数据仓库服务（DWS）

工具指南

天翼云科技有限公司

目 录

1 工具简介

本手册介绍数据仓库服务的工具使用，提供了客户端工具和服务端工具，客户端工具如表 1-1 所示，服务端工具如表 1-2 所示。

客户端工具：参见 2 客户端工具下载获取。

服务端工具：位于安装数据库服务器的$GPHOME/script 和$GAUSSHOME/bin 路径下。

表1-1 客户端工具

工具名称	工具简介
gsql	一款运行在 Linux 操作系统的命令行工具，用于连接 DWS 集群中的数据库，并对数据库进行操作和维护。
Data Studio	用于连接数据库的客户端工具，有着丰富的 GUI 界面，能够管理数据库和数据库对象，编辑、运行、调试 SQL 脚本，查看执行计划等。Data Studio 工具可运行在 32 位或 64 位 windows 操作系统上，解压软件包后免安装即可使用。
GDS	一款运行在 Linux 操作系统的命令行工具，通过和外表机制的配合，实现数据的高速导入导出。GDS 工具包需要安装在数据源文件所在的服务器上，数据源文件所在的服务器称为数据服务器，也叫 GDS 服务器。
DSC	用于将 Teradata 或 Oracle 数据库中的 sql 脚本迁移为适用于 GaussDB(DWS)的 sql 脚本，便于在 GaussDB(DWS)中重建数据库。DSC 工具是运行在 Linux 操作系统的命令行工具，解压软件包免安装即可使用。

表1-2 服务端工具

工具名称	简介
7.1 gs_dump	gs_dump 是一款用于导出数据库相关信息的工具，支持导出完整一致的数据库对象（数据库、模式、表、视图等）数

工具名称	简介
	据，同时不影响用户对数据库的正常访问。
7.2 gs_dumpall	gs_dumpall 是一款用于导出数据库相关信息的工具，支持导出完整一致的集群数据库所有数据，同时不影响用户对数据库的正常访问。
7.3 gs_restore	gs_restore 是 GaussDB(DWS)提供的针对 gs_dump 导出数据的导入工具。通过此工具可由 gs_dump 生成的导出文件进行导入。
7.4 gds_check	gds_check 用于对 GDS 部署环境进行检查，包括操作系统参数、网络环境、磁盘占用情况等，也支持对可修复系统参数的修复校正，有助于在部署运行 GDS 时提前发现潜在问题，提高执行成功率。
7.5 gds_install	gds_install 是用于批量安装 gds 的脚本工具，可大大提高 GDS 部署效率。
7.6 gds_uninstall	gds_uninstall 是用于批量卸载 GDS 的脚本工具。
7.7 gds_ctl	gds_ctl 是一个批量控制 GDS 启停的脚本工具，一次执行可以在多个节点上启动/停止相同端口的 GDS 服务进程，并在启动时为每一个进程设置看护程序，用于看护 GDS 进程。
7.8 gs_sshexkey	集群在安装过程中，需要在集群中的节点间执行命令，传送文件等操作。gs_sshexkey 工具来帮助用户建立互信，需要确保互信是连通的。

2 客户端工具下载

步骤 1 登录 GaussDB(DWS) 管理控制台。

步骤 2 在左侧导航栏中，单击“连接管理”。

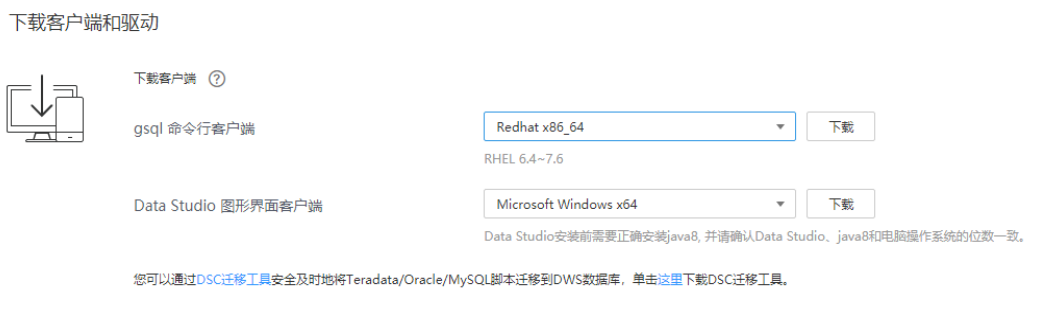
步骤 3 在“下载客户端和驱动”区域，请根据计算机的操作系统，选择对应版本的工具进行下载。

此处可以下载以下工具：

- gsql 命令行客户端：gsql 工具包中包含了 gsql 客户端工具、GDS 并行数据加载工具以及 gs_dump、gs_dumpall 和 gs_restore 工具。
- Data Studio 图形界面客户端
- DSC 迁移工具

对于 gsql 客户端工具、Data Studio 客户端工具，存在多个历史版本，单击“历史版本”可根据集群版本下载相应版本的工具。GaussDB(DWS) 集群可向下兼容 gsql、Data Studio 工具，建议按集群版本下载配套的工具版本。

图2-1 下载客户端



----结束

3 gsql 命令行客户端工具

3.1 gsql 概述

基本功能

- **连接数据库：** 通过 gsql 客户端远程连接 GaussDB(DWS)数据库。

  说明

  gsql 创建连接时，会有 5 分钟超时时间。如果在这个时间内，数据库未正确地接受连接并对身份进行认证，gsql 将超时退出。

  针对此问题，可以参考常见问题处理。

- **执行 SQL 语句：**支持交互式地键入并执行 SQL 语句，也可以执行一个文件中指定的 SQL 语句。
- **执行元命令：**元命令可以帮助管理员查看数据库对象的信息、查询缓存区信息、格式化 SQL 输出结果，以及连接到新的数据库等。元命令的详细说明请参见 3.5 元命令参考。

高级特性

gsql 的高级特性如表 3-1 所示。

表3-1 gsql 高级特性

特性名称	描述
变量	gsql 提供类似于 Linux 的 shell 命令的变量特性，可以使用 gsql 的元命令\set 设置一个变量，格式如下： `\set varname value` 要删除一个变量请使用如下方式： `\unset varname` 说明 • 变量只是简单的名称/值对，值的长度由特殊变量 VAR_MAX_LENGTH 决定，详细参见表 3-2。

特性名称	描述
	• 变量名称必须由字母（包括非拉丁字母）、数字和下划线组成，且对大小写敏感。 • 如果使用\set varname 的格式（不带第二个参数），则只是设置这个变量而没有给变量赋值。 • 可以使用不带参数的\set 来显示所有变量的值。 变量的示例和详细说明请参见变量。
SQL 代换	利用 gsql 的变量特性，可以将常用的 SQL 语句设置为变量，以简化操作。 SQL 代换的示例和详细说明请参见 SQL 代换。
自定义提示符	gsql 使用的提示符支持用户自定义。可以通过修改 gsql 预留的三个变量 PROMPT1、PROMPT2、PROMPT3 来改变提示符。 这三个变量的值可以用户自定义，也可以使用 gsql 预定义的值。详细请参见提示符。
客户端操作历史记录	gsql 支持客户端操作历史记录，当客户端连接时指定“-r”参数，此功能被打开。可以通过\set 设置记录历史的条数，例如，\set HISTSIZE 50，将记录历史的条数设置为 50，\set HISTSIZE 0，不记录历史。 说明 • 客户端操作历史记录条数默认设置为 32 条，最多支持记录 500 条。当客户端交互式输入包含中文字符时，只支持 UTF-8 的编码环境。 • 出于安全考虑，将包含 PASSWORD、IDENTIFIED 敏感词的记录识别为敏感信息，不会记录到历史信息中，即不能通过上下翻回显。

- 变量

  可以使用 gsql 元命令\set 设置一个变量。例如把变量 foo 的值设置为 bar：

  ```
  \set foo bar
  ```

  要引用变量的值，在变量前面加冒号。例如查看变量的值：

  ```
  \echo :foo
  bar
  ```

  这种变量的引用方法适用于规则的 SQL 语句和元命令。

  在使用命令行参数--dynamic-param（详见表 3-7），或设置特殊变量 DYNAMIC_PARAM_ENABLE（详见表 3-2）为 true 时，可通过执行 SQL 语句设置变量。变量名为 SQL 执行结果的列名，也可使用${}方式引用。例如：

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

```
\echo ${Name}
Jack
```

上述示例中，通过 SELECT 语句执行设置 Name 变量，并在后面使用${}的引用方式获得变量 Name 的值。示例中通过特殊变量 DYNAMIC_PARAM_ENABLE 控制这一功能，也可通过命令行参数--dynamic-param 控制，如 gsql -d　postgres -p 25308 --dynamic-param -r。

