting this option enables you to copy column headers from the query results.
Include row number	Selecting this option enables you to copy the selected content along with the row number from the query results.

Step 4 Click **OK**.

NOTE

Click **Restore Defaults** from **Query Results** pane to reset to default values. The default value is **Include column header**.

----End

Set preference to decide the behavior of opening up result set window/s:

Step 1 Choose **Settings** > **Preferences** from the main menu.

The **Preferences** dialog box is displayed.

Step 2 Go to **Result Management > Result Window**.

Step 3 Select the required option.

Option	Outcome
Overwrite Resultset	Current result set opened window/s are closed and new result set window is opened.
Retain Current	New result set window/s are opened retaining already opened result set window/s.

Step 4 Click **OK**.

----End

Edit Table Data

Set save behavior of edit table data operation:

Step 1 Choose **Settings** > **Preferences** from the main menu. The **Preferences** dialog box is displayed.

Step 2 Choose **Result Management > Edit Table Data**. The **Edit Table Data** pane is displayed.

Select the required option:

Table 4-28 Edit table data

Server Type	Auto Commit	Reuse Connection	Table Data Save Option	Behavior
GaussDB(DWS)	ON	ON	Save Valid Data	All the valid data will be saved and committed. Incorrect data will be omitted.
	ON	ON	Do Not Save	If an error occurs , no data will be saved.
	ON	OFF	Save Valid Data	All the valid data will be saved and committed. Incorrect data will be omitted.
	ON	OFF	Do Not Save	If an error occurs, no data will be saved.
	OFF	ON	Save Valid Data	If an error occurs, no data will be saved. Perform Commit/ Rollback to proceed further.
	OFF	ON	Do Not Save	If an error occurs, no data will be saved. Perform Commit/ Rollback to proceed further.

Step 3 Click **OK**.

NOTE

Click **Restore Defaults** from **Edit Table Data** pane to reset to default values. The default value is **Save Valid Data**.

----End

Result Data Encoding

You can enable/disable to display the data encoding type in edit, view, and query results window.

Follow the steps to modify display of encoding option:

Step 1 Choose **Settings** > **Preferences** from the main menu.

The **Preferences** dialog box is displayed.

Step 2 Choose **Result Management > Query Results**.

The **Query Results** pane is displayed.

Step 3 Select **Include result data encoding** to include the **Encoding** drop-down in edit, view, and query results table.

Step 4 Click **OK**.

NOTE

- Click **Restore Defaults** from **Result Management** pane to reset to default values. **Include result data encoding** will be unselected by default.
- Edit table, view table properties and query execution again to apply the changes.

----End

4.22.5 Security

This section provides details on how to personalize password and security disclaimer display.

Save Password Permanently

You can enable/disable to display the permanent option to save password in the connection window.

Follow the steps below to modify display of permanent save password option:

Step 1 Choose **Settings** > **Preferences** from the main menu.

The **Preferences** dialog box is displayed.

Step 2 Choose **Security > Password**.

The **Password** pane is displayed.

Step 3 Select the required option. Refer table below to understand the customization options available:

Option	Outcome
Yes	Selecting this option enables you to view the "Permanently" save password option from the **Save Password** drop-down list in the connection window.
No	Selecting this option removes the "Permanently" save password option from the **Save Password** drop-down list in the connection window and removes the saved passwords.

Step 4 Click **OK**. The **Restart Data Studio** dialog box is displayed.

Step 5 Click **Yes** to restart Data Studio. If any export, import or execution operations are in progress, then Data Studio displays the **Process is running** dialog box.

Click **Force Restart** to discard operations and restart Data Studio.

Step 6 Click **OK** to continue performing operations.

NOTE

Click **Restore Defaults** from **Password** pane to reset to default values. The default value is **No**.

----End

Password Expiry

This section provides details on how to continue/discontinue working with Data Studio once password expires using the Password setting.

Follow the steps below to modify the behavior of Data Studio once password expires:

Step 1 Choose **Settings** > **Preferences** from the main menu.

The **Preferences** dialog box is displayed.

Step 2 Choose **Security > Password**.

The **Password** pane is displayed.

Step 3 Select the required option. Refer table below to understand the customization options available:

Option	Outcome
Yes	Selecting this option allows you to login to Data Studio after the password has expired. **NOTE** A message displays informing you that the password has expired and some operations may not work as expected in the following scenarios: • Establishing a new connection. • Editing a connection. • Connecting to a database while creating the database when no other database is connected in that connection profile. • Connecting to a database when no other database is connected in that connection profile.
No	Selecting this option will not allow you to login to Data Studio once the password has expired. A message displays informing you that the password has expired.

Step 4 Click **OK**. The **Restart Data Studio** dialog box is displayed.

Step 5 Click **Yes** to restart Data Studio. If any export, import or execution operations are in progress, then Data Studio displays the **Process is running** dialog box.

Step 6 Click **Force Restart** to discard operations and restart Data Studio. Click **OK** to continue performing operations.

NOTE

The default value is **Yes**.

----End

Security Disclaimer

You can enable/disable to display the security disclaimer for any unsecured connection/file operations.

Follow the steps below to modify the display of security disclaimer:

Step 1 Choose **Settings** > **Preferences** from the main menu.

The **Preferences** dialog box is displayed.

Step 2 Choose **Security > Security Disclaimer**.

The **Security Disclaimer** pane is displayed.

Step 3 Select the required option. Refer table below to understand the customization options available:

Option	Outcome
Enable	Selecting this option displays the security disclaimer each time you try to establish an unsecure connection or perform a file operation.

Option	Outcome
Disable	Selecting this option will not display the security disclaimer while establishing an unsecure connection or performing a file operation. You need to agree to the security implications that may arise due to unsecure connection.

Step 4 Click **OK**. The **Restart Data Studio** dialog box is displayed.

Step 5 Click **Yes** to restart Data Studio. If any export, import or execution operations are in progress, then Data Studio displays the **Process is running** dialog box.

Step 6 Click **Force Restart** to discard operations and restart Data Studio. Click **OK** to continue performing operations.

NOTE

Click **Restore Defaults** from **Security Disclaimer** pane to reset to default values. The default value is **Enable**.

----End

4.23 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 4-29 describes the minimum access permissions required to list objects in the **Object Browser**.

Table 4-29 Minimum permission requirements

Object Type	Type	Object Browser - Minimum Permission
Database	Create, Connect, Temporary/Temp, All	Connect
Schema	Create, Usage, All	Usage
Table	Select, Insert, Update, Delete, Truncate, References, All	Select

Object Type	Type	Object Browser - Minimum Permission
Column	Select, Insert, Update, References, All	Select
View	Select, Insert, Update, Delete, Truncate, References, All	Select
Sequences	Usage, Select, Update, All	Usage
Function	Execute, All	Execute

To improve the performance of find and replace 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.

4.24 Security Management

4.24.1 Overview

NOTICE

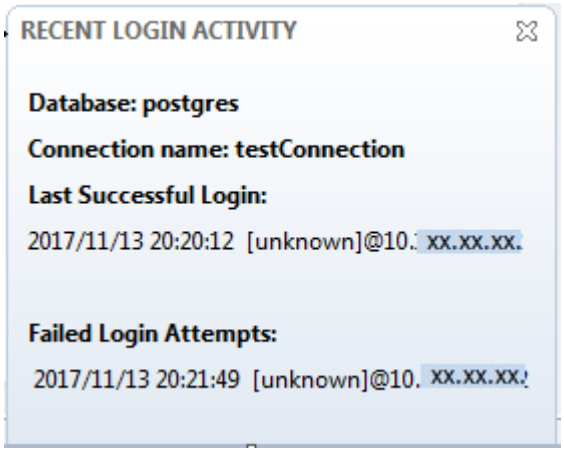
Ensure that the operating system and the required software's (refer to **System Requirements** for more details) are updated with the latest patches to prevent vulnerabilities and other security issues.

This section provides the security management information for Data Studio.

4.24.2 Login History

The following information is critical to the security management for Data Studio:

- 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.

4.24.3 Password Expiry Notification

The following information is critical to manage security for Data Studio:

- 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.

4.24.4 Securing the Application In-Memory Data

The following information is critical to manage security for Data Studio:

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.

Alternatively, you can choose **Do Not Save** while connecting to the database, so that password does not get saved in the memory.

4.24.5 Data Encryption for Saved Data

The following information is critical to manage security for Data Studio:

You can ensure encryption of auto saved data by enabling encryption option from **Preferences** page. Refer to **Query/Function/Procedure Backup** section for steps to encrypt the saved data.

4.24.6 SQL History

The following information is critical to manage security for Data Studio:

- 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
- Few 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 [...]]

4.24.7 SSL Certificates

NOTICE

The information on using SSL certificates is for reference only. For details on the certificates and for security guidelines for managing the certificates and related files, refer to the database server documentation.

Data Studio can connect to the database using the Secure Sockets Layer [SSL] option. **Adding a Connection** lists the files required.

#	Certificate/Key	Description
1	Client SSL Certificate	Provided by System/Database Administrator
2	Client SSL Key	Provided by System/Database Administrator
3	Root Certificate	Provided by System/Database Administrator

Server Configuration

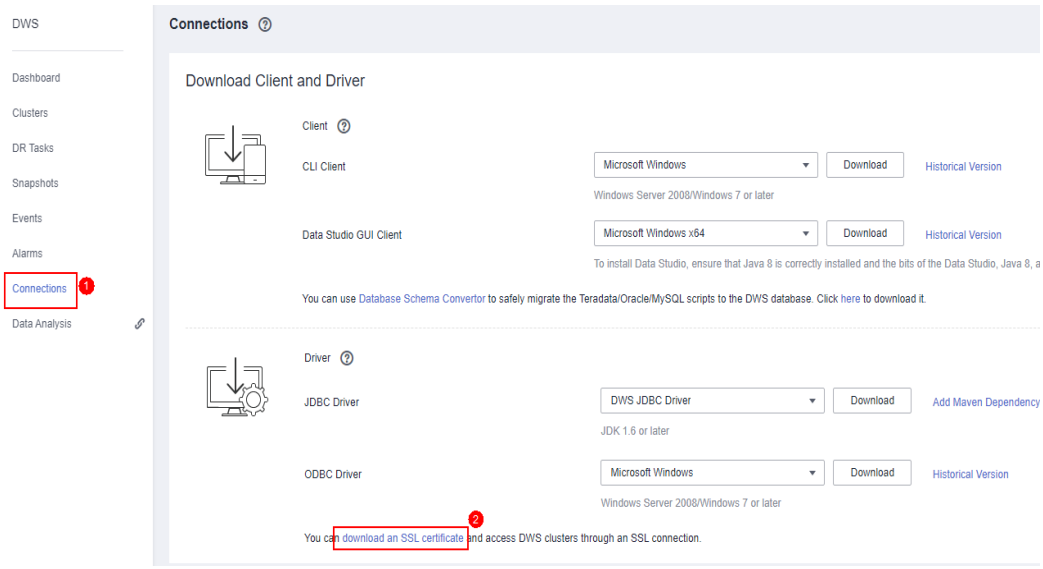
After a GaussDB(DWS) cluster is deployed, the SSL authentication mode is enabled by default. The server certificate, private key, and root certificate have been configured by default.

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) management console. In the navigation pane, choose **Connections**. In the **Driver** area, click **download an SSL certificate**.



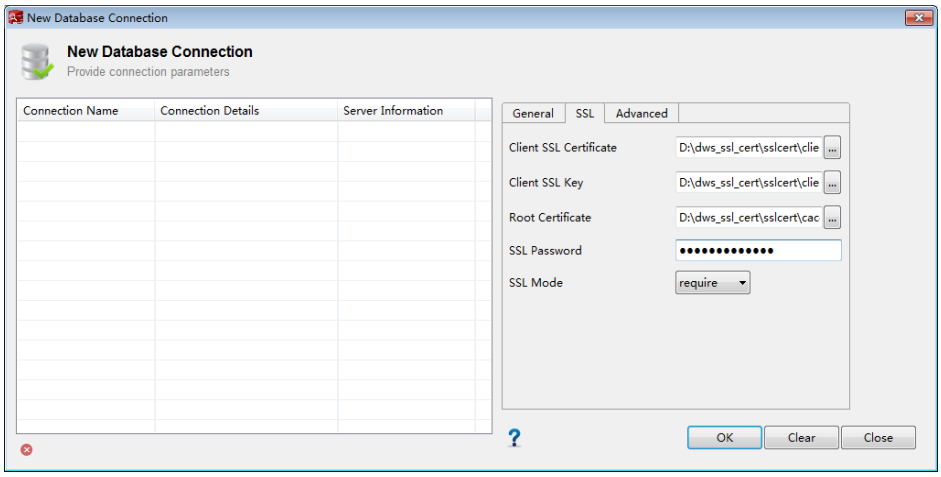
Step 2 Decompress the downloaded **dws_ssl_cert.zip** package to obtain the certificate file. Click the **SSL** tab on the Data Studio client and set the following parameters:

Table 4-30 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	SSL key password in PK8 format on the client.

Parameter	Description
SSL Mode	GaussDB(DWS) supports the following SSL modes: • require • verify-ca GaussDB(DWS) does not support the **verify-full** mode.

Figure 4-1 SSL parameters



----End

4.25 Troubleshooting

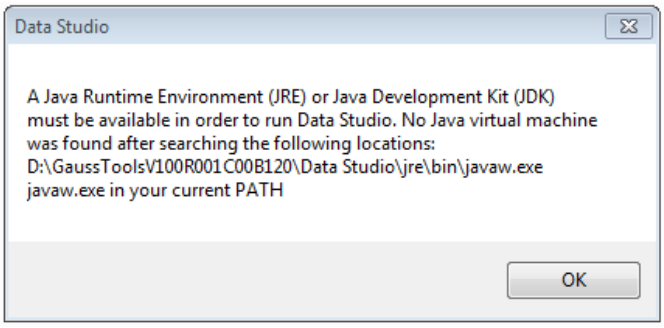
1. **Data Studio does not open.**

   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:

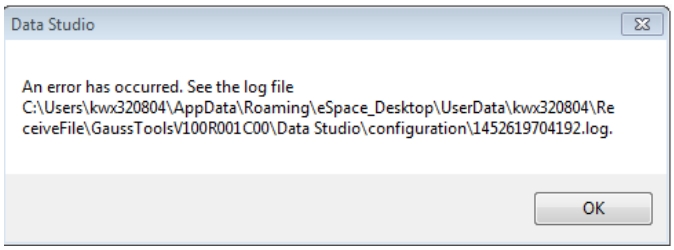
   – Without JRE:

   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 are more than one version of Java installed, set the **-vm** parameter in the configuration file by referring to **Installing and 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 **Quick Start**.

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 trying to run 64-bit Data Studio in the following environment: • 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 the 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 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.

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 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?**

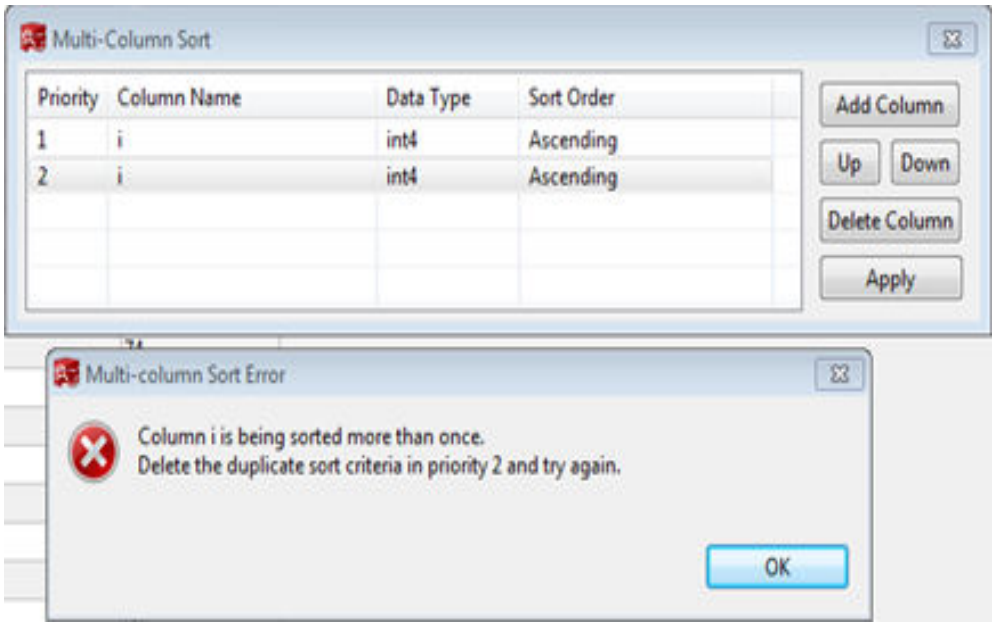
    Solution: This problem occurs if you choose **Preferences** > **Query Results** and set the column headers to be included. The selected cells include the column header cells as well. Modify the settings to disable the Include column headers option and try again.

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**.

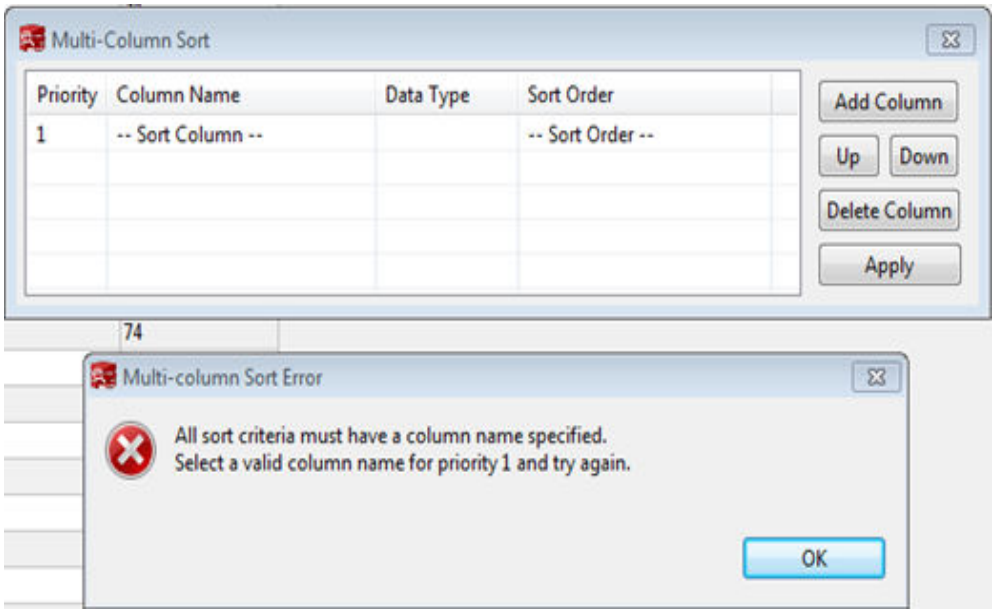
19. **What happens when I add the same column multiple times in a multi-column sort dialog box?**

    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 new security folder and generate a new key.
    d. Enter the password again to log in to Data Studio.

4.26 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. For details, see the server manual.
   - 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 can't Data Studio be started after the -Xmx parameter is modified?**

    Answer: The value of **-Xmx** may be invalid. For details, see **Installing and 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 4-31 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 4-32**.

Table 4-32 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: The additional context menu options like Right to left Reading order, Show Unicode control characters and so on are provided by Windows 7 in case the keyboard you are using supports right to left and left to right 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 resultset 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. **When I query a temp table from a new SQL Terminal the resultset displays incorrect table details. Why does this happen?**

    Answer: When you query a temp table from a new SQL Terminal or with the **Reuse Connection** off, the resultset 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 resultset 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.

5 GDS: Parallel Data Loader

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

1. GDS can be installed in the following OSs:

   Kunpeng platform:

   - Community Enterprise Operating System 7.6
   - EulerOS 2.0 SP8
   - Red Hat Enterprise Linux Server release 7.5
   - NeoKylin 7.5/7.6

   x86:

   - SUSE Linux Enterprise Server 10 SP4 x86_64
   - SUSE Linux Enterprise Server 11 SP1/SP2/SP3/SP4 x86_64
   - SUSE Linux Enterprise Server 12 SP0/SP1/SP2/SP3/SP5 x86_64
   - Red Hat Enterprise Linux Server release 6.4/6.5/6.6/6.7/6.8/6.9/7.0/7.1/7.2/7.3/7.4/7.5 x86_64
   - Community Enterprise Operating System 6.4/6.5/6.6/6.7/6.8/6.9/7.0/7.1/7.2/7.3/7.4 x86_64
   - EulerOS 2.5 x86_64

2. 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 For details about how to import or export data using GDS, see .

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_8.1.*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_8.1.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 8.1.*x*).

```
cd /opt/bin/dws/gds/bin
source gds_env
```

Step 10 Start GDS.

GDS is green software and can be started after being decompressed. There are two ways to start GDS. One is to run the **gds** command to configure startup parameters. The other is to write the startup parameters into the **gds.conf** configuration file and run the **gds_ctl.py** command to start 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 .

- 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 32. If
  without setting, the default value is 1.
   -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	-
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 32 (integer) Default value: **1**

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.fforeign_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.

- -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**

- --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: SQL Syntax Migration Tool

6.1 Overview

After switching to GaussDB(DWS) databases, you may need to migrate user data and application SQL scripts to new databases. In particular,the migration of application SQL scripts is complex, risky, and time-consuming.

Database Schema Convertor (DSC) is a CLI tool running on Linux or Windows. It is designed to provide simple, fast, and reliable migration of application SQL scripts. DSC parses the SQL scripts of source database applications using its syntax migration logic and converts the scripts to the ones 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 objects of Teradata, Oracle, Netezza, MySQL, and DB2:

- Common objects supported by Teradata, Oracle, Netezza, MySQL, and DB2: SQL schemas and SQL queries
- Objects supported only by Oracle and Netezza: PL/SQL objects
- Objects supported only by Teradata: Perl files containing **BTEQ** and **SQL_LANG** scripts

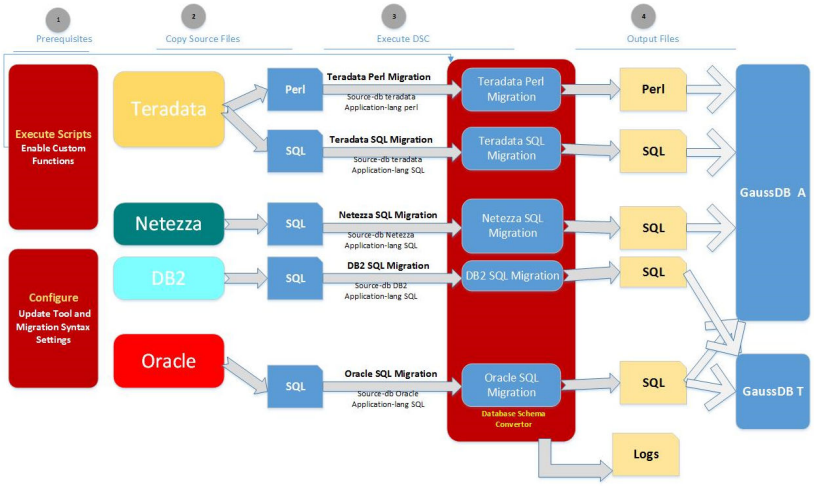
Migration Process

The process of using DSC to migrate SQL scripts is as follows:

1. Export the SQL scripts from a Teradata or Oracle database to the Linux or Windows server installed with DSC.
2. Use DSC to migrate syntax. Specify the paths of the input files, output files, and logs in the command.

3. DSC automatically archives the migrated SQL scripts and the logs into the specified paths.

Figure 6-1 Migration process using DSC



6.2 Supported Keywords and Features

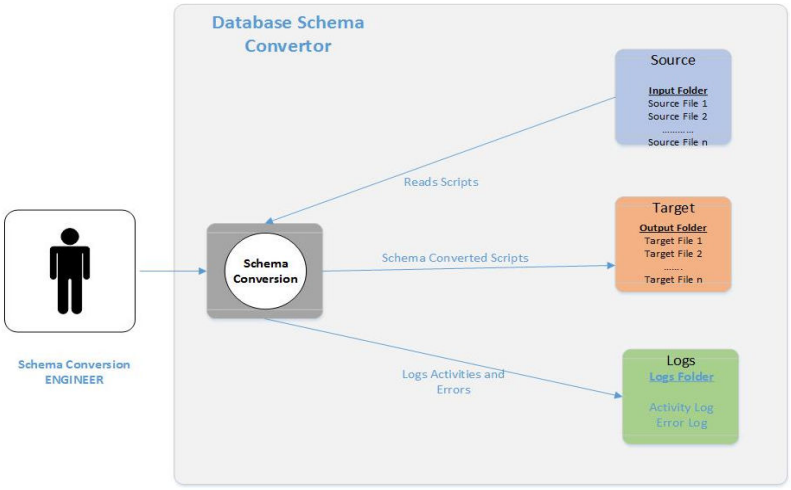
This section provides the system context of DSC. DSC is a CLI tool that helps data migration engineers migrate scripts of source databases, such as Teradata, Oracle, Netezza, MySQL, and DB2, to GaussDB(DWS) databases.

DSC reads the scripts from the input folder. After the migration is complete, the migrated scripts are saved in the output folder. DSC also logs operations and errors to the log folder.

The rules of migrating keywords in SQL scripts are essential to the syntax migration using DSC. This section describes the Teradata, Oracle, Netezza, and MySQL keywords and features supported by DSC, and the parameters for keyword migration rules. Configure these parameters based on source databases, target databases, and migration scenarios.

Keywords that cannot be migrated are recorded in error logs after migration. Error logs also contain details about the files to which the error keywords belong, such as file names and locations. For details, see **Log Reference** and **Troubleshooting**.

Figure 6-2 System context of DSC



6.3 Constraints and Limitations

This section describes the constraints and limitations of DSC.

General

- 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**, errors may occur during script migration.

Teradata

- If a **case** statement containing **FORMAT** is not enclosed in parentheses, this statement will not be processed.

  For example:

  ```
  case when column1='0' then column1='value' end (FORMAT 'YYYYMMDD')as alias1
  ```

  In this example, **case when column1= "0", 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 and 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
                     );
```

Oracle

- **ROWID** is not processed in the following case:

```
SELECT empno,ename ,d.deptno,d.rowid  FROM ( SELECT * FROM employees where deptno is not null) e  , dept d WHERE d.deptno = e.deptno;
```

- The aggregate function in Window functions (RANK and ROW_NUMBER) is not available in the column list.

  For example:

```
SELECT
     TIME, Region, ROW_NUMBER ( ) OVER (ORDER BY SUM (profit) DESC) AS rownumber
      , GROUPING (TIME) AS T
      , GROUPING (Region) AS R
   FROM Sales GROUP BY
      CUBE (TIME, Region)
   ORDER BY
      TIME, Region;
```

- The **INSERT FIRST** and **INSERT ALL** queries cannot be migrated if an asterisk (*) is specified in the **SELECT** statement and the **VALUES** clause is not specified.
- DSC supports the **ROWNUM** condition specified at the end of the **WHERE** clause.

  For example:

```
SELECT ROWNUM, ename,empid
FROM employees
WHERE empid = 10 AND deptno = 10 AND ROWNUM < 3
ORDER BY ename;
```

- DSC does not support the **ROWNUM** function in the **UPDATE** clause.

  For example:

```
UPDATE tableName SET empno = ROWNUM
  where column = "value";
```

- DSC does not support **PARTITION BY LIST** in the **INDEX** clause.

  For example:

```
CREATE TABLE sales(acct_no NUMBER(5)
    ORGANIZATION INDEX
          INCLUDING acct_no
          OVERFLOW TABLESPACE example
    PARTITION BY LIST (acct_no)
         (PARTITION VALUES (1)
```



```
    PARTITION VALUES (DEFAULT)
            TABLESPACE example);
```

- DSC does not support the **JOIN** condition without a table name or table alias.
- Stored procedures and functions must end with a slash (/).

6.4 System Requirements

Supported Databases

Table 6-1 lists the source databases supported by DSC.

Table 6-1 Supported source databases

Database Name	Database Version
Netezza	7.0
Teradata	13.1
Oracle	11g Release 2 12c Release 1
MySQL	5.6

Software Requirements

OS requirements

Table 6-2 lists the operating systems compatible with DSC.

Table 6-2 Compatible OSs

Server	Operating System	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)

Server	Operating System	Version
		6.6-x86_64 (RHEL 6.6)
		6.7-x86_64 (RHEL 6.7)
		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

Other software requirements

Table 6-3 lists the requirements of DSC on other software versions.

Table 6-3 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 or later	Used to migrate Perl files in Windows.

NOTE

Perl 5.8.8 is a fixed version. To ensure syntax compatibility, do not use a later version.

6.5 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 **Compatible OSs**.

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 **System Requirements**.

  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 environment variable in **Path** is *c:|windows|system32;*. Set **Path** to *c:|windows|system32;***C:\Program Files\Java\jdk1.8.0_141\bin;**.

Procedure

DSC is an installation-free tool. You can use it after downloading and decompressing the software package.

The following procedure describes how to obtain the package. Decompress the package **DSC.zip** and then **DSC.rar** to obtain the files shown in **Table 6-4**.

Windows:

Step 1 Log in to the GaussDB(DWS) management console and download the DCS software package. For details, see **Downloading Client Tools**.

Step 2 Decompress the **DSC.zip** package.

The **DSC** folder is generated.

NOTE

You can decompress **DSC.zip** to any folder you need.

Step 3 Go to the **DSC** directory.

Step 4 Find and check the files in the **DSC** directory.

Table 6-4 describes the obtained folders and files.

----End

Linux:

Step 1 Log in to the GaussDB(DWS) management console and download the DCS software package. For details, see **Downloading Client Tools**.

Step 2 Extract files from **DSC.zip**.

```
sh install.sh
```

Step 3 Go to the **DSC** directory.

```
cd DSC
```

Step 4 Check the files in the **DSC** directory.

```
ls
config   lib   scripts   bin   input output runDSC.sh   runDSC.bat
```

----End

Table 6-4 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		Instructions of installation and configuration

NOTE

If you do not need DSC, you can uninstall it by deleting the **DSC** folder.

6.6 Configuring DSC

6.6.1 Overview

DSC provides configuration files and parameters to configure migration logic and rules. Before migrating scripts, you need to configure the following as required:

1. Tool configuration before migration: Configure DSC after its installation. For details, see **DSC Configuration**.
2. Rule configuration before migration: Configure migration rules by modifying configuration files for **Teradata SQL**, **Oracle SQL**, **Teradata Perl**, or **MySQL SQL**.
3. Custom configuration after migration: **Customize rules** in the destination database to migrate the input keywords that are not supported by the destination database.

6.6.2 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 the application parameters that are used to configure the behavior of DSC. They are general parameters and 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 6-5 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 6-5 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

Parameter	Description	Value Range	Default Value	Example
• commentSeparatorFlag	If there are any changes in this configuration item, then the tool will not function as expected.	• true • false	true	commentSeparatorFlag=true
• queryDelimiter	If there are any changes in this configuration item, then the tool will not function as expected.	NA	NA	queryDelimiter=;
• blogicDelimiter	If there are any changes in this configuration item, then the tool will not function as expected.	NA	NA	blogicDelimiter=/
• Timeout	Tool migration timeout duration. If 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 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**: overwrites the output folder. 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

Parameter	Description	Value Range	Default Value	Example
• 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, etc.	Default based on locale.	encodingFormat=UTF8
• NoOfThreads	Number of threads used for migration.	Depending on available system resources	3	NoOfThreads=3

Parameter	Description	Value Range	Default Value	Example
• 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 to 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, process will startup with 256 MB of memory

Parameter	Description	Value Range	Default Value	Example
• maxJVMMemory	Maximum memory	N/A	1024 MB	maxJVMMemory=2048m This indicates, the process will use up to 2048 MB of memory.

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 6-6 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**

6.6.3 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 6-7** as required.

Table 6-7 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	Whether to enable the migration of **3** (grouping sets/cube/rollup). **true**: enables the migration of **GROUP BY()**. **false**: disables the migration of **GROUP BY()**.	• true • false	false	extendedGroupByClause=false
• inToExists	Whether to enable the migration of **IN and NOT IN Conversion** to **EXISTS/NOT EXISTS** for query optimization.	• 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 **creating a 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 start 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 Names 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 **NULLIFZERO()**. **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**: Comment out such views during migration. **false**: Disable the migration of such views. The migration tool copies the query and logs the following information: 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. The value can be **local temporary** or **unlogged**. **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**: Comment out **CHARACTER SET** and **CASESPECIFIC** during script migration. **false**: Disable 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	Migration support for teradata utilities. Possible values - TRUE/FALSE • true • false	• true • false	true	terdataUtilities=true
• unique_primary_index_in_column_table	Creating a unique index for a column store table is not supported.	• true • false	true	unique_primary_index_in_column_table=true
• default_charset	Migration support for default_charset. Possible values: **LATIN/UNICODE/ GRAPHIC**	• LATIN • UNICODE • GRAPHIC	LATIN	default_charset=LATIN
• mergeImplementation	mergeImplementation has the following two types: • using WITH clause • splitting the queries Possible values: • With • Split • None	• With • Split • None	None	mergeImplementation=None
• dsqlSupport	This parameter supports dsql. Possible values: • true • false	• true • false	false	dsqlSupport=false
• tdcolumnInSensitive	Whether to remove column names that contain double quotes during migration. Possible values: • true • false	• true • false	false	tdcolumnInSensitive=false

Parameter	Description	Value Range	Default Value	Example
• 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
• tdMigrateAddMonth	Migration support for addMonth. Possible values: **TRUE** and **FALSE**	• true • false	false	tdMigrateAddMonth=false

6.6.4 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 6-8** as needed.

Table 6-8 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.

6.6.5 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 6-9** as required.

NOTE

- Parameter values are case-insensitive.
- You can modify value of the following two parameters **db-bteq-tag-name** and **db-tdsql-tag-name** parameters in the following table:

Table 6-9 Parameters in the perl-migration.properties file

Parameter	Description	Value Range	Default Value	Example
• db-bteq-tag-name	Scripts to be processed in Perl files. **BTEQ**: Only the scripts under the **BTEQ** tag will be processed.	• bteq	bteq	db-bteq-tag-name=bteq
• db-tdsql-tag-name	Only the scripts under the **db-tdsql-tag-name** tag will be processed. **SQL_LANG:** Only the scripts under the **SQL_LANG** tag will be 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 Value	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**: Delete the intermediate files. **false**: Do not delete the intermediate files.	• true • false	true	remove-intermediate-files=true

Parameter	Description	Value Range	Default Value	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:
	migrate-variables is set to **true** and input is as follows:
	`$V_SQL = "CT X1(C1 INT,C2 CHAR(30))";`
	Output
	`CREATE TABLE X1(C1 INT,C2 CHAR(30));`
	Example 2:
	Input
	`$onesql ="SELECT trim(tablename) from dbc.tables WHERE`
		• true
		• false
			true	migrate-variables=true

Parameter	Description	Value Range	Default Value	Example
	databasename = '${AUTO_DQDB}' and 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 Value	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 Value	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.
	For 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

6.6.6 MySQL SQL 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 6-10 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 (HASH) or replication table (REPLICATION)	• HASH • REPLICATION	• HASH	• table.type=HASH
• table.tablespace	Tablespace options	• COMMENT • RESERVE	• RESERVE	• table.tablespace=RESERVE
• 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=

Parameter	Description	Value Range	Default Value	Example
• 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. 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. Valid values for row-store tables are **YES** and **NO**, and the default is **NO**. Valid values for column-store tables are **YES**, **NO**, **LOW**, **MIDDLE**, and **HIGH**, and the default is **LOW**.	• YES • NO • YES • NO • LOW • MIDDLE • HIGH	• NO • LOW	• table.compress.row=NO • table.compress.column=LOW

Parameter	Description	Value Range	Default Value	Example
• 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

6.6.7 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 6-11** as required.

Table 6-11 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
• 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

6.7 Using DSC

6.7.1 Migration Process

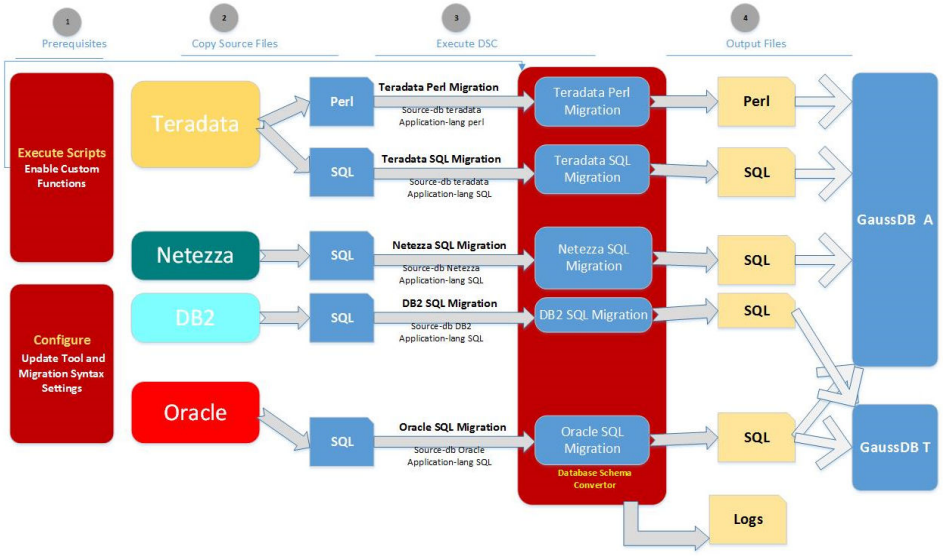
6.7.1.1 Overview

The following use cases for migration are supported by DSC:

- Migrate Teradata SQL
- Migrate Oracle SQL
- Migrate Teradata Perl files
- Migrate Netezza
- Migrate MySQL SQL
- Migrate DB2

Figure 6-3 shows the DSC Migration Process.

Figure 6-3 Syntax migration process



This section contains information about the prerequisites to be completed before starting the migration process.

Executing Custom Scripts

The DSC configuration contains the following custom DB scripts in the *DSC/scripts*:

- *date_functions.sql* : Custom DB script for Oracle date functions
- *environment_functions.sql*: Custom DB script for Oracle environment functions
- *string_functions.sql*: Custom DB script for Oracle string functions
- *pkg_variable_scripts.sql*: Custom DB script for Oracle package variable functions
- *sequence_scripts.sql*: Custom DB script for Oracle sequence functions
- *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

These DB scripts are required to support certain input keywords not present in one or more versions of the target DB. These scripts need to be executed once in the target DB prior to migration.

For details about executing custom database scripts, see **Custom DB Script Configuration**.

Use any of the following methods to execute the required scripts in all target GaussDB(DWS) databases for which migration is to be performed:

- Run the following command to connect to the GaussDB(DWS) database and paste all content in the .sql file to **gsql**, which will automatically execute the pasted content.

  Run the following command to connect to the GaussDB(DWS) 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 GaussDB(DWS) database and execute the **.sql** file:

  Run the following command to connect to the GaussDB(DWS) database and execute 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 to connect to the GaussDB(DWS) database, and open and execute the **.sql** file in Data Studio.
- Before Oracle PL/SQL objects (procedures or functions) are migrated, migrate all DDL and DML using the **Bulk** migration type. Then, migrate the scripts containing PL/SQL objects using the **BLogic** migration type.

  NOTE

  If the migration type is **Bulk**, the input file cannot contain any PL/SQL objects.

  Similarly, if the migration type is **BLogic**, the input files must not contain any DDL/DML.

Configuring DSC and Migration Properties

The DSC configuration contains the following configuration files in the *DSC/config* folder:

- *application.properties*: Configuration parameters for the DSC
- *features-teradata.properties*: Configuration parameters for Teradata SQL Migration
- *features-oracle.properties*: Configuration parameters for Oracle SQL Migration
- *oracle-migration.properties*: Configuration parameters for Oracle (Beta) SQL Migration
- *perl-migration.properties*: Configuration parameters for Perl Migration
- *features-netezza.properties*: Configuration parameters for Netezza Migration
- *features-mysql.properties*: Configuration parameters for MySQL SQL Migration

For details about how to update configuration parameters, see **DSC Configuration**.

6.7.1.2 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 6-12 describes the custom scripts in the **DSC/scripts/** directory. For details about how to execute custom scripts, see **Custom DB Script Configuration**.

Table 6-12 Custom DB scripts

Script	Description
date_functions.sql	Custom DB script for Oracle date functions
environment_functions.sql	Custom DB script for Oracle environment functions
string_functions.sql	Custom DB script for Oracle string functions.
pkg_variable_scripts.sql	Custom DB script for Oracle package variable functions
sequence_scripts.sql	Custom DB script for Oracle sequence functions
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

Table 6-13 Custom DB scripts (Oracle to GaussDB T)

Script	Description
create_user_and_temptable_enable.sql	Custom DB script for create user and local temporary table is used to address Oracle Package feature.
pkg_variable_scripts.sql	Custom DB script for Oracle package variable functions

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 6-14** lists the folders and files in the **scripts** folder.

The SQL files contain the scripts for the custom migration functions. These functions are required to support the specific features of Teradata and Oracle.

Table 6-14 Custom DB scripts for DSC

Folder	Script File	Description
-- scripts	-	Folder: all scripts
------ oracle	-	Folder: Oracle functions and scripts
-------- sequence	-	Folder: scripts to configure Oracle sequences
-	sequence_scripts.sql	Script: used to enable migration of Oracle **sequence**
-------- package	-	Folder: scripts to configure Oracle package variables
-	pkg_variable_scripts.sql	Script: used to enable migration of Oracle installation **package variables**
-------- function	-	Folder: scripts to configure Oracle system functions

Folder	Script File	Description
-	date_functions.sql	Script: used to enable migration of Oracle date functions
-	environment_functions.sql	Script: used to enable migration of Oracle environment functions
-	string_functions.sql	Script: used to enable migration of Oracle string functions
------ 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
-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 6-15** describes the parameters.

Table 6-15 Parameters for configuring DSC

Scenario	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
Oracle SQL Migration	• DSC: *application.properties* • **Oracle SQL Configuration**: *features-oracle.properties*	exceptionHandler=True/**False** TxHandler=**True**/False foreignKeyHandler=**True**/False globalTempTable=**GLOBAL**/LOCAL onCommitDeleteRows=Delete/**Preserve** maxValInSequence=0..**9223372036854775807** mergeImplementation=**WITH**/SPLIT RemoveHashPartition=**True**/False RemoveHashSubPartition=**True**/False RemoveListPartition=**True**/False RemoveListSubPartition=**True**/False RemoveRangeSubPartition=**True**/False MigSupportSequence=**True**/False
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
MySQL SQL Migration	• DSC: *application.properties* • **MySQL SQL Configuration**: *features-mysql.properties*	table.databaseAsSchema=true table.defaultSchema=public table.schema= table.orientation=ROW table.type=HASH table.partition-key.choose.strategy=com.huawei.hwclouds.scs.dws.DWSPartitionKey-ChooserStrategy table.partition-key.name= table.compress.mode=NOCOMPRESS table.compress.level=0 table.compress.row=NO table.compress.column=LOW table.database.template=template0

Scenario	Configuration File	Parameter
Netezza SQL Migration	• DSC: *application.properties* • Netezza Configuration: *features-netezza.properties*	rowstoreToColumnstore=false
DB2 Syntax Migration	DSC: *application.properties*	exceptionHandler=True/**False** TxHandler=**True**/False foreignKeyHandler=**True**/False globalTempTable=**GLOBAL**/LOCAL onCommitDeleteRows=Delete/**Preserve** maxValInSequence=0..**9223372036854775807** mergeImplementation=**WITH**/SPLIT RemoveHashPartition=**True**/False RemoveHashSubPartition=**True**/False RemoveListPartition=**True**/False RemoveListSubPartition=**True**/False RemoveRangeSubPartition=**True**/False MigSupportSequence=**True**/False

6.7.1.3 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

6.7.1.4 Migrating Data Using DSC

Precautions

- 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 or files cannot be restored using 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**.

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 6-16**.

Table 6-16 Migration on Windows and Linux

Scenario	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]
Oracle SQL Migration	**./runDSC.sh** **--source-db** Oracle [**--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** Oracle [**--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**]
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**]

Scenario	CLI Parameter
Netezza SQL Migration	> **./runDSC.sh** **--source-db** Netezza **[--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** Netezza **[--application-lang** SQL] **[--input-folder** <input-script-path>] **[--output-folder** <output-script-path>] [**--log-folder** <log-path>] [**--target-db/-T**][Optional]
DB2 Syntax Migration	> **./runDSC.sh** **--source-db** db2 **[--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** db2 **[--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** or **Oracle**, 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. **BLogic** is applicable to Oracle PL/SQL and Netezza.
  - **target-db** specifies the target database. The 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.
- For details about CLI parameters, see **Database Schema Conversion**.

Task Example

- Example 1: Run the following command to migrate the SQL file of the Oracle database to the SQL script of GaussDB(DWS) on Linux:

  ./runDSC.sh --source-db *Oracle* **--input-folder** D:\test\conversion\input **--output-folder** *D:|test|conversion|output* **--log-folder** *D:|test|conversion|log* **--conversion-type** *bulk* **--target-db** gaussdbA

- 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 *Oracle* **--input-folder** D:\test\conversion\input **--output-folder** *D:|test|conversion|output* **--log-folder** *D:|test|conversion|log* **--conversion-type** *bulk* **--target-db** gaussdbA

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 **********
```

6.7.1.5 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 they do not, contact technical engineer 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**.