说明

- SQL 执行失败时，不设置变量。
- SQL 执行结果为空，以列名设置变量，赋值空字符串。
- SQL 执行结果为一条记录，以列名设置变量，赋值对应字符串。
- SQL 执行结果为多条记录，以列名设置变量，使用特定字符串拼接，然后赋值。特定字符串由特殊变量 RESULT_DELIMITER（详见表 3-2）控制，默认为 “,”。

执行 SQL 语句设置变量示例：

```
\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
--  执行失败时，不设置变量
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}

--  执行结果为多条记录时，使用特定字符串拼接
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

-- 执行结果为一条记录时
SELECT id, name FROM student where id = 1;
 id | name
----+------
  1 | Jack
(1 row)

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

-- 执行结果为空时，赋值空字符串
SELECT id, name FROM student where id = 4;
```

```
 id | name
----+------
(0 rows)

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

gsql 预定义了一些特殊变量，同时也规划了变量的取值。为了保证和后续版本最大限度地兼容，请避免以其他目的使用这些变量。所有特殊变量见表 3-2。

说明

- 所有特殊变量都由大写字母、数字和下划线组成。
- 要查看特殊变量的默认值，请使用元命令**\echo :***varname*（例如\echo :DBNAME）。

表3-2 特殊变量设置

变量	设置方法	变量说明
DBNAME	`\set DBNAME dbname`	当前连接的数据库的名字。每次连接数据库时都会被重新设置。
ECHO	`\set ECHO all | queries`	• 如果设置为 all，只显示查询信息。设置为 all 等效于使用 gsql 连接数据库时指定-a 参数。 • 如果设置为 queries，显示命令行和查询信息。等效于使用 gsql 连接数据库时指定-e 参数。
ECHO_HIDDEN	`\set ECHO_HIDDEN on | off | noexec`	当使用元命令查询数据库信息（例如\dg）时，此变量的取值决定了查询的行为： • 设置为 on，先显示元命令实际调用的查询语句，然后显示查询结果。等效于使用 gsql 连接数据库时指定-E 参数。 • 设置为 off，则只显示查询结果。 • 设置为 noexec，则只显示查询信息，不执行查询操作。
ENCODING	`\set ENCODING encoding`	当前客户端的字符集编码。
FETCH_COUNT	`\set FETCH_COUNT variable`	• 如果该变量的值为大于 0 的整数，假设为 n，则执行 SELECT 语句时每次从结果集中取 n 行到缓存并显示到屏幕。 • 如果不设置此变量，或设置的值小于等于 0，则执行 SELECT 语句时一次性把结果都取到缓存。 说明 设置合理的变量值，将减少内存使用量。一般来说，设为 100 到 1000 之间的值比较合理。

变量	设置方法	变量说明
HISTCONTROL	`\set HISTCONTROL ignorespace | ignoredups | ignoreboth | none`	• ignorespace：以空格开始的行将不会写入历史列表。 • ignoredups：与以前历史记录里匹配的行不会写入历史记录。 • ignoreboth、none 或者其他值：所有以交互模式读入的行都被保存到历史列表。 **说明** none 表示不设置 HISTCONTROL。
HISTFILE	`\set HISTFILE filename`	此文件用于存储历史名列表。缺省值是~/.bash_history。
HISTSIZE	`\set HISTSIZE size`	保存在历史命令里命令的个数。缺省值是500。
HOST	`\set HOST hostname`	已连接的数据库主机名称。
IGNOREEOF	`\set IGNOREEOF variable`	• 若设置此变量为数值，假设为 10，则在 gsql 中输入的前 9 次 EOF 字符（通常是 Ctrl+C 组合键）都会被忽略，在第 10 次按 Ctrl+C 才能退出 gsql 程序。 • 若设置此变量为非数值，则缺省为 10。 • 若删除此变量，则向交互的 gsql 会话发送一个 EOF 终止应用。
LASTOID	`\set LASTOID oid`	最后影响的 oid 值，即为从一条 INSERT 或 lo_import 命令返回的值。此变量只保证在下一条 SQL 语句的结果显示之前有效。
ON_ERROR_ROLLBACK	`\set ON_ERROR_ROLLBACK on | interactive | off`	• 如果是 on，当一个事务块里的语句产生错误的时候，这个错误将被忽略而事务继续。 • 如果是 interactive，这样的错误只是在交互的会话里忽略。 • 如果是 off（缺省），事务块里一个语句生成的错误将会回滚整个事务。on_error_rollback-on 模式是通过在一个事务块的每个命令前隐含地发出一个 SAVEPOINT 的方式工作的，在发生错误的时候回滚到该事务块。
ON_ERROR_STOP	`\set ON_ERROR_STOP on | off`	• on：命令执行错误时会立即停止，在交互模式下，gsql 会立即返回已执行命令的结果。 • off（缺省）：命令执行错误时将会跳过错误继续执行。
PORT	`\set PORT port`	正连接数据库的端口号。

变量	设置方法	变量说明
USER	`\set USER username`	当前用于连接的数据库用户。
VERBOSITY	`\set VERBOSITY terse | default | verbose`	这个选项可以设置为值 terse、default、verbose 之一以控制错误报告的冗余行。 • terse：仅返回严重且主要的错误文本以及文本位置（一般适合于单行错误信息）。 • default：返回严重且主要的错误文本及其位置，还包括详细的错误细节、错误提示（可能会跨越多行）。 • verbose：返回所有的错误信息。
VAR_NOT_FOUND	\set VAR_NOT_FOUND default | null | error	可以设置为 default、null、error 之一以控制引用变量不存在时的处理方式。 • default：不做变量替换，保持原有字符串。 • null：将原有字符串替换为空字符串。 • error：输出报错信息，保持原有字符串。
VAR_MAX_LENGTH	\set VAR_MAX_LENGTH *variable*	用于控制变量值的长度，默认为 4096。如果变量值的长度超过该值，变量值会被截断，并输出告警信息。
ERROR_LEVEL	\set ERROR_LEVEL transaction | statement	表示 ERROR 标识成功或者失败类型，取值为 transaction 或 statement，默认为 transaction。 • statement：ERROR 记录上一条 SQL 语句是否执行成功。 • transaction：ERROR 记录上一条 SQL 语句是否执行成功，或上一个事务内部执行是否出错。
ERROR	\set ERROR true | false	表示上一条 SQL 语句执行成功或失败，或上一个事务内部执行是否出错，成功取值 false，失败取值 true，默认为 false。执行 SQL 语句更新，不建议手动设置。
LAST_ERROR_SQLSTATE	\set LAST_ERROR_SQLSTATE state	表示上一条执行失败的 SQL 语句的错误状态码，默认为‘00000’。执行 SQL 语句更新，不建议手动设置。
LAST_ERROR_MESSAGE	\set LAST_ERROR_MESSAGE message	表示上一条执行失败的 SQL 语句的错误信息，默认为空字符串。执行 SQL 语句更新，不建议手动设置。
ROW_COUNT	\set ROW_COUNT count	• ERROR_LEVEL 为 statement 时，表示上一条 SQL 语句执行返回的行数或受影响的行数。 • ERROR_LEVEL 为 transaction 时，如果事务结束时内部有错，表示事务内最后一个 SQL

变量	设置方法	变量说明
		语句执行返回的行数或受影响的行数，否则表示上一条 SQL 语句执行返回的行数或受影响的行数。 如果 SQL 语句执行失败设置为 0，默认为 0。执行 SQL 语句更新，不建议手动设置。
SQLSTATE	\set SQLSTATE state	• ERROR_LEVEL 为 statement 时，表示上一条 SQL 语句执行的状态码。 • ERROR_LEVEL 为 transaction 时，如果事务结束时内部有错，表示事务内最后一个 SQL 语句执行的状态码，否则表示上一条 SQL 语句执行的状态码。 默认为‘00000’。执行 SQL 语句更新，不建议手动设置。
LAST_SYS_CODE	\set LAST_SYS_CODE code	表示上一条系统命令执行的返回值，默认为 0。使用元命令\!调用系统命令更新，不建议手动设置。
DYNAMIC_PARAM_ENABLE	\set DYNAMIC_PARAM_ENABLE true | false	用于控制执行 SQL 语句生成变量和${}变量引用方式，默认为 fasle。 • true：执行 SQL 语句生成变量，支持${}变量引用方式。 • false：执行 SQL 语句不生成变量，不支持${}变量引用方式。
CONVERT_QUOTE_IN_DYNAMIC_PARAM	\set CONVERT_QUOTE_IN_DYNAMIC_PARAM true | false	用于控制动态变量解析是否需要对单引号、双引号、反斜线进行转义，默认为 true。 • true：动态变量解析需要对单引号、双引号、反斜线进行转义，SQL 代换时会自动转义变量中的引号和反斜线。 • false：动态变量解析不需要对单引号、双引号、反斜线进行转义，SQL 代换时不对变量中的字符串做处理，需要用户根据不同的情况进行手动转义。 使用示例详见 CONVERT_QUOTE_IN_DYNAMIC...。
RESULT_DELIMITER	\set RESULT_DELIMITER delimiter	执行 SQL 语句生成变量时，多条记录之间的拼接使用该参数控制，默认为“,”。
COMPARE_STRATEGY	\set COMPARE_STRATEGY default | natural | equal	用于控制\if 表达式中大小比较的策略，默认为 default。 • default：默认的比较策略，只支持字符串或数字比较，不支持混合比较。单引号内的按照字符串处理，单引号外的按照数字处理。

变量	设置方法	变量说明
		• natural：在 default 的基础上，包含动态变量的按照字符串处理。当比较操作符有一侧是数字，尝试将另一侧转换为数字，然后比较。如果转换失败，报错且比较结果为假。 • equal：只支持等值比较，所有情况按照字符串比较。 详细说明和使用示例见\if 条件块比较规则说明与示例。
COMMAND_ERROR_STOP	\set COMMAND_ERROR_STOP on | off	用于控制元命令执行错误时是否报错退出，默认不退出。 使用示例详见 COMMAND_ERROR_STOP 使用示例。

- 特殊变量 ERROR_LEVEL 和 ERROR 使用示例：

当 ERROR_LEVEL 为 statement 时，ERROR 只记录上一条 SQL 语句是否执行成功。示例如下，当事务中出现 SQL 执行报错，事务结束时，ERROR 值为 false。此时的 ERROR 只记录上一个 SQL 语句 end 是否执行成功。