6.7.1.6 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 details, see **Troubleshooting** and **FAQs**.

- Migration syntax issues: DSC did not correctly recognize or migrate the migration syntax. For details, see **Constraints and Limitations**.

6.7.2 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 path, 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 are written into **log files**.

```
********** 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 **Migrating Data Using DSC**.

NOTE

During the migration, the metadata of the input script can be called and is stored in the following files:

- Teradata migration

  teradata-set-table.properties
- Oracle migration
  1. global-temp-table.properties
  2. global-temp-tables.properties
  3. primary-key-constraints.properties
  4. package-definition.properties
  5. package-names-oracle.properties
  6. create-types-UDT.properties

Clear the preceding files in the following scenarios:

- Migration of different files
- Migration of the same file with different parameter settings

6.7.3 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, which include the folder paths in the example and the **conversion-type** set to **Bulk**:

  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 are written into **log files**.

```
********** 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, which includes the folder paths in the example and the **conversion-type** set to **BLogic**:

  ```
  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 are written into **log files**.

  ```
  ********** 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 **Migrating Data Using DSC**.

6.7.4 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. Perform the operations in **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. The DSC invokes the Teradata SQL to migrate the extracted SQL scripts. For details about Teradata SQL migration, see **Teradata Syntax 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

6.7.5 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 are written into **log files**.

```
********** 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 **Migrating Data Using DSC**.

6.7.6 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 can be SQL or Perl. The default language is SQL. The migration type can be **Bulk** or **BLogic**.

```
./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.

```
********** 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.

```
********** 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 ************************
```

- Run the following commands to migrate objects, such as functions, store procedures, and packages that contain PL/SQL statements:

  Linux:

  ```
  ./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/
  ```

  Windows:

  ```
  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
  ```

  During the execution of DSC, the migration summary, including the progress and completion status, is displayed on the console. Execution information and errors are written into **log files**.

  ```
  ********************* 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 ************************
  ```

6.7.7 SQL Formatter

SQL Formatter improves the readability of SQL statements. It formats SQL statements by adding/deleting rows and indenting context, and is used to format migrated output files. It can also be used for importing files.

Use the **formattedSourceRequired** parameter to enable/disable the SQL formatter for the source SQL files. If this parameter is set to **true**, the copy of the input file is formatted and saved to the *Output path***formattedSource** directory.

SQL Formatter also supports Teradata SQL migration and Oracle SQL migration. SQL scripts in Teradata Perl file migration are also formatted. The Oracle (Beta) tool does not support SQL Formatter.

Input: SQL FORMATTER

```
select
p1.parti_encode ,p1.accting_type_cd ,p1.prod_code ,p1.cust_type_cd
,p1.accting_amt_type_cd ,p1.accting_num_1 ,p1.accting_num_2 ,p1.accti
ng_num_3 ,p1.accting_num_4 ,p1.accting_num_5 ,p1.accting_num_6
,p1.start_dt ,p1.pre_effect_debit_gl_num ,p1.pre_effect_crdt_gl_num ,p
1.after_effect_debit_gl_num ,p1.after_effect_crdt_gl_num ,coalesce( p2.start_dt ,cast( '3
0001231' as date format 'yyyymmdd' ) ) ,p1.accting_term ,p1.etl_job
from ( select rank ( start_dt
asc ) as
start_dt_id ,parti_encode ,accting_type_cd ,prod_c
ode ,cust_type_cd ,accting_amt_type_cd ,accting_
num_1 ,accting_num_2 ,accting_num_3 ,accting_n
um_4 ,accting_num_5 ,accting_num_6 ,start_dt
,pre_effect_debit_gl_num ,pre_effect_crdt_gl_num ,after
```



```
_effect_debit_gl_num ,after_effect_crdt_gl_num ,accting_term
,etl_job from ccting_subj_para_h_mf0_a_cur_i )
p1 left join ( select rank ( start_dt
asc ) - 1 as
start_dt_id ,parti_encode ,accting_type_cd ,prod_cod
e ,cust_type_cd ,accting_amt_type_cd ,accting_nu
m_1 ,accting_num_2 ,accting_num_3 ,accting_num_4
,accting_num_5 ,accting_num_6 ,start_dt
,accting_term from
ccting_subj_para_h_mf0_a_cur_i ) p2 on p1.start_dt_id = p2.start_dt_id
and p1.parti_encode = p2.parti_encode and p1.accting_type_cd = p2.accting_type_cd
and p1.prod_code = p2.prod_code and p1.cust_type_cd = p2.cust_type_cd and
p1.accting_amt_type_cd = p2.accting_amt_type_cd and p1.accting_num_1 =
p2.accting_num_1 and p1.accting_num_2 = p2.accting_num_2 and p1.accting_num_3
= p2.accting_num_3 and p1.accting_num_4 = p2.accting_num_4 and
p1.accting_num_5 = p2.accting_num_5 and p1.accting_num_6 = p2.accting_num_6
and p1.accting_term = p2.accting_term ;
```

Output

```
SELECT
    p1.parti_encode ,p1.accting_type_cd
    ,p1.prod_code
    ,p1.cust_type_cd ,p1.accting_amt_type_cd
    ,p1.accting_num_1
    ,p1.accting_num_2
    ,p1.accting_num_3
    ,p1.accting_num_4
    ,p1.accting_num_5
    ,p1.accting_num_6
    ,p1.start_dt
    ,p1.pre_effect_debit_gl_num
    ,p1.pre_effect_crdt_gl_num
    ,p1.after_effect_debit_gl_num
    ,p1.after_effect_crdt_gl_num
    ,COALESCE( p2.start_dt ,CAST( '30001231' AS DATE ) )
    ,p1.accting_term ,p1.etl_job
FROM
    (
        SELECT
            rank (
            ) over( ORDER BY start_dt ASC ) AS start_dt_id
            ,parti_encode
            ,accting_type_cd
            ,prod_code
            ,cust_type_cd
            ,accting_amt_type_cd
            ,accting_num_1
            ,accting_num_2
            ,accting_num_3
            ,accting_num_4
            ,accting_num_5
            ,accting_num_6
            ,start_dt ,pre_effect_debit_gl_num ,pre_effect_c
rdt_gl_num
            ,after_effect_debit_gl_num
            ,after_effect_crdt_gl_num
            ,accting_term ,etl_job FROM
            ccting_subj_para_h_mf0_a_cur_i
    ) p1 LEFT JOIN (
        SELECT
            rank (
            ) over( ORDER BY start_dt ASC ) - 1 AS start_dt_id
            ,parti_encode
            ,accting_type_cd
            ,prod_code
            ,cust_type_cd
            ,accting_amt_type_cd
            ,accting_num_1
            ,accting_num_2
```



```
                ,accting_num_3
                ,accting_num_4
                ,accting_num_5
                ,accting_num_6
                ,start_dt
                ,accting_term
            FROM
                ccting_subj_para_h_mf0_a_cur_i
    ) p2
        ON p1.start_dt_id = p2.start_dt_id
    AND p1.parti_encode = p2.parti_encode
    AND p1.accting_type_cd = p2.accting_type_cd
    AND p1.prod_code = p2.prod_code
    AND p1.cust_type_cd = p2.cust_type_cd
    AND p1.accting_amt_type_cd = p2.accting_amt_type_cd
    AND p1.accting_num_1 = p2.accting_num_1
    AND p1.accting_num_2 = p2.accting_num_2
    AND p1.accting_num_3 = p2.accting_num_3
    AND p1.accting_num_4 = p2.accting_num_4
    AND p1.accting_num_5 = p2.accting_num_5
    AND p1.accting_num_6 = p2.accting_num_6
    AND p1.accting_term = p2.accting_term
;
```

6.8 Teradata Syntax Migration

6.8.1 Overview

This section lists the Teradata features supported by the syntax migration tool, and provides the Teradata syntax and the equivalent GaussDB syntax for each feature. The syntax listed in this section illustrates the internal migration logic for Teradata script migration.

This section is also a reference for the database migration team and for the on-site verification of Teradata script migration.

6.8.2 Schema Objects

This section describes the syntax for migrating Teradata schema objects. The migration syntax determines how the keywords and features are migrated.

The schema refers to the data structure of a database. DSC facilitates schema migration from Teradata to GaussDB(DWS).

This section includes the following sub-chapters:

Table Migration, Index Migration, View Migration, COLLECT STATISTICS, ACCESS LOCK, DBC.COLUMNS, DBC.TABLES, DBC.INDICES, and SHOW STATS VALUES SEQUENCED. For details, see **Table Migration** to **SHOW STATS VALUES SEQUENCED**.

6.8.2.1 Table Migration

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.

For details, see the following topics:

CREATE TALBE

CHARACTER SET and CASESPECIFIC

VOLATILE

SET

MULTISET

TITLE

INDEX

CONSTRAINT

COLUMN STORE

PARTITION

ANALYZE

Data Types

Support for Specified Columns

CREATE TALBE

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 using CREATE *tab2* AS *tab1*, a new table *tab2* is created with the structure copied from *tab1*. If the CREATE TABLE statement includes WITH DATA operator, then the data from *tab1* is also copied into *tab2*. When using CREATE AS, the behavior of the CONSTRAINT from the source table is retained in the new target table.

- If **session_mode** = *Teradata*, the default table type is **SET** in which duplicate records must be removed. This is done by adding the **MINUS** operator in the migrated scripts.
- If **session_mode** = *ANSI*, the default table type is **MULTISET** in which duplicate records must be allowed.

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;
```

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), string data types 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)
        )
```

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
```



```
;
/
```

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 ;
```

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
      )
;
```

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
;
```


1. 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;
```

2. 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};
```

INDEX

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 )
  )
;
```

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.

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);
```

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** or **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 another 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 create table script with 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 )
                ) */
;
```

ANALYZE

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);
```

Data Types

The following data type mappings are supported by the DSC.

Input	Output
Numeric	**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	**Character**
CHAR[(n)] / CHARACTER [(n)]	CHAR(n)
CLOB	CLOB
LONG VARCHAR	TEXT

Input	Output
VARCHAR(n) / CHAR VARYING(n) / CHARACTER VARYING(n)	VARCHAR(n)
DateTime	**DateTime**
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
Period	**Period**
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	**Binary**
BLOB[(n)]	blob
BYTE[(n)]	bytea
VARBYTE[(n)]	bytea

For example: BYTEINT

Input

```
select cast(col as byteint) from tab;
```

Output

```
SELECT CAST( col AS SMALLINT ) FROM tab ;
```

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
;
```

6.8.2.2 Index Migration

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 )
  )
;
```

6.8.2.3 View Migration

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 **tdMigrateVIEWCHECKOPTION** 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 **tdMigrateVIEWCHECKOPTION** configuration parameter to configure migration of views containing the **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 6-4 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 6-5 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 ) ;
```

6.8.2.4 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 the **COLLECT STAT** statement.

For details, see **1**.

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 ;
```

6.8.2.5 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 ...;
```

6.8.2.6 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 **System Requirements** and **Prerequisites** sections respectively.

The DSC migrates the following dbc.columns to their corresponding information_schema columns.

Table 6-17 Migration of dbc.columns to information_schema columns

dbc.columns	information_schema.columns
ColumnName	Column_Name

dbc.columns	information_schema.columns
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
;
```

6.8.2.7 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 > ] ) ) )
;
```

6.8.2.8 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();
```

6.8.3 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"*/
```

6.8.4 DML

This section describes the syntax for migrating Teradata DML. The migration syntax determines how the keywords and features are migrated.

In Teradata, SQL queries in a file that contains the **SELECT**, **INSERT**, **UPDATE**, **DELETE**, or **MERGE** statement can be migrated to GaussDB(DWS).

For details, see the following topics:

INSERT

SELECT

UPDATE

DELETE

MERGE

NAMED

ACTIVITYCOUNT

TIMESTAMP

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 exists in Teradata SQL, but is not supported by GaussDB(DWS). GaussDB(DWS) supports only the **INSERT INTO table_name** syntax. 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;
```

SELECT

1. **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;
```

2. **Order of Clausses**

   For Teradata migration of **SELECT** statements, all the clauses (**FROM**, **WHERE**, **HAVING** and **GROUP BY**) can be listed in any order. The tool will not migrate the statement if it contains **a QUALIFY** as an **ALIAS** before the **FROM** clause.

   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 expr1 <> 'IC'
       AND expr2 <> 'IC'
       AND expr3 <> '';
```

   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, the 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
        )
;
```

3. **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 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 expr1 <> 'IC'
       AND expr2 <> 'IC'
       AND expr3 <> '';
```

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
      );
```

4. **TOP and SAMPLE**

   The **TOP** and **SAMPLE** clauses of Teradata are migrated to **LIMIT** in GaussDB(DWS).

   a. TOP

      The DSC also supports migration of **TOP** statements with dynamic parameters.

      NOTE

      - For **TOP** clauses containing **WITH TIES**, the ORDER BY clause is also required. Otherwise, the tool will not migrate the statement and copy it 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 valid for dynamic parameters: a-z, A-Z, 0-9 and "_".

      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 with 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
;
```

b. SAMPLE

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;
```

UPDATE

The tool supports and migrates 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;
```

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;
```

MERGE

NOTE

Gauss database in 6.5.0 or later versions support the MERGE function.

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
                )
;
```

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);
```

ACTIVITYCOUNT

Input

It's 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 ;
/
```

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 Types Casting

Input

```
COALESCE( a.Snd_Tm ,TIMESTAMP '0001-01-01 00:00:00' )
should be migrated as below:
COALESCE( a.Snd_Tm , CAST('0001-01-01 00:00:00' AS TIMESTAMP) )
```

Output

```
COALESCE( a.Snd_Tm , CAST('0001-01-01 00:00:00' AS TIMESTAMP) )
```

6.8.5 Querying Migration Operators

This section contains the migration syntax for migrating Teradata query migration operators. The migration syntax determines how the keywords and features are migrated.

For details, see the following topics:

QUALIFY

ALIAS

FORMAT and CAST

Short Keys Migration

Object Names Starting with $

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
;
```

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.

Tools support ALIAS names for columns. If the ALIAS name is same as the column name, then it should be specified for that column only and not for other columns in that table. In the following example, there is a conflict between DATA_DT column name and the DATA_DT alias. This 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
;
```

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 Gauss 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;
```

Short Keys Migration

Table 6-18 lists the Teradata short keys supported by GaussDB(DWS) and their equivalent syntax in GaussDB(DWS).

Table 6-18 List of short keys

Teradata Short Key	Equivalent Syntax in GaussDB(DWS)
SEL	SELECT
INS	INSERT
UPD	UPDATE

Teradata Short Key	Equivalent Syntax in GaussDB(DWS)
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;
```

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 / NOT IN conversion**

Table 6-19 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
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 to TRUE)

```
SELECT
	"$C1"
FROM
	p11
WHERE
	"$C1" NOT LIKE ANY (
	ARRAY[ "$sql1" ]
	)
;
```

Output (tdMigrateDollar to 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 ;
```

6.8.6 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**.

IN and NOT IN Conversion

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 a **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;
```

6.8.7 System Functions and Operators

This section contains the migration syntax for migrating Teradata system functions and operators. The migration syntax determines how the keywords and features are migrated.

Schema

The database with the schema name should be changed to **SET SESSION CURRENT_SCHEMA**.

Oracle Syntax	Syntax After Migration
DATABASE SCHTERA	SET SESSION CURRENT_SCHEMA TO SCHTERA

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.

1. **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;
```

2. **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
;
```

3. **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 occurrance of window function in QUALIFY clause. Multiple window functions in a QUALIFY may result in invalid migration.

4. **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;
```

5. **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;
```

6. **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 ...;
```

7. **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
;
```

8. **COMPRESS specified with *****

   Input

```
ORDCADBRN VARCHAR(6) CHARACTER SET LATIN CASESPECIFIC TITLE '    ' COMPRESS '******'
```

   Output

```
ORDCADBRN VARCHAR( 6 ) /* CHARACTER SET LATIN*/ /* CASESPECIFIC*/ /*TITLE '    '*/ /*
COMPRESS  '******' */
```


Comparison and List Operators

NOTE

The comparison operators LT, LE, GT, GE, EQ, and NE must not be used as TABLE alias or COLUMN alias.

The following comparison and list operators are supported:

1. **^= 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;
```

2. **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;
```

3. **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;
```

4. **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;
```

5. **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.

6. **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);
   ```

7. **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%' ]);
   ```

8. **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%' ]);
   ```

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);
```


6.8.8 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' ;
```

6.8.9 String Functions

CHAR Function

Input: CHAR

```
CHAR( expression1 )
```

Output

```
LENGTH( expression1 )
```

CHARACTERS Function

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 ... ;
```

6.8.10 Date and Time Functions

DATE

Migration tools support the migration of Teradata DATE FORMAT in SELECT statements, 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
;
```

Migration tools support 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

```
ADD_MONTHS(CAST(SUBSTR( T1.GRANT_DATE ,1 ,8 ) || '01' AS DATE ),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);
```

6.8.11 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**

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;
```

Expressions

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
;
```

6.8.12 BTEQ Utility Command

The BTEQ utility commands are as follows:

LOGOFF QUIT

LOGOFF ends the current RDBMS sessions without exiting BTEQ.

QUIT command ends the current Teradata database sessions and exits BTEQ.

Input

```
SELECT 'StartDTTM' as a
        ,CURRENT_TIMESTAMP (FORMAT 'HH:MI:SSBMMMBDD,BYYYY') ;
.LOGOFF;
.QUIT;
```

Output

```
SELECT 'StartDTTM' as a
        ,CURRENT_TIMESTAMP (FORMAT 'HH:MI:SSBMMMBDD,BYYYY') ;
.LOGOFF;
.QUIT;
```

NOTE

Gauss does not support, so it is required to comment.

.IF ERRORCODE and .QUIT

BTEQ statements specified with .IF and .QUIT.

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 ;
```

.IF

Input:

```
.IF END_MONTH_FLAG <> 'Y' THEN .GOTO LABEL_1;
```

Output:

```
IF END_MONTH_FLAG <> 'Y' THEN
            GOTO LABEL_1 ;
END IF ;
```

.QUIT

Input:

```
.IF ERRORCODE <> 0 THEN .QUIT 12;
```

Output:

```
IF ERRORCODE <> 0 THEN
        RAISE EXCEPTION '12' ;
END IF;
```

.IF & .QUIT

.IF and .QUIT should be moved inside the block.

Oracle Syntax	Syntax After Migration

INSERT INTO EMP
SELECT * FROM EMP;
.IF ERRORCODE <> 0 THEN .QUIT 12;
	DECLARE
	lv_mig_errorcode NUMBER (4) ;
	BEGIN
	BEGIN
	INSERT INTO EMP
	SELECT * FROM EMP;
	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 ;
	/

.RETURN

Input:

```
.if ERRORCODE = 0 then .RETURN;
```

Output:


```
IF ERRORCODE = 0 THEN
        RETURN;
END IF;
```

.GOTO

Input:

```
.IF END_MONTH_FLAG <> 'Y' THEN .GOTO LABEL_1;
```

Output:

```
IF END_MONTH_FLAG <> 'Y' THEN
     GOTO LABEL_1;
END IF ;
```

Label

Input:

```
.LABEL LABEL_1
```

Output:

```
<<LABEL_1>>
```

ERRORCODE 3807

Input:

```
SELECT end_mon AS END_MONTH_FLAG
  FROM tab2 ;

.IF END_MONTH_FLAG <> 'Y' THEN
       .GOTO LABEL_1;

.IF ERRORCODE = 3807 THEN
      .QUIT 8888;
```

Output

```
DECLARE lv_mig_errorcode NUMBER (4);
     lv_mig_END_MONTH_FLAG TEXT;
BEGIN
    BEGIN
        SELECT end_mon
          INTO lv_mig_END_MONTH_FLAG
          FROM tab2 ;

         lv_mig_errorcode := 0 ;
    EXCEPTION
         WHEN UNDEFINED_TABLE THEN
              lv_mig_errorcode := 3807 ;

        WHEN OTHERS THEN
            lv_mig_errorcode := - 1 ;
    END ;

    IF lv_mig_END_MONTH_FLAG <> 'Y' THEN
            GOTO LABEL_1 ;
    END IF ;
    IF lv_mig_errorcode = 3807 THEN
        RAISE EXCEPTION '8888' ;
    END IF ;
END;
/
```


BT with BTEQ commands

Input

```
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;
```

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;
```

ET with BTEQ Commands

Input

```
ET;
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;
```

Output

```
--
  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;
```

.Export FILE

.EXPORT FILE should be changed to COPY and teradata utilities is set to **false**.

Oracle Syntax	Syntax After Migration
.export file = $FILENAME; select empno, ename from emp where deptno = 1;	COPY (select empno, ename from emp where deptno = 1) TO '$FILENAME' CSV HEADER DELIMITER E'\t';

.EXPORT FILE should be changed to qCOPY and moved inside the block when teradataUtilities is set to **true**.

Oracle Syntax	Syntax After Migration

print BTEQ <<ENDOFINPUT;
  DATABASE HPBUS;
.export file = $FILENAME;
select empno, ename from emp where deptno = 1;
.IF ERRORCODE <> 0 THEN .QUIT 12;
.LOGOFF;
.QUIT 0;
ENDOFINPUT
	DECLARE lv_mig_obj_exists_check NUMBER(1);
	 lv_mig_errorcode NUMBER(4);
	BEGIN
	 SET SESSION CURRENT_SCHEMA TO public ;
	
	  BEGIN
	 SELECT COUNT(*) INTO lv_mig_obj_exists_check
	  FROM (select empno, ename from emp where deptno = 1);
	  lv_mig_errorcode := 0;
	
	  EXCEPTION
	  WHEN OTHERS THEN
	  lv_mig_errorcode := -1;
	  END ;
	 EXECUTE qCOPY (select empno, ename from emp where deptno = 1) TO '$FILENAME' CSV HEADER DELIMITER E'\t'q;
	
	  IF lv_mig_errorcode <> 0 THEN
	  RAISE EXCEPTION '12';
	  END IF ;
	 RAISE EXCEPTION '-99';
	  RETURN ;
	END ;
	/

SQL_Lang & BTEQ

Multiple tag names should be handled for SQL_Lang.

Oracle Syntax	Syntax After Migration

#!/usr/bin/perl
#####################################
#####################################
# BTEQ script in Perl 	
# Date Time : 2013-10-15 	
# Table : TB_NET_LTE_GPRS_CDR (4G GPRS» 	
°µ¥)
# Script Name : XXX00_TB_NET_LTE_GPRS_CDR 	
# Source Table : TB_04024-GPRS 	
# Load strategy: S4 	
use strict; # Declare using Perl strict syntax
 	
my $HOME = $ENV{"AUTO_HOME"};
unshift(@INC, "$HOME/bin");
require etl_pub;
 	
# ------------ Variable Section ------------ 	
# DATABASE NAMES 	
my $TEMPDB = $ETL::TEMPDB;
my $SDATADB = $ETL::SDATADB;
my $PDATADB = $ETL::PDATADB;
my $PARADB = $ETL::PARADB;
my $PDDL = $ETL::PDDL;
 	
my $PCODE = $ETL::PCODE;
if ($#ARGV < 0) {
exit(1);
}
my $CONTROL_FILE = $ARGV[0];
my $TX_DATE = substr(${CONTROL_FILE},length($
{CONTROL_FILE})-12, 8);
 	
open(STDERR, ">&STDOUT");
my $TRG_COL_LIST =<<TRGCOLLIST;
MSISDN
,dat_rcd_dt
,vst_rgn_cd
 	
TRGCOLLIST
 	
my $MAP_COL_LIST =<<MAPCOLLIST;
TB_04024.msisdn
,CAST('$TX_DATE' AS DATE FORMAT
'YYYYMMDD')
,TB_04024.vst_rgn_cd
MAPCOLLIST
 	
my $FILTER = "CMCC_Prov_Prvd_Id=$PCODE";
my $table_today ="${TEMPDB}.TB_04024_$
{PCODE}";
my $table_target = "Z" . substr($PCODE,1,2) .
"NET_LTE_GPRS_CDR";
my $UNIT_FLAG;
 	
sub BTEQ_S4
{
my ($dbh) = @_;
my $BTEQ_CMD_S4 = <<ENDOFINPUT;
 	
INSERT INTO ${PDATADB}.$table_target (
$TRG_COL_LIST)
SELECT
$MAP_COL_LIST
FROM $SDATADB.TB_${PCODE}_04024_$
{UNIT_FLAG}_${TX_DATE} TB_04024;
.IF ERRORCODE <> 0 THEN .QUIT 12;
	#!/usr/bin/perl
	#####################################
	#####################################
	# BTEQ script in Perl
	# Date Time : 2013-10-15
	# Table : TB_NET_LTE_GPRS_CDR (4G GPRS)
	# Script Name : XXX00_TB_NET_LTE_GPRS_CDR
	# Source Table : TB_04024-GPRS
	# Load strategy: S4
	use strict; # Declare using Perl strict syntax
	
	my $HOME = $ENV{"AUTO_HOME"};
	unshift(@INC, "$HOME/bin");
	require etl_pub;
	
	# ------------ Variable Section ------------
	# DATABASE NAMES
	my $TEMPDB = $ETL::TEMPDB;
	my $SDATADB = $ETL::SDATADB;
	my $PDATADB = $ETL::PDATADB;
	my $PARADB = $ETL::PARADB;
	my $PDDL = $ETL::PDDL;
	
	my $PCODE = $ETL::PCODE;
	if ($#ARGV < 0) {
	exit(1);
	}
	my $CONTROL_FILE = $ARGV[0];
	my $TX_DATE = substr(${CONTROL_FILE},length($
	{CONTROL_FILE})-12, 8);
	
	open(STDERR, ">&STDOUT");
	my $TRG_COL_LIST=<<TRGCOLLIST ;
	MSISDN
	,dat_rcd_dt
	,vst_rgn_cd
	
	TRGCOLLIST
	
	my $MAP_COL_LIST=<<MAPCOLLIST ;
	TB_04024.msisdn
	,CAST('$TX_DATE' AS DATE)
	,TB_04024.vst_rgn_cd
	MAPCOLLIST
	
	my $FILTER = "CMCC_Prov_Prvd_Id=$PCODE";
	my $table_today ="${TEMPDB}.TB_04024_$
	{PCODE}";
	my $table_target = "Z" . substr($PCODE,1,2) .
	"NET_LTE_GPRS_CDR";
	my $UNIT_FLAG;
	
	sub BTEQ_S4
	{
	my ($dbh) = @_;
	my $BTEQ_CMD_S4=<<ENDOFINPUT ;
	
	DECLARE lv_mig_obj_exists_check NUMBER (1) ;
	lv_mig_errorcode NUMBER (4) ;
	
	BEGIN
	
	BEGIN

Oracle Syntax	Syntax After Migration

ENDOFINPUT
return $BTEQ_CMD_S4;
}
my $unit_num = "04024";
 	
sub BTEQ_Z1
{
my $BTEQ_CMD_Z1 = <<ENDOFINPUT;
INSERT INTO ${PDATADB}.$table_target (
$TRG_COL_LIST)
SELECT
$MAP_COL_LIST
FROM $SDATADB.TB_${PCODE}_${unit_num}_$
{UNIT_FLAG}_${TX_DATE} tb_${unit_num};
.IF ERRORCODE <> 0 THEN .QUIT 12;
ENDOFINPUT
return $BTEQ_CMD_Z1;
}
 	
sub main()
{
my $dbh=ETL::DBconnect();
# SDATA 	
$UNIT_FLAG = ETL::get_UNIT_FLAG($dbh,
"TB_04024", $PCODE, $TX_DATE);
my $BTEQCMD = "";
if ($UNIT_FLAG eq " ") {
print "Ô!\n";
ETL::disconnectETL($dbh);
return 1;
} elsif ($UNIT_FLAG eq "S") {
$BTEQCMD = BTEQ_S4($dbh);
} elsif ($UNIT_FLAG eq "Z") {
$BTEQCMD = BTEQ_Z1($dbh);
} else {
print "\n";
return 1;
}
ETL::disconnectETL($dbh);
 	
return ETL::ExecuteBTEQ($BTEQCMD, $TX_DATE);
}
my $ret= main();
exit($ret);
	INSERT INTO ${PDATADB}.$table_target
	($TRG_COL_LIST)
	SELECT
	$MAP_COL_LIST
	FROM
	$SDATADB.TB_${PCODE}_04024_$
	{UNIT_FLAG}_${TX_DATE} TB_04024 ;
	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 ;
	/
	ENDOFINPUT
	return $BTEQ_CMD_S4;
	}
	my $unit_num = "04024";
	
	sub BTEQ_Z1
	{
	my $BTEQ_CMD_Z1=<<ENDOFINPUT ;
	
	DECLARE lv_mig_obj_exists_check NUMBER (1) ;
	lv_mig_errorcode NUMBER (4) ;
	
	BEGIN
	INSERT INTO ${PDATADB}.$table_target
	($TRG_COL_LIST)
	SELECT
	$MAP_COL_LIST
	FROM
	$SDATADB.TB_${PCODE}_$
	{unit_num}_${UNIT_FLAG}_${TX_DATE} tb_$
	{unit_num} ;
	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 ;
	/
	ENDOFINPUT
	return $BTEQ_CMD_Z1;
	}
	
	sub main()
	{
	my $dbh=ETL::DBconnect();
	# SDATA
	$UNIT_FLAG = ETL::get_UNIT_FLAG($dbh,
	"TB_04024", $PCODE, $TX_DATE);

Oracle Syntax	Syntax After Migration

	my $BTEQCMD = "";
	if ($UNIT_FLAG eq " ") {
	print "!\n";
	ETL::disconnectETL($dbh);
	return 1;
	} elsif ($UNIT_FLAG eq "S") {
	$BTEQCMD = BTEQ_S4($dbh);
	} elsif ($UNIT_FLAG eq "Z") {
	$BTEQCMD = BTEQ_Z1($dbh);
	} else {
	print "˛\n";
	return 1;
	}
	ETL::disconnectETL($dbh);
	
	return ETL::ExecuteBTEQ($BTEQCMD, $TX_DATE);
	}
	my $ret= main();
	exit($ret);

6.8.13 DSQL

DSQL Variable: With AS

Input

```
/* VARIABLE INPUT */
select cast(cast('${TX_DATE}' as date format 'yyyymmdd') as date format 'yyyy-mm-dd') as v_Trx_Dt;
.IF ERRORCODE <> 0 THEN .QUIT 12 ;

insert into VT_RTL_ACT_ORG_INF          /* VT_         */
(
   Act_Id          /*      */
  ,Act_Mdf          /*       */
  ,Act_Agr_Typ_Cd          /*          */
  ,Opn_BBK          /*      */
  ,Opn_BRN          /*      */
)
select
   a.Agr_Id as Act_Id          /*      */
  ,a.Agr_Mdf as Act_Mdf          /*       */
  ,a.Agr_Typ_Cd as Act_Agr_Typ_Cd          /*          */
  ,a.BBK_Org_Id as Opn_BBK          /*      */
  ,a.Opn_Org_Id as Opn_BRN          /*      */
from ${PDM_VIEW}.T03_AGR_TRSR_BND_ACT as a          /*   _     */
where  a.DW_Start_Dt <= cast('${v_Trx_Dt}' as date format 'yyyy-mm-dd')
          AND a.DW_End_Dt > cast('${v_Trx_Dt}' as date format 'yyyy-mm-dd')
          AND a.Agr_Typ_Cd IN ('20301')          /*      */
;
.IF ERRORCODE <> 0 THEN .QUIT 12 ;
```

Output

```
DECLARE lv_mig_obj_exists_check    NUMBER ( 1 ) ;
 lv_mig_errorcode           NUMBER ( 4 ) ;
 lv_mig_v_Trx_Dt            TEXT ;

BEGIN
      /*VARIABLE INPUT */
     BEGIN
         SELECT CAST( CAST( '${TX_DATE}' AS DATE ) AS DATE )
     INTO lv_mig_v_Trx_Dt ;
```



```
  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 VT_RTL_ACT_ORG_INF /* VT_         */
        (
            Act_Id /*      */
            ,Act_Mdf /*       */
            ,Act_Agr_Typ_Cd /*          */
            ,Opn_BBK /*      */
            ,Opn_BRN /*      */
        )
        SELECT     a.Agr_Id AS Act_Id /*      */
                ,a.Agr_Mdf AS Act_Mdf /*       */
                ,a.Agr_Typ_Cd AS Act_Agr_Typ_Cd /*          */
                ,a.BBK_Org_Id AS Opn_BBK /*       */
                ,a.Opn_Org_Id AS Opn_BRN /*      */
            FROM ${PDM_VIEW}.T03_AGR_TRSR_BND_ACT AS a /*   _     */
            WHERE a.DW_Start_Dt <= CAST( lv_mig_v_Trx_Dt AS DATE )
                AND a.DW_End_Dt > CAST( lv_mig_v_Trx_Dt AS DATE )
                AND a.Agr_Typ_Cd IN ( '20301' ) /*       */
                ;
                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 ;
/
```

DSQL Variable: Without AS

Input

```
/* SESSION INPUT */
select Session ;
.IF ERRORCODE <> 0 THEN .QUIT 12 ;

select username,clientsystemuserid,clientipaddress,clientprogramname
  from dbc.sessioninfoV
 where sessionno = '$Session' ;
.IF ERRORCODE <> 0 THEN .QUIT 12 ;
```

Output

```
DECLARE lv_mig_obj_exists_check    NUMBER ( 1 ) ;
 lv_mig_errorcode           NUMBER ( 4 ) ;
 lv_mig_Session             TEXT ;

BEGIN
    /*SESSION INPUT */
    BEGIN
        SELECT pg_backend_pid ( )
    INTO lv_mig_Session ;
```



```
  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
        SELECT COUNT( * ) INTO lv_mig_obj_exists_check
          FROM ( SELECT username
                       ,clientsystemuserid
                       ,clientipaddress
                       ,clientprogramname
                   FROM dbc.sessioninfoV
                  WHERE sessionno = lv_mig_Session )
    LIMIT 1 ;

    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 ;
/
```

Specifying Variables in BTEQ Statements

Input

```
select case when cast('${v_Trx_Dt}' as date format'yyyy-mm-dd') = cast('${v_Tx_Mon_End_Date}' as date
format'yyyy-mm-dd')          then '0'          else '1' end  as v_IsLastDay;
.IF ERRORCODE <> 0 THEN .QUIT 12 ;
insert into VT_AS_CUST_WLTH_GRP           /* VT_       */
(
   Cust_UID           /*   UID */
  ,BBK_Org_Id           /*        */
  ,Wlth_grp_Cd           /*        */
  ,Card_Grd_Cd           /*      */
)
select
   a.Cust_UID as Cust_UID           /*   UID */
  ,a.BBK_Org_Id as BBK_Org_Id           /*        */
  ,'11' as Wlth_grp_Cd           /*        */
  ,coalesce(b.Card_Grd_Cd,'') as Card_Grd_Cd           /*      */
from VT_CUST_WLTH_GRP_LAST as a           /* VT_           */
LEFT OUTER JOIN ${BRTL_VEXT}.BRTL_AS_CUST_CARD_GRD_S as b           /* AS_         */
  on  b.Cust_Uid = a.Cust_Uid
  AND b.BBK_Org_Id = a.BBK_Org_Id
  AND b.Card_Sts_Scp_Cd = '001'           /* 001-          */
  AND b.Card_Grd_Cd IN ('080','060','040','020','010','000')
  AND b.DW_Snsh_Dt = cast('${v_Trx_Dt}' as date format'yyyy-mm-dd')
where  b.Cust_UID IS null
;
.IF v_IsLastDay = 1 THEN .GOTO doit
```

Output


```
BEGIN
    select case when cast('${v_Trx_Dt}' as date format'yyyy-mm-dd') = cast('${v_Tx_Mon_End_Date}' as date
format'yyyy-mm-dd')      then '0'          else '1' end  INTO lv_mig_v_IsLastDay;
      lv_mig_ERRORCODE = 0;
EXCEPTION
    WHEN OTHERS THEN
        lv_mig_ERRORCODE = -1;
END;
IF lv_mig_ERRORCODE <> 0 THEN
  RAISE EXCEPTION '12';
END IF;

insert into VT_AS_CUST_WLTH_GRP           /* VT_       */
(
   Cust_UID           /*   UID */
  ,BBK_Org_Id           /*        */
  ,Wlth_grp_Cd           /*        */
  ,Card_Grd_Cd           /*      */
)
select
   a.Cust_UID as Cust_UID           /*   UID */
  ,a.BBK_Org_Id as BBK_Org_Id           /*        */
  ,'11' as Wlth_grp_Cd           /*        */
  ,coalesce(b.Card_Grd_Cd,'') as Card_Grd_Cd           /*      */
from VT_CUST_WLTH_GRP_LAST as a           /* VT_           */
LEFT OUTER JOIN ${BRTL_VEXT}.BRTL_AS_CUST_CARD_GRD_S as b           /* AS_         */
  on  b.Cust_Uid = a.Cust_Uid
  AND b.BBK_Org_Id = a.BBK_Org_Id
  AND b.Card_Sts_Scp_Cd = '001'           /* 001-         */
  AND b.Card_Grd_Cd IN ('080','060','040','020','010','000')
  AND b.DW_Snsh_Dt = cast('${v_Trx_Dt}' as date format'yyyy-mm-dd')
where  b.Cust_UID IS null
;
IF lv_mig_v_IsLastDay = 1 THEN
          GOTO doit;
END IF;
```

Specifying Variables Using SELECT EXTRACT FROM

Input

```
select EXTRACT (MONTH FROM cast('${TX_DATE}' as  date format 'YYYYMMDD')) as CURR_MON;
.if ERRORCODE <> 0 then .quit 12;
--
.IF CURR_MON <> 1 THEN .goto NON_JAN;
.if ERRORCODE <> 0 then .quit 12;
```

Output

```
BEGIN
   select EXTRACT (MONTH FROM cast('${TX_DATE}' as  date format 'YYYYMMDD')) INTO CURR_MON;
...
END;
...
```

DATE Type Casting: Specifying DSQL Variables

Input

```
select
   case when a.DW_Stat_Dt in
(date'${v_Tx_Pre_1_Mon_End_Date}'
,date'${v_Tx_Pre_2_Mon_End_Date}'
,date'${v_Tx_Pre_3_Mon_End_Date}')
then '1' else '2' end as Quarter_Flg           /* PK-     1--    2--     3--   */
  ,a.BBK_Org_Id as BBK_Org_Id           /* PK-         */
  ,a.Cust_UID as Cust_UID           /* PK-  UID */
  ,a.Entp_Cust_Id as Entp_Cust_Id           /* PK-       */
```



```
 ,coalesce(b.BBK_Nbr,'') as Agn_BBK_Org_Id          /* PK-       */
from ${BRTL_VEXT}.BRTL_AS_AGN_ENTP_CUID_TRX_SR as a          /* AS_      UID     */
LEFT OUTER JOIN ${BRTL_VCOR}.BRTL_OR_RTL_ORG_INF_S as b          /* OR_         */
  on  a.Agn_BRN_Org_Id = b.Rtl_Org_Id
and b.DW_Snsh_Dt = cast('${v_Trx_Dt}' as date format 'yyyy-mm-dd')
where  a.DW_Stat_Dt IN (date'${v_Tx_Pre_1_Mon_End_Date}'
,date'${v_Tx_Pre_2_Mon_End_Date}'
,date'${v_Tx_Pre_3_Mon_End_Date}'
,date'${v_Tx_Pre_4_Mon_End_Date}'
,date'${v_Tx_Pre_5_Mon_End_Date}'
,date'${v_Tx_Pre_6_Mon_End_Date}')
          AND a.Stat_Prd_Cd = 'M001'
group by Quarter_Flg, a.BBK_Org_Id, a.Cust_UID, a.Entp_Cust_Id, coalesce(b.BBK_Nbr,'')
;
should be migrated as below:
          SELECT
                  CASE WHEN a.DW_Stat_Dt IN
      ( CAST(lv_mig_v_Tx_Pre_1_Mon_End_Date AS DATE)
                   , CAST(lv_mig_v_Tx_Pre_2_Mon_End_Date AS DATE)
                   , CAST(lv_mig_v_Tx_Pre_3_Mon_End_Date AS DATE) )
                      THEN '1'
                     ELSE '2'
                  END AS Quarter_Flg /* PK-     1--    2--     3--   */
                  ,a.BBK_Org_Id AS BBK_Org_Id /* PK-         */
                  ,a.Cust_UID AS Cust_UID /* PK-  UID */
                  ,a.Entp_Cust_Id AS Entp_Cust_Id /* PK-       */
                  ,COALESCE( b.BBK_Nbr ,'' ) AS Agn_BBK_Org_Id /* PK-       */
              FROM
                  BRTL_VEXT.BRTL_AS_AGN_ENTP_CUID_TRX_SR AS a /* AS_      UID     */
                  LEFT OUTER JOIN BRTL_VCOR.BRTL_OR_RTL_ORG_INF_S AS b /* OR_         */
                      ON a.Agn_BRN_Org_Id = b.Rtl_Org_Id
                  AND b.DW_Snsh_Dt = CAST( lv_mig_v_Trx_Dt AS DATE )
              WHERE
                  a.DW_Stat_Dt IN ( CAST(lv_mig_v_Tx_Pre_1_Mon_End_Date AS DATE)
                  ,CAST(lv_mig_v_Tx_Pre_2_Mon_End_Date AS DATE)
                  ,CAST(lv_mig_v_Tx_Pre_3_Mon_End_Date AS DATE)
                  ,CAST(lv_mig_v_Tx_Pre_4_Mon_End_Date AS DATE)
                  ,CAST(lv_mig_v_Tx_Pre_5_Mon_End_Date AS DATE)
                  ,CAST(lv_mig_v_Tx_Pre_6_Mon_End_Date AS DATE) )
                  AND a.Stat_Prd_Cd = 'M001'
              GROUP BY Quarter_Flg, a.BBK_Org_Id, a.Cust_UID, a.Entp_Cust_Id, COALESCE( b.BBK_Nbr ,'' )
 ;
```