```
\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
```

当 ERROR_LEVEL 为 transaction 时，ERROR 可以捕获事务内的 SQL 执行错误。示例如下，事务中出现 SQL 执行报错，事务结束时，ERROR 值为 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
```

- 特殊变量 COMMAND_ERROR_STOP 使用示例：

当 COMMAND_ERROR_STOP 为 on 时，元命令执行错误时，报错退出。开启时能有效的识别到元命令的执行错误。

当 COMMAND_ERROR_STOP 为 off 时，元命令执行错误时，打印相关信息不退出，脚本继续执行。

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

select id, name from student;
```

如上脚本中 COMMAND_ERROR_STOP 设置为 on，元命令报错之后输出错误信息，脚本不再执行。

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

如果 COMMAND_ERROR_STOP 设置为 off，元命令报错之后输出错误信息，继续执行 SELECT 语句。

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

- SQL 代换

  像元命令的参数一样，gsql 变量的一个关键特性是可以把 gsql 变量替换成正规的 SQL 语句。此外，gsql 还提供为变量更换新的别名或其他标识符等功能。使用 SQL 代换方式替换一个变量的值可在变量前加冒号。例如：

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

执行以上命令，将会查询 HR.areaS 表。

须知

变量的值是逐字复制的，甚至可以包含不对称的引号或反斜杠命令。所以必须保证输入的内容有意义。

- 特殊变量 CONVERT_QUOTE_IN_DYNAMIC_PARAM 使用示例：

  当 CONVERT_QUOTE_IN_DYNAMIC_PARAM 为 true 时，SQL 代换时会自动转义变量中的引号和反斜线。

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

```
-- 单引号转义，结果中还是两个单引号
 select '${SpecialCharacters}' as "test";
   test
-----------
 ""abc''\\
(1 row)

-- 单引号、反斜线转义，结果中还是两个单引号、两个反斜线
 select E'${SpecialCharacters}' as "test";
   test
-----------
 ""abc''\\
(1 row)

-- 双引号转义，结果中还是两个单引号
-- 因为列名中有字母、数字、下划线之外的其他字符，所以有错误信息
 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)
```

当 CONVERT_QUOTE_IN_DYNAMIC_PARAM 为 false 时，SQL 代换时不对变量中的字符串做处理，需要用户根据不同的情况进行手动转义。

说明

不建议用户设置 CONVERT_QUOTE_IN_DYNAMIC_PARAM 为 false，建议使用默认的 true。

因为 SQL 代换时，"内需要对单引号转义，E"内需要对单引号、反斜线转义，""内需要对双引号转义。用户需要根据变量所在的位置不同，对引号和反斜线进行不同的处理。这使得 SQL 代换中变量使用逻辑复杂且易出错。

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

-- 单引号未转义，结果中只有一个单引号
 select '${SpecialCharacters}' as "test";
   test
----------
 ""abc'\\
(1 row)

-- 单引号、反斜线未转义，结果中只有一个单引号、一个反斜线
 select E'${SpecialCharacters}' as "test";
  test
---------
 ""abc'\
(1 row)
```

```
-- 双引号未转义，结果中只有一个双引号
-- 因为列名中有字母、数字、下划线之外的其他字符，所以有错误信息
 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)
```

- 提示符

  通过表 3-3 的三个变量可以设置 gsql 的提示符，这些变量是由字符和特殊的转义字符所组成。

表3-3 提示符变量

变量	描述	示例
PROMPT1	gsql 请求一个新命令时使用的正常提示符。 PROMPT1 的默认值为： `%/%R%#`	使用变量 PROMPT1 切换提示符： • 提示符变为[local]： `\set PROMPT1 %M` `[local:/tmp/gaussdba_mppdb]` • 提示符变为 name： `\set PROMPT1 name` `name` • 提示符变为=： `\set PROMPT1 %R` `=`
PROMPT2	在一个命令输入期待更多输入时（例如，查询没有用一个分号结束或者引号不完整）显示的提示符。	使用变量 PROMPT2 显示提示符： `\set PROMPT2 TEST` `select * from HR.areaS TEST;` ` area_id |  area_name` `---------+--------------------` `  1 | Wood` `  2 | Lake` `  4 | Iron` `  3 | Desert` `(4 rows))`
PROMPT3	当执行 COPY 命令，并期望在终端输入数据时（例如，COPY FROM STDIN），显示提示符。	使用变量 PROMPT3 显示 COPY 提示符： `\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` `>>>>\.`

提示符变量的值是按实际字符显示的，但是，当设置提示符的命令中出现“%”时，变量的值根据“%”后的字符，替换为已定义的内容，已定义的提示符请参见表 3-4。

表3-4 已定义的替换

符号	符号说明
%M	主机的全名（包含域名），若连接是通过 Unix 域套接字进行的，则全名为[local]，若 Unix 域套接字不是编译的缺省位置，就是[local:/dir/name]。
%m	主机名删去第一个点后面的部分。若通过 Unix 域套接字连接，则为[local]。
%>	主机正在侦听的端口号。
%n	数据库会话的用户名。
%/	当前数据库名称。
%~	类似 %/，如果数据库是缺省数据库时输出的是波浪线~。
%#	如果会话用户是数据库系统管理员，使用#，否则用>。
%R	• 对于 PROMPT1 通常是“=”，如果是单行模式则是“^”，如果会话与数据库断开（如果\connect 失败可能发生）则是“!”。 • 对于 PROMPT2 该序列被“ -”、“ *”、单引号、双引号或“$”(取决于 gsql 是否等待更多的输入：查询没有终止、正在一个 /* ... */ 注释里、正在引号或者美元符扩展里)代替。
%x	事务状态： • 如果不在事务块里，则是一个空字符串。 • 如果在事务块里，则是“*”。 • 如果在一个失败的事务块里则是“!”。 • 如果无法判断事务状态时为“?”（比如没有连接）。
%digits	指定字节值的字符将被替换到该位置。
%:name	gsql 变量“name”的值。
%command	command 的输出，类似于使用“^”替换。
%[. . . %]	提示可以包含终端控制字符，这些字符可以改变颜色、背景、提示文本的风格、终端窗口的标题。例如， potgres=> \set PROMPT1 '%[%033[1;33;40m%]%n@%/%R%[%033[0m%]%#' 这个句式的结果是在 VT100 兼容的可显示彩色的终端上的一个宽体（1;）黑底黄字（33;40）。

环境变量

表3-5 与 gsql 相关的环境变量

名称	描述
COLUMNS	如果\set columns 为 0，则由此参数控制 wrapped 格式的宽度。这个宽度用于决定在自动扩展的模式下，是否要把宽输出模式变成竖线的格式。
PAGER	如果查询结果无法在一页显示，它们就会被重定向到这个命令。可以用\pset 命令关闭分页器。典型的是用命令 more 或 less 来实现逐页查看。缺省值是平台相关的。 说明 less 的文本显示，受系统环境变量 LC_CTYPE 影响。
PSQL_EDITOR	\e 和\ef 命令使用环境变量指定的编辑器。变量是按照列出的先后顺序检查的。在 Unix 系统上默认的编辑工具是 vi。
EDITOR	
VISUAL	
PSQL_EDITOR_LINENUMBER_ARG	当\e 和\ef 带上一行数字参数使用时，这个变量指定的命令行参数用于向编辑器传递起始行数。像 Emacs 或 vi 这样的编辑器，这只是个加号。如果选项和行号之间需要空白，在变量的值后加一个空格。例如： `PSQL_EDITOR_LINENUMBER_ARG = '+'` `PSQL_EDITOR_LINENUMBER_ARG='--line '` Unix 系统默认的是+。
PSQLRC	用户的.gsqlrc 文件的交互位置。
SHELL	使用\!命令跟 shell 执行的命令是一样的效果。
TMPDIR	存储临时文件的目录。缺省是/tmp。