Output

```
SELECT
                  CASE WHEN a.DW_Stat_Dt IN
      ( CAST(lv_mig_v_Tx_Pre_1_Mon_End_Date AS DATE)
                   , CAST(lv_mig_v_Tx_Pre_2_Mon_End_Date AS DATE)
                   , CAST(lv_mig_v_Tx_Pre_3_Mon_End_Date AS DATE) )
                      THEN '1'
                     ELSE '2'
                  END AS Quarter_Flg /* PK-     1--    2--     3--   */
                  ,a.BBK_Org_Id AS BBK_Org_Id /* PK-         */
                  ,a.Cust_UID AS Cust_UID /* PK-  UID */
                  ,a.Entp_Cust_Id AS Entp_Cust_Id /* PK-       */
                  ,COALESCE( b.BBK_Nbr ,'' ) AS Agn_BBK_Org_Id /* PK-       */
              FROM
                  BRTL_VEXT.BRTL_AS_AGN_ENTP_CUID_TRX_SR AS a /* AS_      UID     */
                  LEFT OUTER JOIN BRTL_VCOR.BRTL_OR_RTL_ORG_INF_S AS b /* OR_         */
                      ON a.Agn_BRN_Org_Id = b.Rtl_Org_Id
                  AND b.DW_Snsh_Dt = CAST( lv_mig_v_Trx_Dt AS DATE )
              WHERE
                  a.DW_Stat_Dt IN ( CAST(lv_mig_v_Tx_Pre_1_Mon_End_Date AS DATE)
                  ,CAST(lv_mig_v_Tx_Pre_2_Mon_End_Date AS DATE)
                  ,CAST(lv_mig_v_Tx_Pre_3_Mon_End_Date AS DATE)
                  ,CAST(lv_mig_v_Tx_Pre_4_Mon_End_Date AS DATE)
                  ,CAST(lv_mig_v_Tx_Pre_5_Mon_End_Date AS DATE)
                  ,CAST(lv_mig_v_Tx_Pre_6_Mon_End_Date AS DATE) )
                  AND a.Stat_Prd_Cd = 'M001'
```



```
        GROUP BY Quarter_Flg, a.BBK_Org_Id, a.Cust_UID, a.Entp_Cust_Id, COALESCE( b.BBK_Nbr ," )
;
```

6.9 Oracle Syntax Migration

6.9.1 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.

6.9.2 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**.

6.9.2.1 Tables

CREATE TABLE

The Oracle **CREATE TABLE** statement is used to create new tables. The target database supports Oracle CREATE TABLE without any migration.

ALTER TABLE

The Oracle ALTER TABLE statement is used to add, rename, modify, or drop/delete columns in a table. The target database supports Oracle ALTER TABLE without any migration.

PRIMARY KEY

The Oracle ALTER TABLE statement 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

The **ALTER TABLE** query contains UNIQUE Constraint. If it is directly executed in GaussDB, the following error shows: "Cannot create index whose evaluation cannot be enforced to remote nodes".

Implementation is similar to PRIMARY KEY. If PRIMARY KEY/UNIQUE is already present, there is no need to migrate and leave it as it is.

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;
```

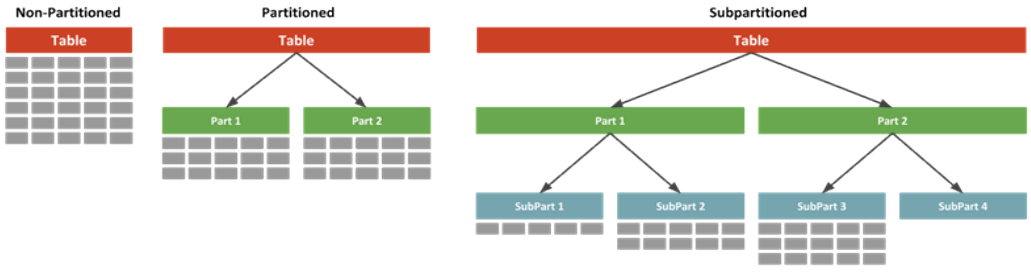
Output

```
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 6-6 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 PARTITIONED 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 ranges of data 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 BL_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 Gauss, hence it is commented in the migrated output. This is based on the following configuration item: **commentStorageParameter=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 Gauss. Storage parameters are commented when it appears in tables or indexes based on the value of the config parameter **commment_storage_parameter**.

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 **commment_storage_parameter** is set TRUE, then storage parameters are commented.

STORE

The STORE keyword for LOB columns is not suppored in Gauss, and it is commented 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 Gauss 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')))*/;

6.9.2.2 Temporary Tables

GaussDB(DWS) does not support **GLOBAL TEMPORARY TABLE**, and it will be migrated to **LOCAL TEMPORARY TABLE**.

ON COMMIT DELETE ROWS is also not supported, and will be migrated to **ON COMMIT PRESERVE ROWS**.

Figure 6-7 Input: TEMPORARY TABLE

```
CREATE
    GLOBAL TEMPORARY TABLE
        schema1.temp_tbl1 (
            col1 VARCHAR2 (400)
            ,col2 DATE NOT NULL
        )
            ON COMMIT DELETE ROWS
;
```

Figure 6-8 Output: TEMPORARY TABLE

```
CREATE
    LOCAL TEMPORARY TABLE
        schema1_temp_tbl1 (
            col1 VARCHAR2 (400)
            ,col2 DATE NOT NULL
        )
            ON COMMIT PRESERVE ROWS
;
```


6.9.2.3 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 ;
```

6.9.2.4 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 6-9 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 6-10 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 )) );
```

6.9.2.5 Views

A view is a logical table that is 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.

Figure 6-11 Input: Views

```
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'
                )
;
```

Figure 6-12 Output: Views

```
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'
                )
    )
;
```


6.9.2.6 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 **custom_scripts.sql** file and paste it to execute the script in all the target databases. For details, see **Migration Process**.

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
        ,999999999999999999999999999
        ,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
        ,999999999999999999999999999
        ,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 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 **custom_scripts.sql** file and paste it to execute the script in all the target databases. For details, see **Migration Process**.

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 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 **custom_scripts.sql** file and paste it to execute the script in all the target databases. For details, see **Migration Process**.

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
;
```

6.9.2.7 PURGE

In Oracle, the **DROP TABLE** statement moves a table to the recycle bin. The **PURGE** statement is used to remove a table or index from the recycle bin and release all of the space associated with the object. This statement also removes the entire recycle bin, or part or all of a dropped tablespace from the recycle bin. The migrated query does not contain **PURGE**.

Figure 6-13 Input: PURGE

```
Execute immediate 'Drop table
table1 purge' ;

drop table test.emp purge ;
```

Figure 6-14 Output: PURGE

```
Execute immediate 'Drop table table1' ;


drop table test.emp ;
```

6.9.2.8 Database Keywords

DSC supports GaussDB(DWS) keywords, such as **NAME**, **LIMIT**, **OWNER**, **KEY**, and **CAST**. These keywords must be enclosed in double quotation marks.

Gauss Keywords (NAME, VERSION, LABEL, 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 Gauss keywords are used as aliases for any column in the SELECT list without defining the AS keyword, the AS keyword to define the aliases.

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 GAUSS.

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)
	);

6.9.3 COMPRESS Phrase

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)
) ;
```

6.9.4 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) ;
```

6.9.5 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))*/
;
```

6.9.6 Supplemental Log Data

Supplemental columns can be recorded in redo log files. The process of recording these additional columns is called supplemental logging. The Oracle database supports this function, but GaussDB 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*/
  )
```

6.9.7 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)
 );
```

6.9.8 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*/
 );
```


6.9.9 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 6-15 Input - Order of Clauses

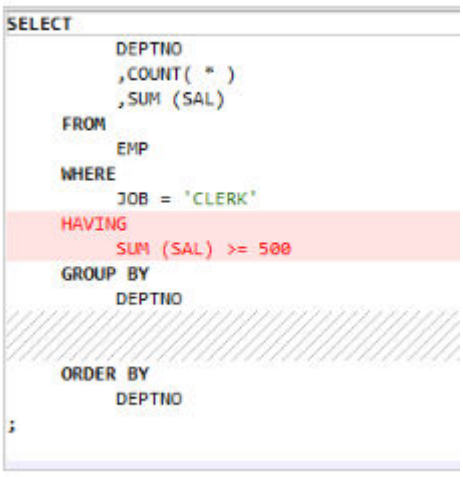
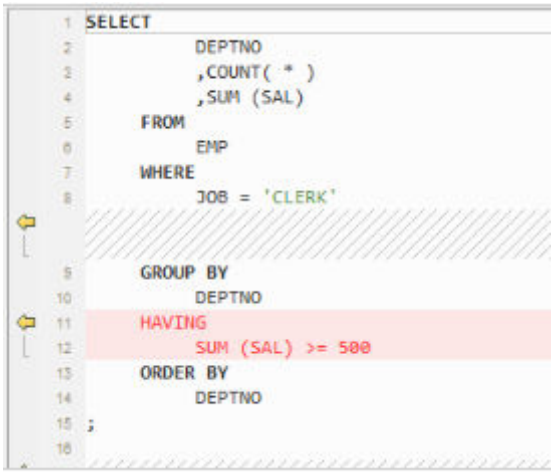


Figure 6-16 Output - Order of Clauses



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 6-17 Input - Extended group by clause

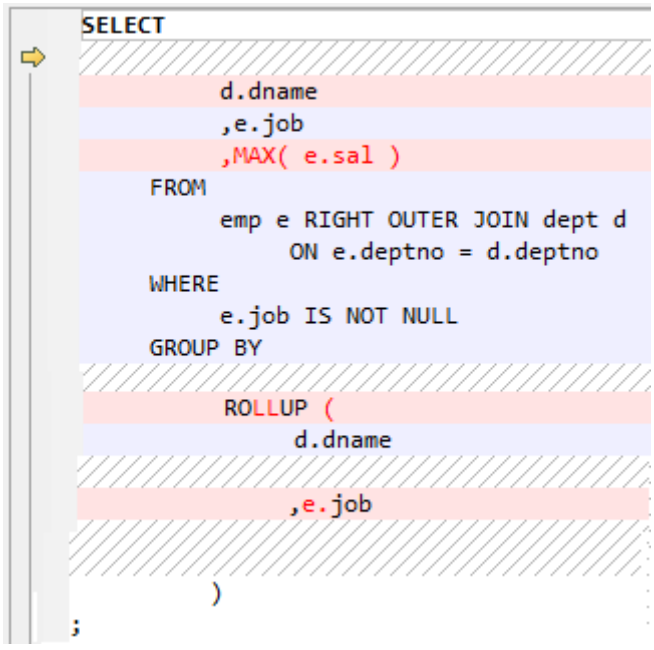
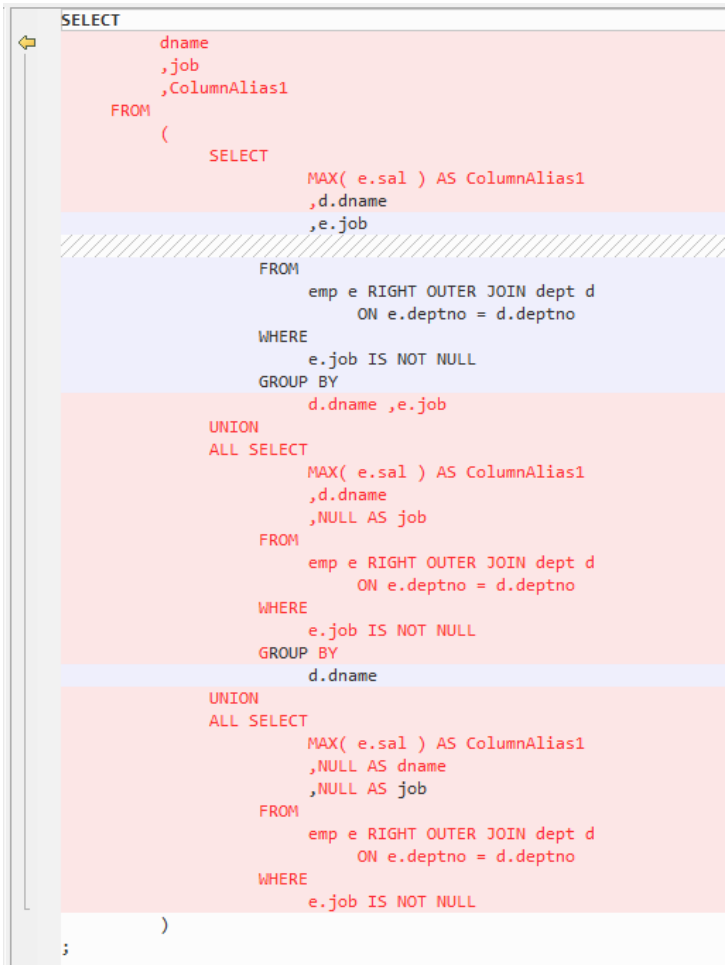


Figure 6-18 Output - Extended group by clause



```
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
        )
;
```

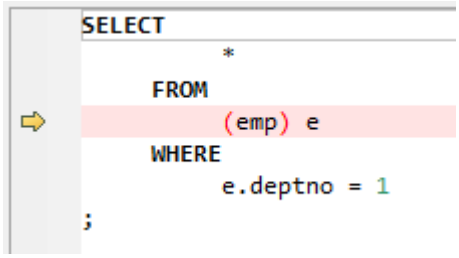
GROUPING_ID and ROLLUP

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**

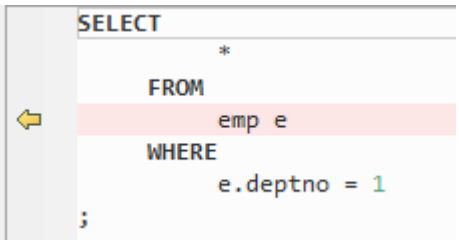
   Table names do not need to be specified within parentheses. However, allows using brackets.

Figure 6-19 Input - Table name inside brackets



```
SELECT
        *
    FROM
        (emp) e
    WHERE
        e.deptno = 1
;
```


Figure 6-20 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.
4. The svproduct_is_for_pa function in **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 6-21 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 6-22 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 6-23 Input - Insert first

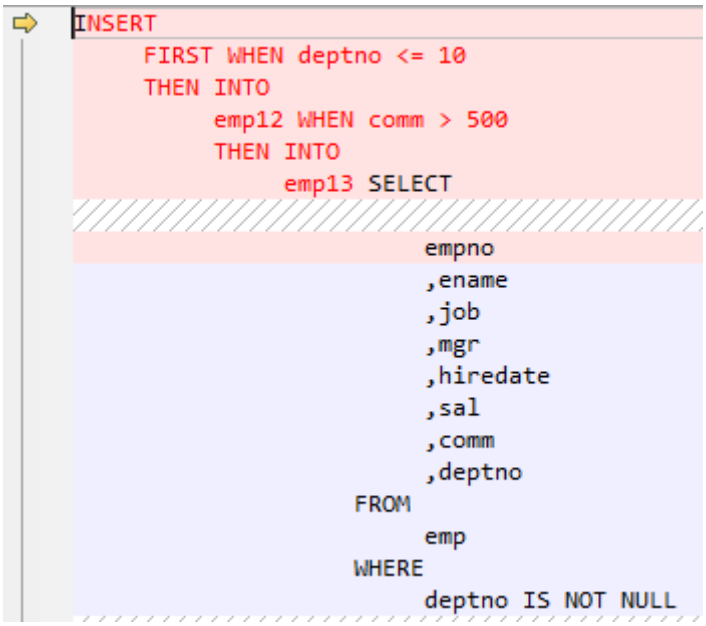
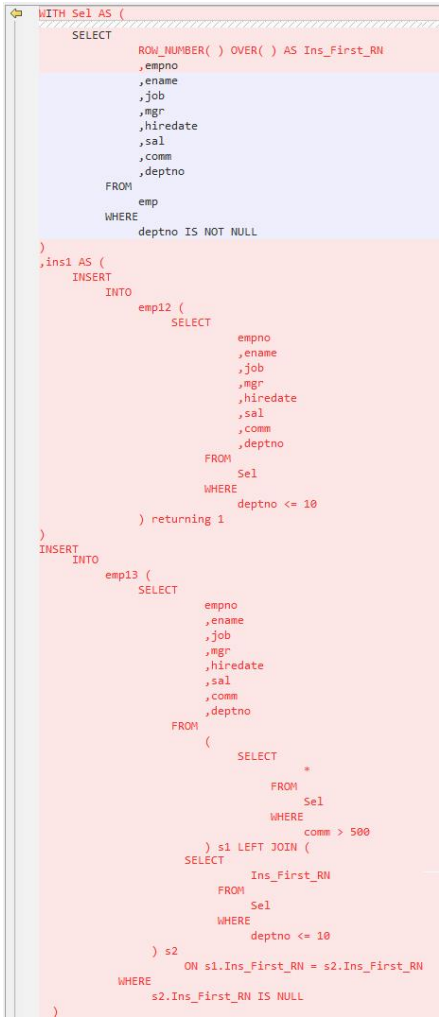


Figure 6-24 Output - Insert first



3. **INSERT with Table Alias**

   The Oracle **table aliases** is 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.

Currently, this function is supported by GaussDB(DWS) 6.5.0 and later. 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 6-25 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 6-26 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 6-27 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 6-28 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;
```

6.9.10 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 6-29 Input - ROWID

```
SELECT
            empid
            ,ename
            ,ROWID
      FROM
            employees
;
```

Figure 6-30 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 6-31 Input - ROWNUM

```
SELECT
        e.empid
        ,e.ename
    FROM
        employees e
    WHERE
        ROWNUM < 6
;
```

Figure 6-32 Output - ROWNUM

```
SELECT
        e.empid
        ,e.ename
    FROM
        employees e LIMIT 6 - 1
;
```

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 exectuing 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

Scripts of UPDATE and DELETE that are migrated through ROWNUM contains LIMIT, which is not supported by Gauss.

```
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
      );
```

6.9.11 OUTER JOIN

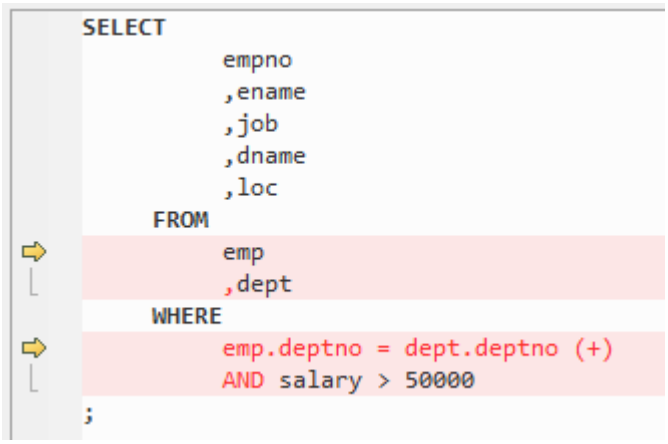
This section describes the migration syntax of Oracle **OUTER JOIN**. The migration syntax determines how the keywords/features are migrated.

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.

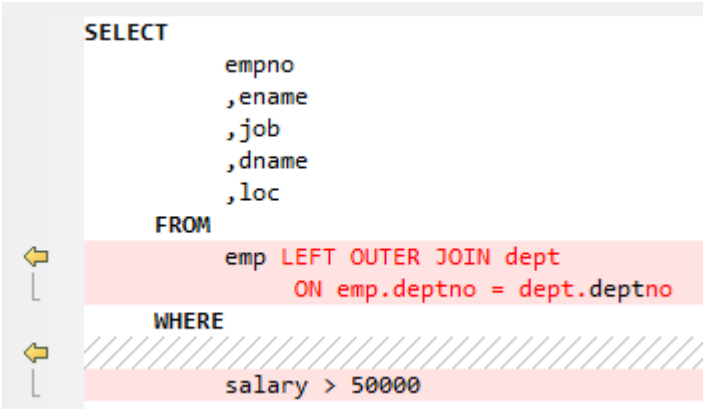
GaussDB(DWS) does not support the **+** operator. The function of this operator is enabled using **LEFT OUTER JOIN** and **RIGHT OUTER JOIN** keywords.