3.2 使用指导

下载安装 gsql 并使用 gsql 连接集群数据库

关于 gsql 的下载、安装以及连接集群数据库的具体操作，请参见《数据仓库服务用户指南》中的使用 gsql 命令行客户端连接集群。

示例

以把一个查询分成多行输入为例。注意提示符的变化：

```
postgres=# CREATE TABLE HR.areaS(
postgres(# area_ID   NUMBER,
```

```
postgres(# area_NAME VARCHAR2(25)
postgres-# )tablespace EXAMPLE;
CREATE TABLE
```

查看表的定义：

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

向 HR.areaS 表插入四行数据：

```
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
```

切换提示符：

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

查看表：

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

可以用\pset 命令以不同的方法显示表：

```
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)
```

使用元命令：

```
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 获取帮助

操作步骤

- 连接数据库时，可以使用如下命令获取帮助信息。

  ```
  gsql --help
  ```

  显示如下帮助信息：

  ```
  ......
  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
  ......
  ```

- 连接到数据库后，可以使用如下命令获取帮助信息。

  ```
  help
  ```

  显示如下帮助信息：

  ```
  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
```

任务示例

步骤 1 查看 gsql 的帮助信息。具体执行命令请参见表 3-6。

表3-6 使用 gsql 联机帮助

描述	示例
查看版权信息	\copyright
查看 GaussDB(DWS)支持的 SQL 语句的帮助	查看 GaussDB(DWS)支持的 SQL 语句的帮助 例如，查看 GaussDB(DWS)支持的所有 SQL 语句： `\h` `Available help:` `  ABORT` `  ALTER DATABAE` `  ALTER DATA SOURCE` `... ...` 例如，查看 CREATE DATABASE 命令的参数可使用下面的命令： `\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 ]}[...] ];`
查看 gsql 命令的帮助	例如，查看 gsql 支持的命令： `\?` `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` `... ...`

----结束

3.4 命令参考

详细的 gsql 参数请参见表 3-7、表 3-8、表 3-9 和表 3-10。

表3-7 常用参数

参数	参数说明	取值范围
-c, --command=COMMAND	声明 gsql 要执行一条字符串命令然后退出。	-
-C, --set-file=FILENAME	使用文件作为命令源而不是交互式输入，gsql 处理完文件后不退出，继续处理其他内容。	绝对路径或相对路径，且满足操作系统路径命名规则。
-d, --dbname=DBNAME	指定想要连接的数据库名称。	字符串。
-D, --dynamic-param	用于控制执行 SQL 语句设置变量和${}变量引用方式，具体示例参见变量。	-
-f, --file=FILENAME	使用文件作为命令源而不是交互式输入。gsql 将在处理完文件后结束。如果 FILENAME 是-（连字符），则从标准输入读取。	绝对路径或相对路径，且满足操作系统路径命名规则。
-l, --list	列出所有可用的数据库，然后退出。	-
-v, --set, --variable=NAME=VALUE	设置 gsql 变量 NAME 为 VALUE。 变量的示例和详细说明请参见变量。	-
-X, --no-gsqlrc	不读取启动文件（系统范围的 gsqlrc 或者用户的~/.gsqlrc 都不读取）。 说明 启动文件默认为~/.gsqlrc，或通过 PSQLRC 环境变量指定。	-
-1 ("one"), --single-transaction	当 gsql 使用-f 选项执行脚本时，会在脚本的开头和结尾分别加上 START TRANSACTION/COMMIT 用以把整个脚本当作一个事务执行。这将保证该脚本完全执行成功，或者脚本无效。 说明 如果脚本中已经使用了 START TRANSACTION，COMMIT，ROLLBACK，则该选项无效。	-
-?, --help	显示关于 gsql 命令行参数的帮助信息然后退出。	-

参数	参数说明	取值范围
-V, --version	打印 gsql 版本信息然后退出。	-

表3-8 输入和输出参数

参数	参数说明	取值范围
-a, --echo-all	在读取行时向标准输出打印所有内容。 注意 使用此参数可能会暴露部分 SQL 语句中的敏感信息，如创建用户语句中的 password 信息等，请谨慎使用。	-
-e, --echo-queries	把所有发送给服务器的查询同时回显到标准输出。 注意 使用此参数可能会暴露部分 SQL 语句中的敏感信息，如创建用户语句中的 password 信息等，请谨慎使用。	-
-E, --echo-hidden	回显由\d 和其他反斜杠命令生成的实际查询。	-
-k, --with-key=KEY	使用 gsql 对导入的加密文件进行解密。 须知 对于本身就是 shell 命令中的关键字符如单引号（'）或双引号（"），Linux shell 会检测输入的单引号（'）或双引号（"）是否匹配。如果不匹配，shell 认为用户没有输入完毕，会一直等待用户输入，从而不会进入到 gsql 程序。	-
-L, --log-file=FILENAME	除了正常的输出源之外，把所有查询输出记录到文件 FILENAME 中。 注意 • 使用此参数可能会暴露部分 SQL 语句中的敏感信息，如创建用户语句中的 password 信息等，请谨慎使用。 • 此参数只保留查询结果到相应文件中，主要目标是为了查询结果能够更好更准确地被其他调用者（例如自动化运维脚本）解析；而不是保留 gsql 运行过程中的相关日志信息。	绝对路径或相对路径，且满足操作系统路径命名规则。
-m, --maintenance	允许在两阶段事务恢复期间连接集群。 说明 该选项是一个开发选项，禁止用户使用，只限专业技术人员使用，功能是：使用该选项时，gsql 可以连接到备机，用于校验主备机数据的一致性。	-
-n, --no-libedit	关闭命令行编辑。	-

参数	参数说明	取值范围
-o, --output=FILENAME	将所有查询输出重定向到文件 FILENAME。	绝对路径或相对路径，且满足操作系统路径命名规则。
-q, --quiet	安静模式，执行时不会打印出额外信息。	缺省时 gsql 将打印许多其他输出信息。
-s, --single-step	单步模式运行。意味着每个查询在发往服务器之前都要提示用户，用这个选项也可以取消执行。此选项主要用于调试脚本。 注意 使用此参数可能会暴露部分 SQL 语句中的敏感信息，如创建用户语句中的 password 信息等，请谨慎使用。	-
-S, --single-line	单行运行模式，这时每个命令都将由换行符结束，像分号那样。	-

表3-9 输出格式参数

参数	参数说明	取值范围
-A, --no-align	切换为非对齐输出模式。	缺省为对齐输出模式。
-F, --field-separator=STRING	设置域分隔符（默认为“|”）。	-
-H, --html	打开 HTML 格式输出。	-
-P, --pset=VAR[=ARG]	在命令行上以\pset 的风格设置打印选项。 说明 这里必须用等号而不是空格分隔名称和值。例如，把输出格式设置为 LaTeX，可以键入-P format=latex	-
-R, --record-separator=STRING	设置记录分隔符。	-
-r	开启客户端操作历史记录功能。	缺省为关闭。
-t, --tuples-only	只打印行。	-
-T, --table-attr=TEXT	允许声明放在 HTML table 标签里的选项。 使用时请搭配参数“-H,--html”，指定为 HTML 格式	-

参数	参数说明	取值范围
	输出。	
-x, --expanded	打开扩展表格式模式。	-
-z, --field-separator-zero	设置非对齐输出模式的域分隔符为空。 使用时请搭配参数“-A, --no-align”，指定为非对齐输出模式。	-
-0, --record-separator-zero	设置非对齐输出模式的记录分隔符为空。 使用时请搭配参数“-A, --no-align”，指定为非对齐输出模式。	-
-g	显示所有 SQL 语句和指定文件的分隔符。 说明 -g 参数必须和-f 参数一起设置。	-

表3-10 连接参数

参数	参数说明	取值范围
-h, --host=HOSTNAME	指定正在运行服务器的主机名或者 Unix 域套接字的路径。	如果省略主机名，gsql 将通过 Unix 域套接字与本地主机的服务器相联，或者在没有 Unix 域套接字的机器上，通过 TCP/IP 与 localhost 连接。
-p, --port=PORT	指定数据库服务器的端口号。 可以通过 port 参数修改默认端口号。	默认为 8000。
-U, --username=USERNAME	指定连接数据库的用户。 说明 • 通过该参数指定用户连接数据库时，需要同时提供用户密码用以身份验证。您可以通过交换方式输入密码，或者通过-W 参数指定密码。 • 用户名中包含有字符$，需要在字符$前增加转义字符才可成功连接数据库。	字符串。默认使用与当前操作系统用户同名的用户。
-W, --password=PASSWORD	当使用-U 参数连接远端数据库时，可通过该选项指定密码。 说明	符合密码复杂度要求。

参数	参数说明	取值范围
	用户密码中包含特殊字符“\”和"`"时，需要增加转义字符才可成功连接数据库。 如果用户未输入该参数，但是数据库连接需要用户密码，这时将出现交互式输入，请用户输入当前连接的密码。该密码最长长度为 999 字节，受限于 GUC 参数 password max length 的最大值。	

3.5 元命令参考

介绍使用 GaussDB(DWS)数据库命令行交互工具登录数据库后，gsql 所提供的元命令。所谓元命令就是在 gsql 里输入的任何以不带引号的反斜杠开头的命令。

注意事项

- 一个 gsql 元命令的格式是反斜杠后面紧跟一个动词，然后是任意参数。参数命令动词和其他参数以任意个空白字符间隔。
- 要在参数里面包含空白，必须用单引号把它引起来。要在这样的参数里包含单引号，可以在前面加一个反斜杠。任何包含在单引号里的内容都会被进一步进行类似 C 语言的替换：\n（新行）、\t（制表符）、\b（退格）、\r（回车）、\f（换页）、\digits（八进制表示的字符）、\xdigits（十六进制表示的字符）。
- 用""包围的内容被当做一个命令行传入 shell。该命令的输出（删除了结尾的新行）被当做参数值。
- 如果不带引号的参数以冒号（:）开头，它会被当做一个 gsql 变量，并且该变量的值最终会成为真正的参数值。
- 有些命令以一个 SQL 标识的名称（比如一个表）为参数。这些参数遵循 SQL 语法关于双引号的规则：不带双引号的标识强制转换成小写，而双引号保护字母不进行大小写转换，并且允许在标识符中使用空白。在双引号中，成对的双引号在结果名字中分析成一个双引号。比如，FOO"BAR"BAZ 解析成 fooBARbaz；而"Aweird""name"解析成 A weird"name。
- 对参数的分析在遇到另一个不带引号的反斜杠时停止。这里会认为是一个新的元命令的开始。特殊的双反斜杠序列（\\）标识参数的结尾并将继续分析后面的 SQL 语句（如果存在）。这样 SQL 和 gsql 命令可以自由的在一行里面混合。但是在任何情况下，一条元命令的参数不能延续超过行尾。

元命令

元命令的详细说明请参见表 3-11、表 3-12、表 3-13、表 3-14、表 3-16、表 3-18、表 3-19、表 3-20 和表 3-22。

须知

以下命令中所提到的 FILE 代表文件路径。此路径可以是绝对路径（如 /home/gauss/file.txt），也可以是相对路径（file.txt，file.txt 会默认在用户执行 gsql 命令所在的路径下创建）。

表3-11 一般的元命令

参数	参数说明	取值范围
\copyright	显示 GaussDB(DWS)的版本和版权信息。	-
\g [FILE] or ;	执行查询（并将结果发送到文件或管道）。	-
\h(\help) [NAME]	给出指定 SQL 语句的语法帮助。	如果没有给出 NAME，gsql 将列出可获得帮助的所有命令。如果 NAME 是一个星号（*），则显示所有 SQL 语句的语法帮助。
\parallel [on [num]|off]	控制并发执行开关。 • on：打开控制并发执行开关，且最大并发数为 num。 • off：关闭控制并发执行开关。 说明 • 不支持事务中开启并发执行以及并发中开启事务。 • 不支持\d 这类元命令的并发。 • 并发 select 返回结果混乱问题，此为客户可接受，core、进程停止响应不可接受。 • 不推荐在并发中使用 set 语句，否则导致结果与预期不一致。 • 不支持创建临时表！如需使用临时表，需要在开启 parallel 之前创建好，并在 parallel 内部使用。parallel 内部不允许创建临时表。 • \parallel 执行时最多会启动 num 个独立的 gsql 进程连接服务器。 • \parallel 中所有作业的持续时间不能超过 session_timeout，否则可能会导致并发执行过程中断连。	num 的默认值：1024。 须知 • 服务器能接受的最大连接数受 max_connection 及当前已有连接数限制。 • 设置 num 时请考虑服务器当前可接受的实际连接数合理指定。

参数	参数说明	取值范围
\q [value]	退出 gsql 程序。在一个脚本文件里，只在脚本终止的时候执行。退出码可由 value 值决定。	-

表3-12 查询缓存区元命令

参数	参数说明
\e [FILE] [LINE]	使用外部编辑器编辑查询缓冲区（或者文件）。
\ef [FUNCNAME [LINE]]	使用外部编辑器编辑函数定义。如果指定了 LINE（即行号），则光标会指到函数体的指定行。
\p	打印当前查询缓冲区到标准输出。
\r	重置（或清空）查询缓冲区。
\w FILE	将当前查询缓冲区输出到文件。

表3-13 输入/输出元命令

参数	参数说明
\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]]	在任何 psql 客户端登录数据库成功后可以执行导入导出数据， 这是一个运行 SQL COPY 命令的操作，但不是读取或写入指定文件的服务器，而是读取或写入文件，并在服务器和本地文件系统之间路由数据。 这意味着文件的可访问性和权限是本地用户的权限，而不是服务器的权限，并且不需要数据库初始化用户权限。 说明 \COPY 只适合小批量，格式良好的数据导入，容错能力较差。导入数据应优先选择 GDS 或 COPY。
\echo [STRING]	把字符串写到标准输出。
\i FILE	从文件 FILE 中读取内容，并将其当作输入，执行查询。
\i+ FILE KEY	执行加密文件中的命令。
\ir FILE	和\i 类似，只是相对于存放当前脚本的路径。
\ir+ FILE KEY	和\i+类似，只是相对于存放当前脚本的路径。
\o [FILE]	把所有的查询结果发送到文件里。

参数	参数说明
\qecho [STRING]	把字符串写到查询结果输出流里。

说明

表 3-14 中的选项 S 表示显示系统对象，+表示显示对象附加的描述信息。PATTERN 用来指定要被显示的对象名称。

表3-14 显示信息元命令

参数	参数说明	取值范围	示例
\d[S+]	列出当前 search_path 中模式下所有的表、视图和序列。当 search_path 中不同模式存在同名对象时，只显示 search_path 中位置靠前模式下的同名对象。	-	列出当前 search_path 中模式下所有的表、视图和序列。 `\d`
\d[S+] NAME	列出指定表结构、视图结构和索引的结构。	-	假设存在表 a，列出指定表 a 的表结构。 `\dtable+ a`
\d+ [PATTERN]	列出所有表、视图和索引。	如果声明了 PATTERN，只显示名字匹配 PATTERN 的表、视图和索引。	列出所有名称以 f 开头的表、视图和索引。 `\d+ f*`
\da[S] [PATTERN]	列出所有可用的聚集函数，以及它们操作的数据类型和返回值类型。	如果声明了 PATTERN，只显示名字匹配 PATTERN 的聚集函数。	列出所有名称以 f 开头可用的聚集函数，以及它们操作的数据类型和返回值类型。 `\da f*`
\db[+] [PATTERN]	列出所有可用的表空间。	如果声明了 PATTERN，只显示名字匹配 PATTERN 的表空间。	列出所有名称以 p 开头的可用表空间。 `\db p*`
\dc[S+] [PATTERN]	列出所有字符集之间的可用转换。	如果声明了 PATTERN，只显示名字匹配 PATTERN 的转换。	列出所有字符集之间的可用转换。 `\dc *`

参数	参数说明	取值范围	示例
\dC[+] [PATTERN]	列出所有类型转换。	如果声明了PATTERN，只显示名字匹配PATTERN 的转换。	列出所有名称以 c 开头的类型转换。 `\dC c*`
\dd[S] [PATTERN]	显示所有匹配 PATTERN 的描述。	如果没有给出参数，则显示所有可视对象。“对象”包括：聚集、函数、操作符、类型、关系(表、视图、索引、序列、大对象)、规则。	列出所有可视对象。 `\dd`
\ddp [PATTERN]	显示所有默认的使用权限。	如果指定了PATTERN，只显示名字匹配PATTERN 的使用权限。	列出所有默认的使用权限。 `\ddp`
\dD[S+] [PATTERN]	列出所有可用域。	如果声明了PATTERN，只显示名字匹配PATTERN 的域。	列出所有可用域。 `\dD`
\ded[+] [PATTERN]	列出所有的 Data Source 对象。	如果声明了PATTERN，只显示名字匹配PATTERN 的对象。	列出所有的 Data Source 对象。 `\ded`
\det[+] [PATTERN]	列出所有的外部表。	如果声明了PATTERN，只显示名字匹配PATTERN 的表。	列出所有的外部表。 `\det`
\des[+] [PATTERN]	列出所有的外部服务器。	如果声明了PATTERN，只显示名字匹配PATTERN 的服务器。	列出所有的外部服务器。 `\des`
\deu[+] [PATTERN]	列出用户映射信息。	如果声明了PATTERN，只显示名字匹配PATTERN 的信息。	列出用户映射信息。 `\deu`

参数	参数说明	取值范围	示例
\dew[+] [PATTERN]	列出封装的外部数据。	如果声明了PATTERN，只显示名字匹配PATTERN 的数据。	列出封装的外部数据。 `\dew`
\df[antw] [S+] [PATTERN]	列出所有可用函数，以及它们的参数和返回的数据类型。a 代表聚集函数，n 代表普通函数，t 代表触发器，w 代表窗口函数。	如果声明了PATTERN，只显示名字匹配PATTERN 的函数。	列出所有可用函数，以及它们的参数和返回的数据类型。 `\df`
\dF[+] [PATTERN]	列出所有的文本搜索配置信息。	如果声明了PATTERN，只显示名字匹配PATTERN 的配置信息。	列出所有的文本搜索配置信息。 `\dF+`
\dFd[+] [PATTERN]	列出所有的文本搜索字典。	如果声明了PATTERN，只显示名字匹配PATTERN 的字典。	列出所有的文本搜索字典。 `\dFd`
\dFp[+] [PATTERN]	列出所有的文本搜索分析器。	如果声明了PATTERN，只显示名字匹配PATTERN 的分析器。	列出所有的文本搜索分析器。 `\dFp`
\dFt[+] [PATTERN]	列出所有的文本搜索模板。	如果声明了PATTERN，只显示名字匹配PATTERN 的模板。	列出所有的文本搜索模板。 `\dFt`
\dg[+] [PATTERN]	列出所有数据库角色。 说明 因为用户和群组的概念被统一为角色，所以这个命令等价于\du。为了和以前兼容，所以保留两个命令。	如果指定了PATTERN，只显示名字匹配PATTERN 的角色。	列出名字为‘j_e’所有数据库角色。 `\dg j?e`
\dl	\lo_list 的别名，显示一个大对象的列表。	-	列出所有的大对象。 `\dl`
\dL[S+] [PATTE	列出可用的程序语言。	如果指定了	列出可用的程序语

参数	参数说明	取值范围	示例
RN]		PATTERN，只列出名字匹配 PATTERN 的语言。	言。 `\dL`
\dn[S+] [PATTERN]	列出所有的模式（名字空间）。	如果声明了 PATTERN，只列出名字匹配 PATTERN 的模式名。缺省时，只列出用户创建的模式。	列出所有名称以 d 开头的模式以及相关信息。 `\dn+ d*`
\do[S] [PATTERN]	列出所有可用的操作符，以及它们的操作数和返回的数据类型。	如果声明了 PATTERN，只列出名字匹配 PATTERN 的操作符。缺省时，只列出用户创建的操作符。	列出所有可用的操作符，以及它们的操作数和返回的数据类型。 `\do`
\dO[S+] [PATTERN]	列出排序规则。	如果声明了 PATTERN，只列出名字匹配 PATTERN 的规则。缺省时，只列出用户创建的规则。	列出排序规则。 `\dO`
\dp [PATTERN]	列出一列可用的表、视图以及相关的权限信息。 \dp 显示结果如下： `rolename=xxxx/yyyy --赋予一个角色的权限` `=xxxx/yyyy --赋予 public 的权限` xxxx 表示赋予的权限，yyyy 表示授予这个权限的角色。权限的参数说明请参见表 3-15。	如果指定了 PATTERN，只列出名字匹配 PATTERN 的表、视图。	列出一列可用的表、视图以及相关的权限信息。 `\dp`