Figure 6-33 Input: OUTER JOIN



```
SELECT
        empno
        ,ename
        ,job
        ,dname
        ,loc
    FROM
        emp
        ,dept
    WHERE
        emp.deptno = dept.deptno (+)
        AND salary > 50000
;
```

Figure 6-34 Output: OUTER JOIN



```
SELECT
        empno
        ,ename
        ,job
        ,dname
        ,loc
    FROM
        emp LEFT OUTER JOIN dept
            ON emp.deptno = dept.deptno
    WHERE

        salary > 50000
```

6.9.12 OUTER QUERY (+)

Join is supported in Gauss 18.2.0, 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
;
```

6.9.13 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;
```

6.9.14 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**.

6.9.14.1 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' */))
 );
```

TO_DATE with SYYYY in Year Format

SYYYY is not supported in date format. This is applicable for GaussDB T.

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'))
 );
```

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'))
 );
```


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 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**.

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;
```

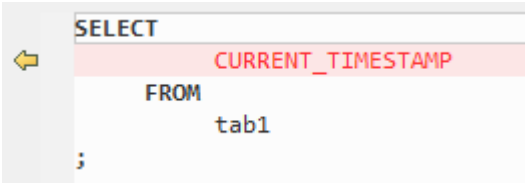
SYSTIMESTAMP

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 6-35 Input - SYSTIMESTAMP

```
SELECT
        SYSTIMESTAMP
    FROM
        tab1
;
```


Figure 6-36 Output - SYSTIMESTAMP



6.9.14.2 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

DBMS_LOB.SUBSTR is supported in V1R8C10. You can specify whether to migrate this fucntion 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);
```

6.9.14.3 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 6-37 Input - Listagg

```
SELECT
        deptno
        ,ename
        ,LISTAGG (
            ename
            ,': '
        ) OVER( PARTITION BY deptno ,ename ORDER BY ename ) AS rn
    FROM
        emp
    ORDER BY
        deptno
        ,ename
;
```


Figure 6-38 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 6-39 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 6-40 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 value2.

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 6-41 Input - Quote

```
SELECT
          q'[It's a string quote operator.]'
     FROM
          dual
;
```

Figure 6-42 Output - Quote

```
SELECT
          $q$It's a string quote operator.$q$
     FROM
          dual
;
```

6.9.14.4 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
;
```


6.9.14.5 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"
```

6.9.15 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**. Comment out related commands in the 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. So it needs to be removed from the target 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 6-43 Input - PL/SQL

```
BEGIN
 ...
 v_rowcount : = SQL%ROWCOUNT;
 ..
END;
```

Figure 6-44 Output - PL/SQL

```
BEGIN
 ...
 v_rowcount := SQL%ROWCOUNT;
 ..
END;
```

END

END with label is not supported in GaussDB T, 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 to V100R200C60, set the **exceptionHandler** parameter to **True**.

For DSC 18.2.0, this parameter must be set to the default value **False**.

Figure 6-45 Input - 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 get_salary ;
/
```

Figure 6-46 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 6-47 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 6-48 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 ;
/
```

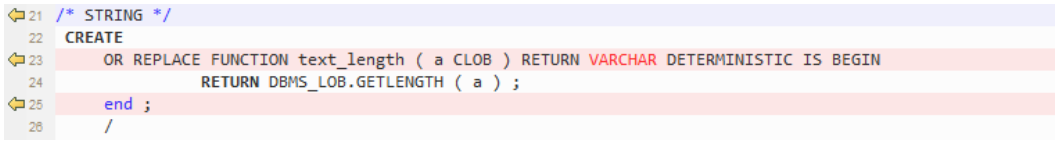
STRING

STRING is an Oracle PL/SQL data type that is not supported by GaussDB T. This data type is handled by using VARCHAR.

Figure 6-49 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 6-50 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 6-51 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 6-52 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 6-53 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 6-54 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 that is SAD.SAD_CALC_ITEM_PKG_TEST_OB#error_msg ( ) but this function error_msg is defined without parameter that is

```
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 ;
/
```

6.9.16 PL/SQL Collections (Using User-Defined Types)

This section descripes 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 uses built-in datatypes and other user-defined datatypes as the building blocks for datatypes that model the structure and behavior of data in applications. 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 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

With the SUBTYPE statement, PL/SQL allows you to define your own subtypes or aliases of predefined datatypes, sometimes referred to as abstract datatypes.

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

As SUBTYPE is not supported in GaussDB, the SUBTYPE variable needs to be replaced with the actual type.

6.9.17 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:

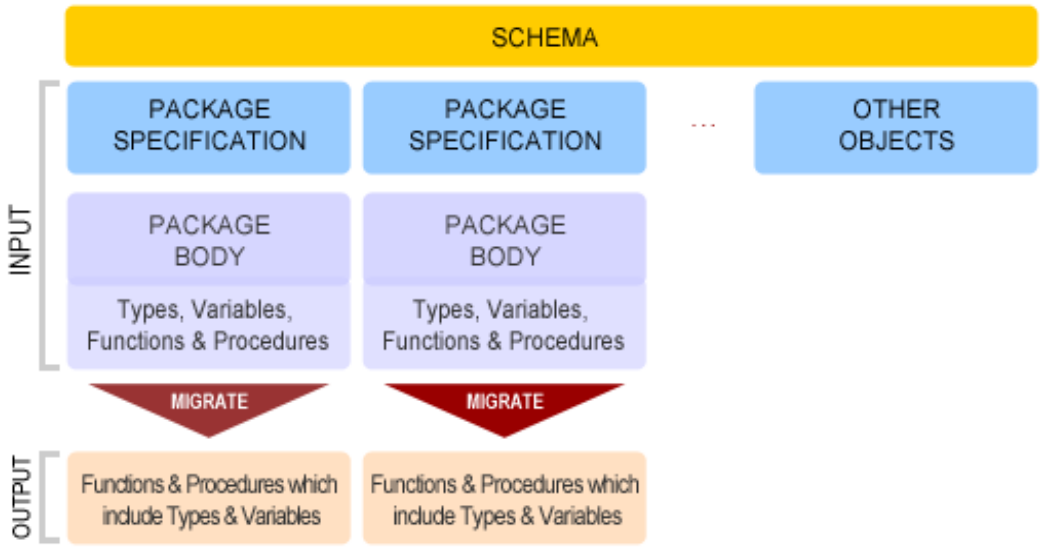
Packages, Package Variables, Package Split, REF CURSOR, VARRAY, Creating a Schema for Package, Grant Execute, Package Name List and Data Type. For details, see **Packages** to **Data Type**.

6.9.17.1 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 6-55 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 actualy 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 NULL constraint is not supported on variable declaration by Gauss, so it is recomended to comment 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 belongs 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 ;
/
```

6.9.17.2 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 veriable with default vaue declared in one package by using CONSTANT keyword and used in another package

The global veriable declared in the package specification is accessed in the same or another pacakge.

```
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 veriable 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 packege 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 packege 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 declared as EXCEPTION, which is not supported in GaussDB.

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.

6.9.17.3 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 ;
/
```

6.9.17.4 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 ;
 /
```

6.9.17.5 Creating a Schema for Package

The pacakge declaration is miagrated 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
;
```

6.9.18 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;
```

6.9.19 Granting Execution Permissions

This feature is used to give privileges to users for specific packages. As GaussDB does not support packages, all the procedures and functions defined in the specific packages will be granted the execution permssion.

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;
```

6.9.20 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.
```

6.9.21 Data Type

Subtype

The customized type 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 package procdure/function contains %ROWTYPE attribute in IN/OUT parameter and 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 ;
/
```

6.9.22 Chinese Character Support

Input-Chinese (

```
create table test11(a int,b int)/*create table test11(a int,b int)*/;
```

Output

```
create table test11(a int,b int)/*create table test11(a int,b int)*/;
```

Input-Chinese)

```
create table test11(a int,b int)/*create table test11(a int,b int)*/;
```

Output

```
create table test11(a int,b int)/*create table test11(a int,b int)*/;
```

Input-Chinese,

```
create table test11(a int,b int)/*create table test11(a int,b int)*/;
```

Output

```
create table test11(a int,b int)/*create table test11(a int,b int)*/;
```

Input-Support Chinese SPACE

```
create table test11(a int,b int)/*create table test11(a int,b int)*/;
```

Output

```
create table test11(a int,b int)/*create table test11(a int,b int)*/;
```

6.10 Netezza Syntax Migration

6.10.1 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) */ ;

6.10.2 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;
	/

6.10.3 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_START,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_START,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 excute 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;
l_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;
	l_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;
	/

Gauss keyword

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;
	/

6.10.4 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;
	/

6.10.5 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

6.10.6 DML

Gauss keyword: SOURCE specified as column-alias without 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;

6.10.7 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.

6.11 MySQL Syntax Migration

This section lists the MySQL syntax features supported by the syntax migration tool, and provides the MySQL syntax and the equivalent GaussDB(DWS) syntax for each feature. The syntax listed in this section illustrates the internal logic for migrating MySQL scripts.

This section is also a reference for the database migration team and for the on-site verification of MySQL script migration.

6.11.1 Basic Data Types

Overview

MySQL supports a number of basic data types, including numeric, date/time, string (character), LOB, set, binary, and Boolean types. GaussDB(DWS) does not support some basic MySQL data types, precision settings of some MySQL data types, or some MySQL keywords such as **UNSIGNED** and **ZEROFILL**. DSC performs migration based on GaussDB support.

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. For details of each types description, see **Table 6-20**

Table 6-20 Basic description of data types

Data Types	Basic Description
Numeric Types	For details, see **Numeric Types**.
Date/Time Types	This document describes the following **date/time** types: **DATETIME**, **TIME**, **TIMESTAMP**, and **YEAR**. GaussDB(DWS) does not support these types, and DSC will convert them. For details, see **Date/Time Types**.
String Types	MySQL interprets length specifications in character column definitions in character units. This applies to the CHAR, VARCHAR, and TEXT types. For details, see **String Types**.

Data Types	Basic Description
Spatial Data Types	MySQL has spatial data types corresponding to the OpenGIS class. For details, see **Spatial Data Types**.
LOB Types	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. For details, see **LOB Types**.
Set Types	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. For details, see **Set Types**.
Boolean Types	MySQL supports both BOOL and BOOLEAN. For details, see **Boolean Types**.
Binary Types	1. In MySQL, the BIT data type is used to store bit values. A type of BIT(*M*) enables storage of *M*-bit values. *M* can range from 1 to 64. 2. MySQL BINARY and VARBINARY types are similar to CHAR and VARCHAR, except that they contain binary strings rather than non-binary strings. For details, see **Binary Types**.

Numeric Types

Table 6-21 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]	FLOAT FLOAT[(M)]

MySQL Numeric Type	MySQL Input	GaussDB(DWS) Output
DOUBLE	DOUBLE[(M[,D])] [UNSIGNED] [ZEROFILL]	FLOAT[(M)]
FIXED	FIXED FIXED[(M[,D])] [UNSIGNED] [ZEROFILL]	DECIMAL DECIMAL[(M[,D])]
FLOAT	FLOAT FLOAT [(M[,D])] [UNSIGNED] [ZEROFILL] FLOAT(p) [UNSIGNED] [ZEROFILL]	FLOAT FLOAT[(M)] FLOAT(p)
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])]	FLOAT[(M)]
SMALLINT	SMALLINT SMALLINT(p) [UNSIGNED] [ZEROFILL]	SMALLINT
TINYINT	TINYINT TINYINT(n) TINYINT(n) ZEROFILL TINYINT(n) UNSIGNED ZEROFILL	TINYINT TINYINT TINYINT SMALLINT

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 "public"."runoob_datatype_test"
(
  "datatype_1" TINYINT,
  "datatype_2" TINYINT,
  "datatype_3" TINYINT,
  "datatype_4" SMALLINT,
  "datatype_5" TINYINT
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_1");
```

Date/Time Types

Table 6-22 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)]	VARCHAR(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 "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 "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 "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 "public"."runoob_datatype_test"
(
 "datatype_1" VARCHAR(4),
 "datatype_2" VARCHAR(4),
 "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");
```

String Types

Table 6-23 String type mapping

MySQL String Type	MySQL Input	GaussDB(DWS) Output
CHAR	CHAR[(0)]	CHAR[(1)]
LONGTEXT	LONGTEXT	TEXT
MEDIUMTEXT	MEDIUMTEXT	TEXT
TEXT	TEXT	TEXT
TINYTEXT	TINYTEXT	TEXT
VARCHAR	VARCHAR[(0)]	VARCHAR[(1)]

Input: CHAR

In MySQL, the length of a CHAR column is fixed to the length that you declare when you create the table. The length can be any value from 0 to 255. When CHAR values are stored, they are right-padded with spaces to the 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 "public"."runoob_datatype_test"
(
 "datatype_1" CHAR NOT NULL,
 "datatype_2" CHAR(1) NOT NULL,
 "datatype_3" CHAR(255) 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 "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 "public"."runoob_datatype_test"
(
  "datatype_1" VARCHAR(1),
  "datatype_2" VARCHAR(1845)
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_1");
```

Spatial Data Types

Table 6-24 Spatial type mapping

MySQL Spatial Type	MySQL Input	GaussDB(DWS) Output
GEOMETRY	GEOMETRY	CIRCLE
POINT	POINT	POINT
LINESTRING	LINESTRING	POLYGON
POLYGON	POLYGON	POLYGON
MULTIPOINT	MULTIPOINT	BOX
MULTILINESTRING	MULTILINESTRING	BOX
MULTIPOLYGON	MULTIPOLYGON	POLYGON
GEOMETRYCOLLECTION	GEOMETRYCOLLECTION	CIRCLE

- 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" CIRCLE 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" CIRCLE NOT NULL,
  PRIMARY KEY ("id")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("id");
```

LOB Types

Table 6-25 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 "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");
```

Set Types

Table 6-26 Set type mapping

MySQL Set Type	MySQL Input	GaussDB(DWS) Output
ENUM	ENUM	VARCHAR(14)
SET	SET	VARCHAR(14)

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 "public"."runoob_datatype_test"
(
  "id" INTEGER(2) PRIMARY KEY,
  "datatype_17" VARCHAR(14)
)
  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 "public"."runoob_tbl_test"
(
  "datatype_18" VARCHAR(14)
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
```



```
 NOCOMPRESS
 DISTRIBUTE BY HASH ("datatype_18");
```

Boolean Types

Input: BOOL/BOOLEAN

```
CREATE TABLE IF NOT EXISTS `runoob_dataType_test`(
   `dataType_1` INT,
   `dataType_2` BOOL,
   `dataType_3` BOOLEAN
);
```

Output

```
CREATE TABLE "public"."runoob_datatype_test"
(
 "datatype_1" INTEGER,
 "datatype_2" BOOLEAN,
 "datatype_3" BOOLEAN
)
 WITH ( ORIENTATION = ROW, COMPRESSION = NO )
 NOCOMPRESS
 DISTRIBUTE BY HASH ("datatype_1");
```

Binary Types

Table 6-27 Binary type mapping

MySQL Binary Type	MySQL Input	GaussDB(DWS) Output
BIT[(M)]	BIT[(M)]	BIT[(M)]
BINARY[(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 "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 "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");
```

6.11.2 Table (Optional)

This section describes the migration syntax of MySQL tables (optional parameter). 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.

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 attribute, which will be converted to **SERIAL** and deleted by DSC during migration.

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(100) 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 "public"."runoob_alter_test"
(
  "datatype1" SERIAL NOT NULL,
  "datatype2" FLOAT(10),
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");
```

AVG_ROW_LENGTH

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(30),
  "runoob_title" VARCHAR(100) NOT NULL,
  "runoob_author" VARCHAR(40) NOT NULL,
  "submission_date" VARCHAR(30)
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("runoob_id");
```

In addition, GaussDB(DWS) does not allow using the **AUTO_INCREMENT** attribute to modify tables. DSC will delete the attribute after 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 "public"."runoob_alter_test"
(
  "datatype1" SERIAL NOT NULL,
  "datatype2" FLOAT(10),
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");
```


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(30),
  "runoob_title" VARCHAR(100) NOT NULL,
  "runoob_author" VARCHAR(40) NOT NULL,
  "submission_date" VARCHAR(30)
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("runoob_id");
```

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" FLOAT(10),
  "datatype3" FLOAT(20),
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");
```


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(30),
  "runoob_title" VARCHAR(100) NOT NULL,
  "runoob_author" VARCHAR(40) NOT NULL,
  "submission_date" VARCHAR(30)
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("runoob_id");
```

COMMENT

In MySQL, **COMMENT** is a comment for a table. GaussDB(DWS) does not support table definition modification using this attribute. DSC will delete the attribute during migration.

Input

```
CREATE TABLE `public`.`runoob_alter_test`(
   `dataType1` int NOT NULL AUTO_INCREMENT,
   `dataType2` FLOAT(10,2),
   PRIMARY KEY(`dataType1`)
) comment='table comment';

ALTER TABLE `public`.`runoob_alter_test` COMMENT 'table comment after modification';
```

Output

```
CREATE TABLE "public"."runoob_alter_test"
(
  "datatype1" SERIAL NOT NULL,
  "datatype2" FLOAT(10),
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");
```

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" FLOAT(20),
  "datatype3" TEXT NOT NULL,
  "datatype4" VARCHAR(4) NOT NULL DEFAULT '2018',
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");
```

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(30),
  "runoob_title" VARCHAR(100) NOT NULL,
  "runoob_author" VARCHAR(40) NOT NULL,
  "submission_date" VARCHAR(30)
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("runoob_id");
```


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" FLOAT(20),
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");

CREATE TABLE "public"."runoob_tbl_test2"
(
  "datatype1" SERIAL NOT NULL,
  "datatype2" FLOAT(20),
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");
```

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" FLOAT(20),
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");

-- A.

-- B.

-- C.
```

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" FLOAT(20),
  "datatype3" TEXT NOT NULL,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");
```

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(30),
  "runoob_title" VARCHAR(100) NOT NULL,
  "runoob_author" VARCHAR(40) NOT NULL,
  "submission_date" VARCHAR(30)
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("runoob_id");
```

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" FLOAT(20),
  "datatype3" TEXT NOT NULL,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");
```

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" FLOAT(20),
  "datatype3" TEXT NOT NULL,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");
```

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" FLOAT(10),
  "datatype3" FLOAT(20),
  "datatype4" TEXT NOT NULL,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");

--A

--B

--C
```

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" FLOAT(20),
  "datatype3" TEXT NOT NULL,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");
```

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" FLOAT(10),
  "datatype3" FLOAT(20),
  "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.
```

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(30),
  "runoob_title" VARCHAR(100) NOT NULL,
  "runoob_author" VARCHAR(40) NOT NULL,
  "submission_date" VARCHAR(30)
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("runoob_id");

-- A.
```



```
-- B.

-- C.
```

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" FLOAT(20),
  "datatype3" TEXT NOT NULL,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");

-- A.

-- B.

-- C.
```

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" FLOAT(20),
  "datatype3" TEXT NOT NULL,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");
```

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=MERGE UNION=(t1,t2) INSERT_METHOD=LAST;
```

Output

```
CREATE TABLE t1 (
    a SERIAL NOT NULL PRIMARY KEY,
    message CHAR(20)
)
WITH ( ORIENTATION = ROW, COMPRESSION = NO )
NOCOMPRESS
DISTRIBUTE BY HASH ("a");

CREATE TABLE t2 (
    a SERIAL NOT NULL PRIMARY KEY,
    message CHAR(20)
)
WITH ( ORIENTATION = ROW, COMPRESSION = NO )
NOCOMPRESS
DISTRIBUTE BY HASH ("a");

CREATE VIEW total(a, message) AS
SELECT * FROM t1
UNION ALL
SELECT * FROM t2;
```

6.11.3 Table Operations

This section contains the migration syntax for migrating MySQL table operation. The migration syntax decides how the supported keywords/features are migrated.

For details, see the following topics:

- **LIKE (Table Cloning)**
- **ADD|DROP COLUMN**
- **MODIFY (Column Modification)**
- **CHANGE (Column Modification)**
- **SET|DROP COLUMN DEFAULT VALUE**
- **DROP (Table Deletion)**
- **TRUNCATE (Table Deletion)**
- **LOCK**
- **RENAME (Table Renaming)**

LIKE (Table Cloning)

MySQL databases support **CREATE TABLE**. **LIKE** is a method with which a table is created by cloning the old table structure, and this method is 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 "public"."runoob_tbl_old"
(
  "datatype_1" VARCHAR(4),
  "datatype_2" VARCHAR(4),
  "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");

CREATE TABLE "public"."runoob_tbl"( LIKE "public"."runoob_tbl_old"
   INCLUDING COMMENTS INCLUDING CONSTRAINTS INCLUDING DEFAULTS INCLUDING INDEXES
INCLUDING STORAGE);
```

ADD|DROP COLUMN

MySQL and GaussDB(DWS) use different statements for adding and deleting columns. DSC will perform adaptation based on GaussDB features during migration.

CAUTION

GaussDB does not support the update of sequence numbers in table definitions. Temporarily, DSC does not support the complete migration of the FIRST and AFTER features. So as a workaround, it simply deletes the keywords.

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" FLOAT(10),
  "datatype3" FLOAT(20),
  "datatype4" TEXT NOT NULL,
  "datatype5" VARCHAR(4) NOT NULL DEFAULT '2018',
  "datatype6" TIMESTAMP WITHOUT TIME ZONE 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 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;
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;
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;
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;
ALTER TABLE "public"."runoob_alter_test" DROP COLUMN "datatype1_1" RESTRICT;

-- E.
ALTER TABLE "public"."runoob_alter_test" ADD COLUMN "datatype1_1" VARCHAR(200) NOT NULL, ADD
COLUMN "datatype1_2" VARCHAR(200) NOT NULL;
ALTER TABLE "public"."runoob_alter_test" DROP COLUMN "datatype1_1" RESTRICT, DROP COLUMN
"datatype1_2" RESTRICT;
```

MODIFY (Column Modification)

MySQL uses the **MODIFY** keyword to change column data types and set **NOT NULL** constraints. DSC will perform adaptation based on GaussDB 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(100),
  "datatype1" BIGINT,
  "datatype2" BIGINT,
  "datatype3" BIGINT
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype0");

-- A.
ALTER TABLE "public"."runoob_alter_test" ALTER COLUMN "datatype1" SET DATA TYPE SMALLINT;

-- B.
ALTER TABLE "public"."runoob_alter_test" ALTER COLUMN "datatype1" SET DATA TYPE SMALLINT, ALTER
COLUMN "datatype1" SET NOT NULL;

-- C.
ALTER TABLE "public"."runoob_alter_test" ALTER COLUMN "datatype1" SET DATA TYPE SMALLINT, ALTER
COLUMN "datatype1" SET NOT NULL;
```



```
-- D.
ALTER TABLE "public"."runoob_alter_test" ALTER COLUMN "datatype1" SET DATA TYPE SMALLINT, ALTER
COLUMN "datatype1" SET NOT NULL;
```

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 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(128),
  "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" RENAME COLUMN "datatype1" TO "datatype1new";
ALTER TABLE "public"."runoob_alter_test" ALTER COLUMN "datatype1new" SET DATA TYPE VARCHAR(50);

-- B.
ALTER TABLE "public"."runoob_alter_test" RENAME COLUMN "datatype2" TO "datatype2new";
ALTER TABLE "public"."runoob_alter_test" ALTER COLUMN "datatype2new" SET DATA TYPE VARCHAR(50),
ALTER COLUMN "datatype2new" SET NOT NULL;

-- C.
ALTER TABLE "public"."runoob_alter_test" RENAME COLUMN "datatype3" TO "datatype3new";
ALTER TABLE "public"."runoob_alter_test" ALTER COLUMN "datatype3new" SET DATA TYPE VARCHAR(100);

-- D.
ALTER TABLE "public"."runoob_alter_test" RENAME COLUMN "datatype4" TO "datatype4new";
ALTER TABLE "public"."runoob_alter_test" ALTER COLUMN "datatype4new" SET DATA TYPE VARCHAR(50);
```


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 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 "public"."runoob_alter_test"
(
  "datatype1" SERIAL NOT NULL,
  "datatype2" FLOAT(10),
  "datatype3" FLOAT(20),
  "datatype4" TEXT NOT NULL,
  "datatype5" VARCHAR(4) NOT NULL DEFAULT '2018',
  "datatype6" TIMESTAMP WITHOUT TIME ZONE 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 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;
```

DROP (Table Deletion)

Both GaussDB(DWS) and MySQL can use the **DROP** statement to delete tables, but GaussDB(DWS) does not support the **RESTRICT | CASCADE** keyword in **DROP**. 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 "public"."express_elb_server"
(
  "runoob_id" VARCHAR(10),
  "runoob_title" VARCHAR(100) NOT NULL,
  "runoob_author" VARCHAR(40) NOT NULL,
  "submission_date" VARCHAR(10)
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("runoob_id");
DROP TABLE "public"."express_elb_server";
```

TRUNCATE (Table Deletion)

In MySQL, the **TABLE** keyword can be omitted when the **TRUNCATE** statement is used to delete table data. GaussDB does not support this usage. In addition, DSC will add **CONTINUE IDENTITY RESTRICT** during **TRUNCATE** 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;
```

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 "public"."runoob_alter_test"
(
```



```
  "datatype1" SERIAL NOT NULL,
  "datatype2" FLOAT(10),
  "datatype4" TEXT NOT NULL,
  "datatype5" VARCHAR(4) NOT NULL DEFAULT '2018',
  "datatype6" TIMESTAMP WITHOUT TIME ZONE NOT NULL,
  "datatype7" CHAR NOT NULL DEFAULT '',
  "datatype8" VARCHAR(50),
  "datatype9" VARCHAR(50) 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.
```

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 features during migration.

CAUTION

Currently, DSC does not support original table names prefixed with **DATABASE/SCHEMA.**

1. MySQL uses 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";
```

6.11.4 Unique Indexes

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. If there is an inline unique index, DSC will delete it directly.

   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 "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 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 "public"."runoob_alter_test"
(
  "datatype1" INTEGER,
  "datatype2" FLOAT(10),
  "datatype3" FLOAT(20)
)
  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 "public"."runoob_alter_test"
(
  "datatype1" INTEGER,
  "datatype2" FLOAT(10),
  "datatype3" FLOAT(20),
  "datatype4" TEXT NOT NULL,
  "datatype5" VARCHAR(4) 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 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(11) NOT NULL,
     "table01_theme" VARCHAR(100) DEFAULT NULL,
     "author_name" CHAR(10) DEFAULT NULL,
     "author_id" INTEGER(11) 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");
   ```


6.11.5 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 features.

1. Inline normal/prefix index

   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 "public"."runoob_datatype_test"
   (
     "id" SERIAL PRIMARY KEY,
     "name" VARCHAR(128) 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(20) NOT NULL,
     "user_name" CHAR(20) DEFAULT NULL,
     "detail" VARCHAR(100) 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 "public"."customer"
   (
   ```



```
  "name" VARCHAR(64) 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");
```

6.11.6 Hash Indexes

GaussDB(DWS) does not support hash indexes. DSC will replace these indexes with normal indexes based on GaussDB features.

1. Inline hash 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 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(100) NOT NULL,
    "theme" VARCHAR(200) DEFAULT NULL,
    "send_id" INTEGER(11) DEFAULT NULL,
    "send_name" CHAR(20) 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 "public"."runoob_alter_test"
   (
    "datatype1" SERIAL NOT NULL,
    "datatype2" FLOAT(10),
    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(30) NOT NULL,
 "iname" VARCHAR(30) 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");
```

6.11.7 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 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(100) NOT NULL,
 "theme" VARCHAR(200) DEFAULT NULL,
 "send_id" INTEGER(11) DEFAULT NULL,
 "send_name" CHAR(20) 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 "public"."runoob_alter_test"
(
 "datatype1" SERIAL NOT NULL,
 "datatype2" FLOAT(10),
 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_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)
);


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(20) NOT NULL,
 "user_name" CHAR(20) DEFAULT NULL,
 "detail" VARCHAR(100) 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");
```

6.11.8 Spatial Indexes

GaussDB(DWS) does not support spatial indexes. DSC will perform adaptation based on GaussDB 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" CIRCLE 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" CIRCLE 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" CIRCLE 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");
```

6.11.9 Delete an Index

MySQL supports both **DROP INDEX** and **ALTER TABLE DROP INDEX** for deleting indexes. DSC will perform adaptation based on GaussDB 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(11) NOT NULL,
 "demand_name" CHAR(100) NOT NULL,
 "theme" VARCHAR(200) DEFAULT NULL,
 "send_id" INTEGER(11) DEFAULT NULL,
 "send_name" CHAR(20) 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 "demand_name_index" RESTRICT;

CREATE INDEX "send_id_index" ON "public"."test_create_table03" USING BTREE ("send_id");
DROP INDEX "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(11) NOT NULL,
 "demand_name" CHAR(100) NOT NULL,
 "theme" VARCHAR(200) DEFAULT NULL,
 "send_id" INTEGER(11) DEFAULT NULL,
 "send_name" CHAR(20) 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 "test_create_table03_name_index" RESTRICT;
```

6.11.10 Comments

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 "public"."runoob_tbl"
(
  "runoob_id" VARCHAR,
  "runoob_title" VARCHAR(100) NOT NULL,
  "runoob_author" VARCHAR(40) NOT NULL,
  "submission_date" VARCHAR
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("runoob_id");
```

6.11.11 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";
```

6.11.12 Data Manipulation Language (DML)

This section describes the migration syntax of MySQL DML. The migration syntax determines how the keywords and features are migrated.

For details, see the following topics:

- **INSERT**
- **UPDATE**
- **REPLACE**

INSERT

In MySQL, **INSERT** allows the following keywords: **HIGH_PRIORITY**, **LOW_PRIORITY**, **PARTITION**, **DELAYED**, **IGNORE**, **VALUES**, and **ON DUPLICATE KEY UPDATE**. GaussDB(DWS) does not support these keywords, and DSC will convert them.

1. **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);
   ```

2. **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);
```

3. **PRATITION**

   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);
```

4. **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);
```

5. **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);test
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);
```

6. **VALUES**

   INSERT statements use the VALUES syntax can insert multiple lines, separated by commas.

   Input

   ```
   INSERT INTO  exmp_tb1 (tb1_name,tb1_sex,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_sex","tb1_address","tb1_number") VALUES
   ('David','male','NewYork','01015827875');
   INSERT INTO "public"."exmp_tb1" ("tb1_name","tb1_sex","tb1_address","tb1_number") VALUES
   ('Rachel','female','NewYork','01015827749');
   INSERT INTO "public"."exmp_tb1" ("tb1_name","tb1_sex","tb1_address","tb1_number") VALUES
   ('Monica','female','NewYork','010158996743');
   ```

7. **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);
```

8. **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');
   ```

UPDATE

In MySQL, **UPDATE** allows the following keywords: **LOW_PRIORITY**, **ORDER BY**, **LIMIT**, and **IGNORE**. GaussDB(DWS) does not support these keywords, and DSC will convert them.

1. **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;
   ```

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;
   ```

3. **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;
```

4. **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;
   ```

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.

1. **LOW_PRIORITY**

   MySQL REPLACE supports the use of LOW_PRIORITY, which is converted by the Migration tool.

   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);
```

2. **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);
```

3. **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);
```


4. **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, insertion will be performed.
Replace INTO  exmp_tb1 (tb1_id,tb1_name,tb1_sex,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_sex,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_sex,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, insertion will be performed.
INSERT INTO "public"."exmp_tb1" ("tb1_id","tb1_name","tb1_sex","tb1_address","tb1_number")
VALUES (17,'David','male','NewYork11','01015827875');
INSERT INTO "public"."exmp_tb1" ("tb1_id","tb1_name","tb1_sex","tb1_address","tb1_number")
VALUES (18,'Rachel','female','NewYork22','01015827749');
INSERT INTO "public"."exmp_tb1" ("tb1_id","tb1_name","tb1_sex","tb1_address","tb1_number")
VALUES (20,'Monica','female','NewYork','010158996743');
INSERT INTO "public"."exmp_tb1" ("tb1_id","tb1_name","tb1_sex","tb1_address","tb1_number")
VALUES (17,'David1','male','NewYork11','01015827875');
INSERT INTO "public"."exmp_tb1" ("tb1_id","tb1_name","tb1_sex","tb1_address","tb1_number")
VALUES (21,'Rachel','female','NewYork22','01015827749');
INSERT INTO "public"."exmp_tb1" ("tb1_id","tb1_name","tb1_sex","tb1_address","tb1_number")
VALUES (22,'Monica','female','NewYork','010158996743');
INSERT INTO "public"."exmp_tb1"
("tb1_id","tb1_name","tb1_sex","tb1_address","tb1_number","tb1_date") VALUES
(17,'David2',DEFAULT,'NewYork11','01015827875',DEFAULT);
INSERT INTO "public"."exmp_tb1"
("tb1_id","tb1_name","tb1_sex","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_sex","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');
```

5. **SET**

   MySQL REPLACE supports the use of SET settings, which the Migration tool 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);
```

6.11.13 Transaction Management and Database Management

This section describes how to migrate keywords and features related to MySQL transaction and database management.

Transaction Management

1. **TRANSACTION**

   DSC will perform adaptation based on GaussDB 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;
   ```

2. **LOCK**

   DSC will perform adaptation based on GaussDB 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;
```

Database Management

1. **SET CHARACTER**

   DSC will replace MySQL **SET CHARACTER SET** with **SET SESSION NAMES** during migration. The following table lists character set mapping.

   Table 6-28

MySQL CHARACTER SET	GaussDB 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

MySQL CHARACTER SET	GaussDB SESSION NAMES
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';.
```

6.12 DB2 Syntax Migration

6.12.1 Tables

GaussDB Keyword

If a keyword is used as a column name, quotes (") must be added, for example, "order".

DB2 Syntax	Syntax After Migration
CREATE TABLE tbl_bd2 (ORDER NUMBER(10), USER varchar2(30), DATE VARCHAR(10);	CREATE TABLE tbl_bd2 ("ORDER" NUMBER(10), "USER" varchar2(30), "DATE" VARCHAR(10);

Data Type (I)

LONG VARCHAR should be changed to CLOB.

DB2 Syntax	Syntax After Migration
CREATE TABLE tbl_db2 (ID VARCHAR(36), NAME LONG VARCHAR);	CREATE TABLE tbl_db2 (ID VARCHAR(36), NAME CLOB);

LONG VARGRAPHIC.

DB2 Syntax	Syntax After Migration
CREATE TABLE tbl_db2 (ID VARCHAR(36), NAME LONG VARGRAPHIC);	CREATE TABLE tbl_db2 (ID VARCHAR(36), NAME CLOB);

Foreign Key

Below attributes of Foreign key constraint should be commented:

- ON UPDATE RESTRICT
- ENFORCED
- ENABLE QUERY OPTIMIZATION

DB2 Syntax	Syntax After Migration
ALTER TABLE "SCH"."TBL_DB2" ADD CONSTRAINT "Const_Name" FOREIGN KEY("ID") REFRENCES "SCH"."TBL_DB2_1"("ID") ON DELETE CASCADE ON UPDATE RESTRICT ENFORCED ENABLE QUERY OPTIMIZATION;	ALTER TABLE "SCH"."TBL_DB2" ADD CONSTRAINT "Const_Name" FOREIGN KEY("ID") REFRENCES "SCH"."TBL_DB2_1"("ID") ON DELETE CASCADE /*ON UPDATE RESTRICT ENFORCED ENABLE QUERY OPTIMIZATION*/;

Sequence

built-in auto-increment function.

DB2 Syntax	Syntax After Migration
CREATE TABLE tbl_db2 (ID BIGINT NOT NULL GENERATED BY DEFAULT AS IDENTITY (START WITH +1 INCREMENT BY +1 MINVALUE +1 MAXVALUE +9223372036854775807 NO CYCLE CACHE 20 NO ORDER) , NAME VARCHAR2(50), ORDER VARCHAR2(100));	CREATE SEQUENCE mseq_tbl_db2_id START WITH +1 INCREMENT BY +1 MINVALUE +1 MAXVALUE +9223372036854775807 NOCYCLE CACHE 20 NOORDER; CREATE TABLE tbl_db2 (ID BIGINT NOT NULL DEFAULT mseq_tbl_db2_id.NEXTVAL, NAME VARCHAR2(50), "ORDER" VARCHAR2(100));

Tablespace

TABLESPACE for a table to be placed is specified with IN clause.

DB2 Syntax	Syntax After Migration
CREATE TABLE tbl_db2 (ID number(20) NOT NULL DEFAULT IDENTITY.NEXTVAL, NAME VARCHAR2(50)) IN tbs1 ;	CREATE TABLE tbl_db2 (ID number(20) NOT NULL DEFAULT IDENTITY.NEXTVAL, NAME VARCHAR2(50)) TABLESPACE tbs1 ;

Default

WITH DEFAULT is specified to specified DEFAULT value.

DB2 Syntax	Syntax After Migration
CREATE TABLE tbl_db2 (ID number(20) , NAME VARCHAR2(50), STATUS CHAR(1) WITH DEFAULT '0');	CREATE TABLE tbl_db2 (ID number(20) , NAME VARCHAR2(50), STATUS CHAR(1) DEFAULT '0');

DEFAULT specified without value.

DB2 Syntax	Syntax After Migration
CREATE TABLE tbl_db2 (ID number(20) , NAME VARCHAR2(50), STATUS CHAR(1) WITH DEFAULT);	CREATE TABLE tbl_db2 (ID number(20) , NAME VARCHAR2(50), STATUS CHAR(1));

Data Type (II)

CLOB(1048576)

DB2 Syntax	Syntax After Migration
CREATE TABLE tbl_db2 (ID number(20) , NAME VARCHAR2(50), REMARKS CLOB(1048576));	CREATE TABLE tbl_db2 (ID number(20) , NAME VARCHAR2(50), REMARKS CLOB);

BLOB(2048000)

DB2 Syntax	Syntax After Migration
CREATE TABLE tbl_db2 (ID number(20) , NAME VARCHAR2(50), REMARKS BLOB(2048000));	CREATE TABLE tbl_db2 (ID number(20) , NAME VARCHAR2(50), REMARKS BLOB);

LOB Options

LOGGED/UNLOGGED

DB2 Syntax	Syntax After Migration
CREATE TABLE tbl_db2 ("ID" number(20) , "NAME" VARCHAR2(50), "REMARKS" BLOB LOGGED);	CREATE TABLE tbl_db2 ("ID" number(20) , "NAME" VARCHAR2(50), "REMARKS" BLOB /*LOGGED */);

COMPACT/NOT COMPACT

DB2 Syntax	Syntax After Migration
CREATE TABLE tbl_db2 ("ID" number(20) , "NAME" VARCHAR2(50), "REMARKS" BLOB LOGGED NOT COMPACT);	CREATE TABLE tbl_db2 ("ID" number(20) , "NAME" VARCHAR2(50), "REMARKS" BLOB /*LOGGED */ /* NOT COMPACT*/);

Organize By

Organize By

DB2 Syntax	Syntax After Migration
CREATE TABLE tbl_db2 ("ID" number(20) , "NAME" VARCHAR2(50), "REMARKS" BLOB) IN tbs1 ORGANIZE BY ("ID","NAME");	CREATE TABLE tbl_db2 ("ID" number(20) , "NAME" VARCHAR2(50), "REMARKS" BLOB) TABLESPACE tbs1 /*ORGANIZE BY ("ID","NAME")*/;

6.12.2 DML

SELECT

FETCH clause

DB2 Syntax	Syntax After Migration
SELECT empno, ename, deptno FROM emp_t ORDER BY salary FETCH FIRST ROW ONLY ----- SELECT empno, ename FROM emp_t WHERE deptno = 10 ORDER BY salary fetch first 2 rows only;	SELECT empno, ename, deptno FROM emp_t ORDER BY salary LIMIT 1; ----- SELECT empno, ename FROM emp_t WHERE deptno = 10 ORDER BY salary LIMIT 2;

NOTE

The fetch-first-clause sets a maximum number of rows that can be retrieved.

WITH AS

WITH AS with column list

DB2 Syntax	Syntax After Migration
WITH rec (emp_no, emp_name, dept_name, dept_no) AS (SELECT empno, ename, cast('admin' as varchar(90)), 100 AS deptno FROM emp_t) SELECT * FROM rec;	WITH rec AS (SELECT empno AS emp_no, ename AS emp_name, cast('admin' as varchar(90)) AS dept_name, 100 AS dept_no FROM emp_t) SELECT * FROM rec;

WITH AS with VALUES clause

DB2 Syntax	Syntax After Migration
WITH rec (baseschema, basename, baselevel) AS (VALUES(cast('SCOTT' as varchar(90)), cast('EMP' as varchar(90)), 10000)) SELECT owner, table_name, baselevel -1 FROM ALL_TABLES, REC WHERE owner = BASESCHEMA AND table_name = BASENAME;	WITH rec AS (SELECT cast('SCOTT' as varchar(90)) AS baseschema, cast('EMP' as varchar(90)) AS basename, 10000 AS baselevel From dual) SELECT owner, table_name, baselevel -1 FROM ALL_TABLES, REC WHERE owner = BASESCHEMA AND table_name = BASENAME;

Table Function

TABLE function is specified with subquery.

DB2 Syntax	Syntax After Migration

SELECT prod_code, prod_desc, (received_qty-issued_qty) stk
  FROM prod p, TABLE(select prod_code, SUM(received_qty) received_qty FROM prod_recd) r
  , TABLE(select prod_code, SUM(issued_qty) issued_qty FROM prod_issue) i
 WHERE r.prod_code = p.prod_code
  AND i.prod_code = p.prod_code;
	SELECT prod_code, prod_desc, (received_qty-issued_qty) stk
	  FROM prod p, (select prod_code, SUM(received_qty) received_qty FROM prod_recd) r
	  , (select prod_code, SUM(issued_qty) issued_qty FROM prod_issue) i
	 WHERE r.prod_code = p.prod_code
	  AND i.prod_code = p.prod_code;

6.12.3 Index

Reverse Scans

Reverse Scans

DB2 Syntax	Syntax After Migration

CREATE INDEX IDX_1 ON EMP
  (ID ASC) ALLOW REVERSE SCANS;
 CREATE INDEX IDX_1 ON EMP
  (ID ASC) DISALLOW REVERSE SCANS;
	CREATE INDEX IDX_1 ON EMP
	  (ID ASC) /*ALLOW REVERSE SCANS*/;
	 CREATE INDEX IDX_1 ON EMP
	  (ID ASC) /*DISALLOW REVERSE SCANS*/;

Schema

index's schema is different from its table's schema.

DB2 Syntax	Syntax After Migration
CREATE INDEX IDXSC.IDX_1 ON EMP (ID ASC);	CREATE INDEX IDX_1 ON EMP (ID ASC);

6.12.4 NICKNAME

NICKNAME

Synonym is referred to **Nickname** in DB2.