\drds [PATTERN1 [PATTERN2]]	列出所有修改过的配置参数。这些设置可以是针对角色的、针对数据库的或者同时针对两者的。PATTERN1 和 PATTERN2 表示要列出的角色 PATTERN 和数据库 PATTERN。	如果声明了 PATTERN，只列出名字匹配 PATTERN 的规则。缺省或指定*时，则会列出所有设置。	列出数据库所有修改过的配置参数。 `\drds *`

参数	参数说明	取值范围	示例
\dT[S+] [PATTERN]	列出所有的数据类型。	如果指定了PATTERN，只列出名字匹配PATTERN 的类型。	列出所有的数据类型。 `\dT`
\du[+] [PATTERN]	列出所有数据库角色。 说明 因为用户和群组的概念被统一为角色，所以这个命令等价于\dg。为了和以前兼容，所以保留两个命令。	如果指定了PATTERN，则只列出名字匹配PATTERN 的角色。	列出所有数据库角色。 `\du`
\dE[S+] [PATTERN] \di[S+] [PATTERN] \ds[S+] [PATTERN] \dt[S+] [PATTERN] \dv[S+] [PATTERN]	这一组命令，字母 E，i，s，t 和 v 分别代表着外部表，索引，序列，表和视图。可以以任意顺序指定其中一个或者它们的组合来列出这些对象。例如：\dit 列出所有的索引和表。在命令名称后面追加+，则每一个对象的物理尺寸以及相关的描述也会被列出。 说明 本版本暂时不支持序列。	如果指定了PATTERN，只列出名称匹配该PATTERN 的对象。默认情况下只会显示用户创建的对象。通过PATTERN 或者 S 修饰符可以把系统对象包括在内。	列出所有的索引和视图。 `\div`
\dx[+] [PATTERN]	列出安装数据库的扩展信息。	如果指定了PATTERN，则只列出名字匹配PATTERN 的扩展信息。	列出安装数据库的扩展信息。 `\dx`
\l[+]	列出服务器上所有数据库的名字、所有者、字符集编码以及使用权限。	-	列出服务器上所有数据库的名字、所有者、字符集编码以及使用权限。 `\l`
\sf[+] FUNCNAME	显示函数的定义。 说明 对于带圆括号的函数名，需要在函数名两端添加双引号，并且在双引号后面加上参数类型列表。参数类型列表两端添加圆括号。	-	假设存在函数function_a 和函数名带圆括号的函数func()name，列出函数的定义。 `\sf function_a` `\sf`

参数	参数说明	取值范围	示例
			`"func()name"(argtype1, argtype2)`
\z [PATTERN]	列出数据库中所有表、视图和序列，以及它们相关的访问特权。	如果给出任何 pattern ，则被当成一个正则表达式，只显示匹配的表、视图、序列。	列出数据库中所有表、视图和序列，以及它们相关的访问特权。 `\z`

表3-15 权限的参数说明

参数	参数说明
r	SELECT：允许对指定的表、视图读取数据。
w	UPDATE：允许对指定表更新字段。
a	INSERT：允许对指定表插入数据。
d	DELETE：允许删除指定表中的数据。
D	TRUNCATE：允许清理指定表中的数据。
x	REFERENCES：允许创建外键约束。
t	TRIGGER：允许在指定表上创建触发器。
X	EXECUTE：允许使用指定的函数，以及利用这些函数实现的操作符。
U	USAGE： • 对于过程语言，允许用户在创建函数时，指定过程语言。 • 对于模式，允许访问包含在指定模式中的对象。 • 对于序列，允许使用 nextval 函数。
C	CREATE： • 对于数据库，允许在该数据库里创建新的模式。 • 对于模式，允许在该模式中创建新的对象。 • 对于表空间，允许在其中创建表，以及允许创建数据库和模式的时候把该表空间指定为其缺省表空间。
c	CONNECT：允许用户连接到指定的数据库。
T	TEMPORARY：允许创建临时表。
A	ANALYZE|ANALYSE：允许分析表。
arwdDxtA	ALL PRIVILEGES：一次性给指定用户/角色赋予所有可赋

参数	参数说明
	予的权限。
*	给前面权限的授权选项。

表3-16 格式化元命令

参数	参数说明
\a	对齐模式和非对齐模式之间的切换。
\C [STRING]	把正在打印的表的标题设置为一个查询的结果或者取消这样的设置。
\f [STRING]	对于不对齐的查询输出，显示或者设置域分隔符。
\H	• 若当前模式为文本格式，则切换为 HTML 输出格式。 • 若当前模式为 HTML 格式，则切换回文本格式。
\pset NAME [VALUE]	设置影响查询结果表输出的选项。NAME 的取值见表 3-17。
\t [on|off]	切换输出的字段名的信息和行计数脚注。
\T [STRING]	指定在使用 HTML 输出格式时放在 table 标签里的属性。如果参数为空，不设置。
\x [on|off|auto]	切换扩展行格式。

表3-17 可调节的打印选项

选项	选项说明	取值范围
border	value 必须是一个数字。通常这个数字越大，表的边界就越宽线就越多，但是这个取决于特定的格式。	• 在 HTML 格式下，取值范围为大于 0 的整数。 • 在其他格式下，取值范围： – 0：无边框 – 1：内部分隔线 – 2：台架
expanded (或 x)	在正常和扩展格式之间切换。	• 当打开扩展格式时，查询结果用两列显示，字段名称在左、数据在右。这个模式在数据无法放进通常的"水平"模式的屏幕时很有用。 • 在正常格式下，当查询输出

选项	选项说明	取值范围
		的格式比屏幕宽时，用扩展格式。正常格式只对 aligned 和 wrapped 格式有用。
fieldsep	声明域分隔符来实现非对齐输出。这样就可以创建其他程序希望的制表符或逗号分隔的输出。要设置制表符域分隔符，键入\pset fieldsep '\t'。缺省域分隔符是 '|' (竖条符)。	-
fieldsep_zero	声明域分隔符来实现非对齐输出到零字节。	-
footer	用来切换脚注。	-
format	设置输出格式。允许使用唯一缩写（这意味着一个字母就够了）。	取值范围： • unaligned：写一行的所有列在一条直线上中，当前活动字段分隔符分隔。 • aligned：此格式是标准的，可读性最好的文本输出。 • wrapped：类似 aligned，但是包装跨行的宽数据值，使其适应目标字段的宽度输出。 • html：把表输出为可用于文档里的对应标记语言。输出不是完整的文档。 • latex：把表输出为可用于文档里的对应标记语言。输出不是完整的文档。 • troff-ms：把表输出为可用于文档里的对应标记语言。输出不是完整的文档。
null	打印一个字符串，用来代替一个 null 值。	缺省是什么都不打印，这样很容易和空字符串混淆。
numericlocale	切换分隔小数点左边的数值的区域相关的分组符号。	• on：显示指定的分隔符。 • off：不显示分隔符。 忽略此参数，显示默认的分隔符。
pager	控制查询和 gsql 帮助输出的分页器。如果设置了环境变量 PAGER，输出将被指向到指定程序，否则使用系统缺省。	• on：当输出到终端且不适合屏幕显示时，使用分页器。 • off：不使用分页器。 • always：当输出到终端无论

选项	选项说明	取值范围
		是否符合屏幕显示时，都使用分页器。
recordsep	声明在非对齐输出格式时的记录分隔符。	-
recordsep_zero	声明在非对齐输出到零字节时的记录分隔符。	-
tableattr（或 T）	声明放在 html 输出格式中 HTML table 标签的属性（例如：cellpadding 或 bgcolor）。注意：这里可能不需要声明 border，因为已经在\pset border 里用过了。如果没有给出 value，则不设置表的属性。	-
title	为随后打印的表设置标题。这个可以用于给输出一个描述性标签。如果没有给出 value，不设置标题。	-
tuples_only（或者 t）	在完全显示和只显示实际的表数据之间切换。完全显示将输出像列头、标题、各种脚注等信息。在 tuples_only 模式下，只显示实际的表数据。	-

表3-18 连接元命令

参数	参数说明	取值范围
\c[onnect] [DBNAME|- USER|- HOST|- PORT|-]	连接到一个新的数据库（当前数据库为 gaussdb）。当数据库名称长度超过 63 个字节时，默认前 63 个字节有效，连接到前 63 个字节对应的数据库，但是 gsql 的命令提示符中显示的数据库对象名仍为截断前的名称。 说明 重新建立连接时，如果切换数据库登录用户，将可能会出现交互式输入，要求输入新用户的连接密码。该密码最长长度为 999 字节，受限于 GUC 参数 password max length 的最大值。	-
\encoding [ENCODING]	设置客户端字符编码格式。	不带参数时，显示当前的编码格式。

参数	参数说明	取值范围
\conninfo	输出当前连接的数据库的信息。	-

表3-19 操作系统元命令

参数	参数说明	取值范围
\cd [DIR]	切换当前的工作目录。	绝对路径或相对路径，且满足操作系统路径命名规则。
\setenv NAME [VALUE]	设置环境变量 NAME 为 VALUE，如果没有给出 VALUE 值，则不设置环境变量。	-
\timing [on|off]	以毫秒为单位显示每条 SQL 语句的执行时间。	• on 表示打开显示。 • off 表示关闭显示。
\! [COMMAND]	返回到一个单独的 Unix shell 或者执行 Unix 命令 COMMAND。	-

表3-20 变量元命令

参数	参数说明
\prompt [TEXT] NAME	提示用户用文本格式来指定变量名字。
\set [NAME [VALUE]]	设置内部变量 NAME 为 VALUE 或者如果给出了多于一个值，设置为所有这些值的连接结果。如果没有给出第二个参数，只设变量不设值。 有一些常用变量被 gsql 特殊对待，它们是一些选项设置，通常所有特殊对待的变量都是由大写字母组成(可能还有数字和下划线)。 表 3-21 是一个所有特殊对待的变量列表。
\set-multi NAME [VALUE] \end-multi	设置内部变量 NAME 为 VALUE，VALUE 可以由多行字符串组成。\set-multi 使用时，第二个参数必须给出。可参考表下面的\set-multi 元命令使用示例。 说明 \set-multi 和\end-multi 中出现的元命令会被忽略。
\unset NAME	不设置（或删除）gsql 变量名。

\set-multi 元命令使用示例

示例文件 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 执行结果：

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

通过\set-multi \end-multi 设置变量 multi_line_var 为一个 SQL 语句，并在后面通过动态变量解析获得这个变量。

示例文件 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 执行结果：

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

 id
----
```

```
 2
(1 row)
```