DB2 Syntax	Syntax After Migration
CREATE NICKNAME sc2.emp FOR sc.emp;	CREATE SYNONYM sc2.emp FOR sc.emp;

6.12.5 Statement

CURRENT SCHEMA

SET CURRENT SCHEMA

DB2 Syntax	Syntax After Migration
CREATE NICKNAME sc2.emp FOR sc.emp;	CREATE SYNONYM sc2.emp FOR sc.emp;

GET DIAGNOSTICS with ROW_COUNT

DB2 Syntax	Syntax After Migration
CC_ORDER_HY_CUSTORCOUNT SET ' ||STR_CUST_COUNT||'_COUNT='|| char(CUST_COUNT)||' ,' ||'SELF_UPDATE_TIME ='||''''|| STR_DATE||''''||' ,' ||'SUB_UPDATE_TIME ='||''''|| STR_DATE1||''''||',' ||'REPORT_STATE ='||''''|| STR_CHAR ||'''' ||' WHERE ORG_CODE ='||''''|| TEMP_ORG_CODE||'''' ||' AND Y='||char(TEMP_YEAR)||' AND HY='|| char(TEMP_HY) ||' AND REGION_CODE='||''''|| TEMP_SALE_REG_CODE||''''||' AND BRAND_CODE='||''''||TEMP_BRAND_CODE||'''' ||' AND exists (select 1 from CC_ORDER_HY_CUSTORCOUNT' ||' where ORG_CODE ='||''''|| TEMP_ORG_CODE||'''' ||' AND Y='||char(TEMP_YEAR)||' AND HY='|| char(TEMP_HY) ||' AND REGION_CODE='||''''|| TEMP_SALE_REG_CODE||''''||' AND BRAND_CODE='||''''||TEMP_BRAND_CODE||''''||')';-- PREPARE S1 FROM STR_UPDATE1; EXECUTE S1; -- GET DIAGNOSTICS V_NUMRECORDS = ROW_COUNT;	V_NUMRECORDS = SQL%ROWCOUNT;

6.12.6 System Functions

DAY

Day

DB2 Syntax	Syntax After Migration
SELECT DAYS(doj) FROM emp;	SELECT (TRUNC(doj - TO_DATE('0001/01/01', 'YYYY/MM/DD'))+1) FROM emp;

MONTH

Month

DB2 Syntax	Syntax After Migration
SELECT (MONTH(ORDER_DATE)-1)/6+1 as TEMP_HY;	SELECT (EXTRACT (MONTH FROM ORDER_DATE) -1)/6+1 as TEMP_HY;

YEAR

Year

DB2 Syntax	Syntax After Migration
SELECT YEAR(ORDER_DATE) as TEMP_HY;	SELECT EXTRACT (YEAR FROM ORDER_DATE) as TEMP_HY;

CURRENT DATE

Current date

DB2 Syntax	Syntax After Migration
SELECT CURRENT DATE FROM DUAL;	SELECT CURRENT_DATE FROM DUAL;

CURRENT TIMESTAMP

Current timestamp

DB2 Syntax	Syntax After Migration
SELECT CURRENT TIMESTAMP - 7 DAYS;	SELECT CURRENT_TIMESTAMP - 7 DAYS;

POSSTR

POSSTR

DB2 Syntax	Syntax After Migration
SELECT POSSTR('THIS IS TEST','TEST') FROM DUAL;	SELECT INSTR('THIS IS TEST','TEST') FROM DUAL;

VALUE

Value

DB2 Syntax	Syntax After Migration
Select VALUE('abc','') from dual;	Select Coalesce('abc','') from dual;

Date

The **date** function returns a date using a value.

DB2 Syntax	Syntax After Migration
SELECT org_code, DATE(order_date) FROM view_cc_order WHERE order_date = DATE((SELECT start_date FROM year_week_mark WHERE year=TEMP_YEAR and week=TEMP_WEEK)); --- SELECT deptno, deptname, DATE(SELECT max(doj) FROM emp e WHERE e.deptno = d.deptno) FROM dept d;	SELECT org_code, mig_db2_ext.mig_db2_fn_date(order_date) FROM view_cc_order WHERE order_date = mig_db2_ext.mig_db2_fn_date((SELECT start_date FROM year_week_mark WHERE year=TEMP_YEAR and week=TEMP_WEEK)); --- SELECT deptno, deptname, mig_db2_ext.mig_db2_fn_date((SELECT max(doj) FROM emp e WHERE e.deptno = d.deptno)) FROM dept d;

6.13 Command Reference

6.13.1 Database Schema Conversion

Function

runDSC.sh or **runDSC.bat** is used to migrate schemas and queries of Teradata, Oracle, Netezza, MySQL, and DB2 to GaussDB(DWS).

Format

Linux:

```
./runDSC.sh
--source-db<source-database>
[--input-folder<input-script-path>]
[--output-folder<output-script-path>]
[-application-lang <application-lang>]
[--conversion-type<conversion-type>]
[--log-folder<log-path>]
[--version-number <Gauss Kernel Version>]
[--target-db<target-database>
```

Windows:

```
runDSC.bat
--source-db<source-database>
[--input-folder<input-script-path>]
[--output-folder<output-script-path>]
[-application-lang <application-lang>]
[--conversion-type<conversion-type>]
[--log-folder<log-path>]
[--version-number <Gauss Kernel Version>]
[--target-db<target-database>
```


Parameter Description

Table 6-29 Parameters

Long	Short	Data Type	Description	Value Range	Default Value	Example
--source-db	-S	String	Source database	• Oracle • Teradata • Netezza • MySQL • DB2	N/A	--source-db *Oracle*(or) -S *Oracle*
--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 *bulk* or -M *bulk*
--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	• gaussdbT • gaussdbA	gaussdbA	--target-db gaussdbT (or) *-T gaussdbT*

Usage Guidelines

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 no log path is specified, DSC creates the **log** folder under **TOOL_HOME** to store logs.

Example

```
./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 gaussdbT
```

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)

GaussDB(DWS): Database Creation and Schema Setup

Step 1 Log into 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 -S oracle -M bulk -I <input path>
```

Configuration Details

Step 1 Set the value of **GaussDBSQLExec** to **True**, and update the **gaussdb.properties** file.

Step 2 Create a user (T) and a database (A). Add all schemas.

----End

Verification After Migration

After DSC converts the source sql files, execute the converted files on target gaussdb 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 between gsql and zsql clients

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 or zsql.

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 ****************************
```

6.13.2 Version

Function

The **version** command is used to display the version number of the DSC.

Format

Linux:

```
./runDSC.sh --version
```

Windows:

```
runDSC.bat --version
```

Parameters

None

Usage Guidelines

Linux:

```
./runDSC.sh --version
```

Windows:

```
runDSC.bat --version
```

System Response

```
Version: DSC (Gauss Tools 8.0.1)
```

6.13.3 Help

Function

The **help** command is used to provide the help information for the commands supported by DSC.

Format

Linux:

```
./runDSC.sh --help
```

Windows:

```
runDSC.bat --help
```

Parameter Description

None

Usage Guide

None

Example

Linux:

```
./runDSC.sh --help
```

Windows:

```
runDSC.bat --help
```

System Response

Linux:


```
./runDSC.sh --help
To migrate teradata/oracle/netezza/mysql/db2 database scripts to FusionInsight LibrA
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.


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 FusionInsight LibrA
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.
```

6.14 Log Reference

6.14.1 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

The topics in this section explain the different logs available for the tool.

6.14.2 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 sWX575686
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 sWX575686
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
```

Success Read

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.
```

Success Write

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.
```

6.14.3 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** configuration parameter.

Logging

The DSC saves all log, 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
```

6.15 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 6-30 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. For details, see **Supported Keywords and Features**.

Symptom/Issue	Cause and Solution
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. For details, see **Database Schema Conversion**.
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. For details, see **Database Schema Conversion**.
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. For details, see **Database Schema Conversion**.

Symptom/Issue	Cause and Solution
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 **********

GaussDB(DWS)

Table 6-31 Error Message Reference

Symptom/Issue	Cause and Solution
Cannot create index whose evaluation is enforced to remote nodes.	**Cause**: This issue is because of Gauss limitation that distribution key columns should be super set of the unique index columns. **Solution**: Currently, Gauss does not support.

Symptom/Issue	Cause and Solution
Type "urowid" does not exist	**Cause**: User defined type is used to create the table. **Solution**: Currently, Gauss does not support.
Syntax error occurs at or near **LOCAL**.	**Cause**: Gauss does not support LOCAL keyword in index. **Solution**: Local index needs to be created.
Syntax error occurs at or near **1**.	**Cause**: GaussDB(DWS) does not support additional parameters in index. **Solution**: This needs to be commented.
Syntax error occurs at or near **=**.	**Cause**: GaussDB(DWS) does not support **=** in constraint. **Solution**: Currently, Gauss does not support.

6.16 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*.

6.17 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.

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. It does not require any network or database connection to run. It can be run on any machine that is isolated from any network.

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.	Only the scala2.11 flink 1.12 version is released.
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.	Supported dws-connector-flink versions: Scala2.11: flink 1.12, 1.13 Scala2.12: flink 1.12, 1.13, 1.15

7.2 dws-client

Description

DWS Client is a high-performance and convenient data import tool based on DWSJDBC. Ensure that JDBC can be connected when using DWS Client.

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>1.0</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 an upsert 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 upsert() throws DwsClientException {
        getClient().upsert("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 are classified into the following types:

1. InvalidException is not thrown and is triggered when the request parameter is invalid.

2. DwsClientException encapsulates all exceptions, including the parsed code and original exceptions.

3. DwsClientRecordException is an extension to DwsClientException. It includes the data sets 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
driverName	JDBC driver of the database	com.huawei.gauss200.jdbc.Driver	
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 is greater than or equal to 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 is greater than or equal to 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: 1. If there is a primary key and the data volume is less than the value of **copyWriteBatchSize**, the **UPSERT** method is used to import the data to the database. Otherwise, the **COPY** + **UPSERT** method is used. 2. If there is no primary key and the data volume is less than the value of **copyWriteBatchSize**, use the **INSERT INTO** method to import data to the database. Otherwise, use the **COPY** method. COPY_MERGE: 1. Use the **COPY** +**MERGE** method to import data to the database. 2. If there is no primary key, use the **COPY** method to import data to the database. **COPY_UPSERT**: 1. If there is no primary key, use the **COPY** method to import data to the database.	AUTO	

Parameter	Description	Default Value	Supported Versions
	2. If there is a primary key, use the **COPY** + **UPSERT** method to import data to the database. **UPSERT**: 1. If there is no primary key, use the **INSERT INTO**method to import data to the database. 2. If there is a primary key, use the **UPSERT**method to import data to the database.		

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. Asynchronous tasks are submitted by table. Multiple tables can be executed concurrently. For operations that involve different columns in a table. Operations involving the same columns are classified into one type. Different types of operations can work concurrently during import. Set this parameter to improve the throughput.	3	
logSwitch	Log switch. If this function is enabled, detailed process logs are recorded for debugging or fault locating.	false	
logDataTables	Tables whose data needs to be printed during data import to the database for data comparison during fault locating.	-	
flushSuccessFunc-tion	Callback function after data is successfully imported to the database	-	

Parameter	Description	Default Value	Supported Versions
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 preferabbly use background threads to submit import tasks, rather than using service threads.	1	

Parameter	Description	Default Value	Supported Versions
tableConfig	If multiple tables share one client, you may need to set **conflictStrategy**, **writeMode**, **copyWriteBatchSize**, **autoFlushMaxIntervalMs**, **autoFlushBatchSize** and **batchOutWeighRatio** to different values based on different tables. This parameter can be used to configure the preceding parameters at the table level and make them take effect globally for the tables that are not configured.	-	

7.3 dws-connector-flink

Description

dws-connector-flink is a tool used to connect dwsclient to flink. The tool encapsulates dwsClient. Its overall import capability is the same as that of dwsClient. Currently, only the **DynamicTableSourceFactory** and **DynamicTableSinkFactory** interfaces are implemented. The **CatalogFactory** interface is not implemented. Therefore, catalogs are not supported.

How to Use

- Through a JAR package

  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>1.0</version>
</dependency>
```


- Through Flink SQL

  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. The following lists the latest download addresses of Scala and Flink versions supported by the dws-connector-flink package with dependencies:

  - **dws-connector-flink_2.11_1.12-1.0.2-all.jar**
  - **dws-connector-flink_2.11_1.13-1.0.2-all.jar**
  - **dws-connector-flink_2.12_1.12-1.0.2-all.jar**
  - **dws-connector-flink_2.12_1.13-1.0.2-all.jar**
  - **dws-connector-flink_2.12_1.15-1.0.2-all.jar**

Scenarios & Usage

- **Sink Mode**

  A DwsSink API is provided to quickly build a sink. An example is as follows:

```
SinkFunction<EventLog> dwsSink = DwsSink.sink(DwsConfig.builder()
                .withUrl("")
                .withPassword("")
                .withUsername("").build(), null, (DwsInvokeFunction<EventLog>) (eventLog, dwsClient,
context) -> {
            dwsClient.upsert("test.test")
                    .setObject("id", eventLog.getGuid())
                    .setObject("name", eventLog.getEventId(), true)
                    .commit();
        });
        DataStreamSource<EventLog> source = evn.addSource(new LogSourceFunction());
        source.addSink(dwsSink);
        evn.execute();
```

  API description:

```
public static <T> SinkFunction<T> sink(DwsConfig config, JSONObject context, DwsInvokeFunction<T>
invoke);
```

  - **config** is a parameter of dwsClient, which is the same as that of dwsClient.
  - **context** is a global context provided for operations such as cache. It can be specified during dwsClient 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 dwsClient 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;
```

- **SQL Mode**

  a. Create a Flink table:

```
tableEnvironment.executeSql("create table dws_test (" +
    "   id int,\n" +
    "   name STRING,\n" +
    "   PRIMARY KEY (id) NOT ENFORCED" +
    ") WITH (\n" +
    "   'connector' = 'dws',\n" +
    "   'url' = 'jdbc:postgresql://****/gaussdb',\n" +
    "   'tablename' = 'test.test'," +
```



```
  "   'username'='dbadmin'," +
  "   'password'='***'" +
   ")");
```

b. Write data in the data source to the test table.

```
tableEnvironment.executeSql("insert into dws_test select guid as id,eventId as name from kafka_event_log")
```

c. Query data from the test table.

```
tableEnvironment.executeSql("select name from dws_test where id = 100 limit 10").print();
```

flinkSQL Configuration Parameters

Table 7-2 Database Configuration

Parameter Description	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	The GaussDB(DWS) table	-

Table 7-3 Connection configuration

Parameter	Description	Default Value
connectionSize	Number of concurrent requests at dwsClient initialization	1
connectionMaxUseTime-Seconds	Number of seconds after which a connection is forcibly released. The unit is second.	3600 (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 7-4 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. 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.	auto
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

Parameter	Description	Default Value
autoFlushMaxInterval	Maximum interval for automatic database update (duration for forming a batch), in seconds.	5
copyWriteBatchSize	When **writeMode** is set to **auto**, the batch size in the **COPY** method is used.	1000
ignoreDelete	Flink tasks generate deletion operations. This parameter specifies if to ignore the deletion operations.	false
ignoreNullWhenUpdate	Indicates if 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

Table 7-5 Query parameters

Parameter	Mandatory	Description	Default Value
fetchSize	No	The fetchSize parameter in the JDBC statement is used to control the number of records returned by the database.	1000

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

  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.

- --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

  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

  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
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

Parameter	Recommended Value
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-8.1.3-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-8.1.3-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
```

8.8 gs_sshexkey

Context

During cluster installation, you need to execute commands and transfer files among hosts in the cluster. Therefore, mutual trust relationships must be established among the hosts before the installation. **gs_sshexkey**, provided by GaussDB(DWS), helps you establish such relationships.

NOTICE

The mutual trust relationships among **root** users have security risks. You are advised to delete the mutual trust relationships among **root** users once operations completed.

Prerequisites

- The SSH service has been enabled.
- You have verified that SSH ports will not be disabled by firewalls.
- Each host name and IP address have been correctly configured in the XML file.

- Communication among all the hosts must be normal.
- If the mutual trust relationship is to be established for common users, the same user needs to be created and password set on each host.
- If the SELinux OS has been installed and started on each host, the security context of the **/root** directory needs to be set to the default value **system_u:object_r:home_root_t:s0** and that of the **/home** directory set to the default value **system_u:object_r:admin_home_t:s0**.

  To check whether the SELinux OS has been installed and started, run the **getenforce** command. If the command output is **Enforcing**, the SELinux OS has been installed and started.

  To check the security contexts of the directories, run the following commands:

  ```
  ls -ldZ  /root | awk '{print $4}'
  ls -ldZ  /home | awk '{print $4}'
  ```

  To restore the security contexts of the directories, run the following commands:

  ```
  restorecon -r -vv /home/
  restorecon -r -vv /root/
  ```

Syntax

- Establish mutual trust relationships.
  ```
  gs_sshexkey -f HOSTFILE [-W PASSWORD] [...] [--skip-hostname-set] [-l LOGFILE]
  ```
- Show help information.
  ```
  gs_sshexkey -? | --help
  ```
- Show version number.
  ```
  gs_sshexkey -V | --version
  ```

Parameter Description

- -f

  Lists the IP addresses of all the hosts among which mutual trust relationships need to be established.

  NOTE

  Ensure that **hostfile** contains only correct IP addresses and no other information.

- -W, --password=PASSWORD

  Specifies the password of the user who will establish mutual trust relationships. If this parameter is not specified, you will be prompted to enter the password when the mutual trust relationship is established. If the password of each host is different, you need to specify multiple **-W** parameters. The password sequence must correspond to the IP address sequence. In interactive mode, you need to enter the password of the host in sequence.

  NOTE

  The password cannot contain the following characters: ;'$

- -l

  Specifies the path of saving log files.

  Value range: any existing, accessible absolute path

- --skip-hostname-set

  Specifies whether to write the mapping relationship between the host name and IP address of the **-f** parameter file to the **/etc/hosts** file. If this parameter is specified, the relationship is not written to the file.

- -?, --help

  Shows help information.

- -V, --version

  Shows version number.

Examples

The following examples describe how to establish mutual trust relationships for user **root**:

- In non-interactive mode, if the user passwords are the same, run the following commands to establish mutual trust.

  Gauss@123 indicates the password of user **root**.

  ```
  ./gs_sshexkey -f /opt/software/hostfile -W Gauss@123
  Checking network information.
  All nodes in the network are Normal.
  Successfully checked network information.
  Creating SSH trust.
  Creating the local key file.
  Appending local ID to authorized_keys.
  Successfully appended local ID to authorized_keys.
  Updating the known_hosts file.
  Successfully updated the known_hosts file.
  Appending authorized_key on the remote node.
  Successfully appended authorized_key on all remote node.
  Checking common authentication file content.
  Successfully checked common authentication content.
  Distributing SSH trust file to all node.
  Successfully distributed SSH trust file to all node.
  Verifying SSH trust on all hosts.
  Successfully verified SSH trust on all hosts.
  Successfully created SSH trust.
  ```

- In non-interactive mode, if the user passwords are different, run the following commands to establish mutual trust.

  Gauss@234 indicates the **root** password of the first host in the host list, and **Gauss@345** indicates the **root** password of the second host in the host list.

  ```
  ./gs_sshexkey -f /opt/software/hostfile -W Gauss@123 -W Gauss@234 -W Gauss@345
  Checking network information.
  All nodes in the network are Normal.
  Successfully checked network information.
  Creating SSH trust.
  Creating the local key file.
  Appending local ID to authorized_keys.
  Successfully appended local ID to authorized_keys.
  Updating the known_hosts file.
  Successfully updated the known_hosts file.
  Appending authorized_key on the remote node.
  Successfully appended authorized_key on all remote node.
  Checking common authentication file content.
  Successfully checked common authentication content.
  Distributing SSH trust file to all node.
  Successfully distributed SSH trust file to all node.
  Verifying SSH trust on all hosts.
  Successfully verified SSH trust on all hosts.
  Successfully created SSH trust.
  ```


- In interactive mode, if the user passwords are the same, run the following commands to establish mutual trust.

```
gs_sshexkey -f /opt/software/hostfile
Please enter password for current user[root].
Password:
Checking network information.
All nodes in the network are Normal.
Successfully checked network information.
Creating SSH trust.
Creating the local key file.
Appending local ID to authorized_keys.
Successfully appended local ID to authorized_keys.
Updating the known_hosts file.
Successfully updated the known_hosts file.
Appending authorized_key on the remote node.
Successfully appended authorized_key on all remote node.
Checking common authentication file content.
Successfully checked common authentication content.
Distributing SSH trust file to all node.
Successfully distributed SSH trust file to all node.
Verifying SSH trust on all hosts.
Successfully verified SSH trust on all hosts.
Successfully created SSH trust.
```

- In interactive mode, if the user passwords are different, run the following commands to establish mutual trust.

```
gs_sshexkey -f /opt/software/hostfile
Please enter password for current user[root].
Password:
Notice :The password of some nodes is incorrect.
Please enter password for current user[root] on the node[10.180.10.112].
Password:
Please enter password for current user[root] on the node[10.180.10.113].
Password:
Checking network information.
All nodes in the network are Normal.
Successfully checked network information.
Creating SSH trust.
Creating the local key file.
Appending local ID to authorized_keys.
Successfully appended local ID to authorized_keys.
Updating the known_hosts file.
Successfully updated the known_hosts file.
Appending authorized_key on the remote node.
Successfully appended authorized_key on all remote node.
Checking common authentication file content.
Successfully checked common authentication content.
Distributing SSH trust file to all node.
Successfully distributed SSH trust file to all node.
Verifying SSH trust on all hosts.
Successfully verified SSH trust on all hosts.
Successfully created SSH trust.
```