通过\set-multi \end-multi 设置变量 multi_line_var 为两个 SQL 语句，并在后面通过动态变量解析获得这个变量。因为变量中的内容以“;”结尾，gsql 发送 SQL 语句并获得打印执行结果。

表3-21 \set 常用命令

名称	命令说明	取值范围
\set VERBOSITY value	这个选项可以设置为值 default， verbose，terse 之一以控制错误报告的冗余行。	value 取值范围：default，verbose，terse
\set ON_ERROR_STOP value	如果设置了这个变量，脚本处理将马上停止。如果该脚本是从另外一个脚本调用的，那个脚本也会按同样的方式停止。如果最外层的脚本不是从一次交互的 gsql 会话中调用的而是用-f 选项调用的，gsql 将返回错误代码 3，以示这个情况与致命错误条件的区别(错误代码为 1)。	value 取值范围为：on/off，true/false，yes/no，1/0
\set RETRY [retry_times]	用于控制是否开启语句出错场景下的重试功能，参数 retry_times 用来指定最大重试次数，缺省值为 5，取值范围为 5-10。当重试功能已经开启时，再次执行\set RETRY 可以关闭该功能。 使用配置文件 retry_errcodes.conf 列举需要重试的错误码列表，该文件和 gsql 可执行程序位于同一级目录下。该配置文件为系统配置，非用户定义，不允许用户直接修改。 当前支持以下 13 类出错场景的重试： • YY001：TCP 通信错误，Connection reset by peer（CN 和 DN 间通信） • YY002：TCP 通信错误，Connection reset by peer（DN 和 DN 间通信） • YY003：锁超时，Lock wait timeout.../wait transaction xxx sync time exceed xxx • YY004：TCP 通信错误，Connection timed out • YY005：SET 命令发送失败，ERROR SET query • YY006：内存申请失败，memory is temporarily unavailable • YY007：通信库错误，Memory allocate error • YY008：通信库错误，No data in buffer	retry_times 取值范围为：5-10

名称	命令说明	取值范围
	• YY009：通信库错误，Close because release memory • YY010：通信库错误，TCP disconnect • YY011：通信库错误，SCTP disconnect • YY012：通信库错误，Stream closed by remote • YY013：通信库错误，Wait poll unknown error • YY014：快照非法，Snapshot invalid • YY015：连接获取错误，Connection receive wrong • 53200：内存耗尽，Out of memory • 08006：GTM 出错，Connection failure • 08000：连接出现错误，和 DN 的通讯失败，Connection exception • 57P01：管理员关闭系统，Admin shutdown • XX003：关闭远程套接字，Stream remote close socket • XX009：重复查询，Duplicate query id • YY016：stream 查询并发更新同一行，Stream concurrent update • CG003：内存分配错误， Allocate error • CG004：致命错误，Fatal error • F0011：临时文件读取错误，File error 同时，出错时 gsql 会查询所有 CN/DN 的连接状态，当状态异常时会 sleep 1 分钟再进行重试，能够覆盖大部分主备切换场景下的出错重试。 说明 1. 不支持事务块中的语句错误重试； 2. 不支持通过 ODBC、JDBC 接口查询的出错重试； 3. 含有 unlogged 表的 sql 语句，不支持节点故障后的出错重试； 4. 当前不支持 CN 和 GTM 节点故障时，gsql 客户端的出错重试； 5. gsql 客户端本身出现的错误，不在重跑考虑范围之内；	

表3-22 大对象元命令

参数	参数说明
\lo_list	显示一个目前存储在该数据库里的所有 GaussDB(DWS)大对象和提供给他们的注释。

表3-23 流程控制元命令

参数	参数说明	取值范围
\if EXPR \elif EXPR \else \endif	这组元命令可实现可嵌套的条件块： • 条件块以\if 开始，\endif 结束。 • \if 和\endif 两者之间可以出现任意数量的\elif 子句，或单一的\else 子句。 • \if 和\elif 命令支持布尔表达式计算和字符串等值判断。 • \elif 不能出现在\else 和\endif 之间。	• 布尔表达式计算和 gsql 原有方式保持一致：true/false、yes/no、on/off、1/0，其他被认为是 true。 • 操作符和字符串之间必须使用空格。 • 支持数字比较和字符串比较，比较规则由固有变量 COMPARE_STRATEGY 控制（详见表 3-2）。默认比较规则中，使用单引号界定字符串和数字。不同规则使用示例详见\if 条件块比较规则说明与示例。 • 支持大小比较和等值判断，支持的操作符有：<，<=，>，>=，==，!=和<>。
\goto LABEL \label LABEL	这组元命令可实现无条件跳转： • \label 元命令用于创建标签。 • \goto 元命令用于跳转，可实现向上跳转和向下跳转。 说明 • 交互模式不支持使用。 • \label 元命令不支持在\if 语句块中使用。 • \goto \label 不建议在事务、PL/SQL 等语句块中使用，避免出现不可预期结果。	• LABEL 大小写敏感。 • LABEL 只能包含大小写字母、数字和下划线。 • LABEL 最大长度限制为 32（含'\0'），如果超过长度限制，会截断使用，并输出告警信息。 • \label 后面的标签名如果在同一个 session 中重复出现，会报错。
\for	这组元命令可实现循环：	-

参数	参数说明	取值范围
\loop \exit-for \end-for	• 循环块以\for 开始，以\end-for 结束。 • \for 和\loop 之间为循环条件，只支持 SQL 语句，不支持变量迭代，例如\for (i=0; i<100; ++i)。 • 循环条件中出现多条 SQL 语句时，以最后一条 SQL 语句的执行结果为循环条件。作为循环条件的 SQL 语句不能以分号结尾。 • \loop 和\end-for 之间为循环体，可以使用\exit-for 退出循环。 • \for 循环块支持多层嵌套。 说明 • 交互模式不支持使用。 • 不支持在\parallel 中使用\for 循环块。 • \for 循环块不建议在事务、PL/SQL 等语句块中使用，避免出现不可预期结果。 • \label 元命令不支持在\for 循环块中使用。 • \for \loop 之间不支持使用匿名块。	

流程控制元命令使用示例：

- \if 条件块使用示例

  示例文件 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 执行结果：

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

I am Jack
```

上面的执行结果表示，第一个 SQL 语句执行成功，并设置 Name 变量，所以进入 \elif 分支，输出“I am Jack”。特殊变量 ERROR 和 LAST_ERROR_MESSAGE 的使用参见表 3-2。

- \if 条件块比较规则说明与示例
  - default：默认的比较策略，只支持字符串或数字比较，不支持混合比较。单引号内的按照字符串处理，单引号外的按照数字处理。

示例文件 test.sql：

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

-- 以单引号界定，在单引号内的使用字符串比较
\if '${Name}' != 'Jack'
   \echo 'I am not Jack'
-- 没有单引号，使用数字比较
\elif ${ID} > 1000
   \echo 'Jack\'id is bigger than 1000'
\else
   \echo 'error'
\endif
```

gsql -d -p 25308 --dynamic-param -f test.sql 执行结果：

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

如果使用操作符两侧，一侧使用了单引号，一侧未使用，认定为字符串和数字比较。不支持，则报错。

```
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：在 default 的基础上，包含动态变量的也按照字符串处理。当比较操作符有一侧是数字比较，尝试将另一侧转换为数字，然后比较。如果转换失败，报错且比较结果为假。
    - 识别为字符串的条件有两个，满足任何一个即可。条件一，使用单引号，如'Jack'；条件二，字符串中包含动态变量（${VAR}和:VAR 两种），不论变量是否存在，如${Name}_data。条件一和条件二同时满足，如'${Name}_data'。
    - 无法识别为字符串的，尝试数字识别。如无法转换成数字，则报错，如 1011Q1 没有使用单引号、不包含动态变量且无法转换成数字。
    - 如果比较符的两侧有一侧未识别为字符串或者数字，无法进行比较，则报错。
    - 如果比较符的一侧识别为数字，按照数字比较，如果另一侧无法转换为数字，则报错。
    - 如果比较符的两侧都识别为字符串，按照字符串比较。

字符串比较，示例文件 test.sql：

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

```
-- 与'${Name}' > 'Jack'效果等同
\if ${Name} == 'Jack'
   \echo 'I am Jack'
\else
   \echo 'I am not Jack'
\endif
```

gsql -d -p 25308 --dynamic-param -f test.sql 执行结果：

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

I am Jack
```

数字比较，示例文件 test.sql：

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

-- 如果使用${id} == '01022'，则结果是不等，因为两侧都是字符串，使用字符串比较，结果为不等
\if ${id} == 01022
   \echo 'id is 1022'
\else
   \echo 'id is not 1022'
\endif
```

gsql -d -p 25308 --dynamic-param -f test.sql 执行结果：

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

id is 1022
```

错误比较示例：

```
-- 操作符有一侧无法识别为字符串或数字
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.
-- 操作符一侧数字无法正确转换
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：只支持等值比较，所有情况按照字符串比较。

示例文件 test.sql：

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

\if ${ERROR}
   \echo 'An error occurred in the SQL statement'
```

```
-- equal 比较规则下只支持字符串等值判断，大小比较直接报错，无定界符。下面的效果与${Name} == Jack 等价
\elif '${Name}' == 'Jack'
   \echo 'I am Jack'
\else
   \echo 'I am not Jack'
\endif
```

gsql -d　-p 25308 --dynamic-param -f test.sql 执行结果：

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

I am Jack
```

- \goto \label 跳转示例

  示例文件 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 执行结果：

```
Tom
```

上面的执行结果表示，\goto 元命令实现跳转，直接执行\echo 命令，没有对变量 Name 重新赋值。

- \if 条件块和\goto \label 结合使用示例

  示例文件 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 执行结果：

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

 Count
-------
     3
(1 row)

Count = 3
```

上面的执行结果表示，通过\if 条件块和\goto \label 的结合实现简单的循环。

- \for 循环块使用示例

  为了展示该功能，示例数据如下：

```
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 循环使用示例文件 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 执行结果：

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

  上面的执行结果表示，通过循环块对 SQL 语句的执行结果进行遍历，\loop 和\end-for 之间可以出现更多语句，实现复杂的逻辑。

  如果作为循环条件的 SQL 语句执行失败或者结果集为空，\loop 和\end-for 之间的语句将不被执行。

  示例文件 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 执行结果：

```
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...
                         ^
```

上面的执行结果表示，student_error 这个表不存在，所以 SQL 语句执行失败，\loop 和\end-for 之间的语句将不被执行。

- \exit-for 退出循环

  示例文件 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 执行结果：

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

  表 student 中的数据超过两行，当 id=2 时，使用\exit-for 退出循环，不再继续执行。这个过程中也有与\if 条件块的配合使用。

- \for 循环嵌套

  示例文件 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 执行结果：

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

  通过两层循环获得 Jack、Tom 相关的 course 表中的信息。

PATTERN

很多\d 命令都可以用一个 PATTERN 参数来指定要被显示的对象名称。在最简单的情况下，PATTERN 正好就是该对象的准确名称。在 PATTERN 中的字符通常会被变成小写形式（就像在 SQL 名称中那样），例如\dt FOO 将会显示名为 foo 的表。就像在 SQL 名称中那样，把 PATTERN 放在双引号中可以阻止它被转换成小写形式。如果需要在一

个 PATTERN 中包括一个真正的双引号字符，则需要把它写成两个相邻的双引号，这同样是符合 SQL 引用标识符的规则。例如，\dt "FOO""BAR"将显示名为 FOO"BAR（不是 foo"bar）的表。和普通的 SQL 名称规则不同，不能只在 PATTERN 的一部分周围放上双引号，例如\dt FOO"FOO"BAR 将会显示名为 fooFOObar 的表。

不使用 PATTERN 参数时，\d 命令会显示当前 schema 搜索路径中可见的全部对象——这等价于用*作为 PATTERN。所谓对象可见是指可以直接用名称引用该对象，而不需要用 schema 来进行限定。要查看数据库中所有的对象而不管它们的可见性，可以把*.*用作 PATTERN。

如果放在一个 PATTERN 中，*将匹配任意字符序列（包括空序列），而?会匹配任意的单个字符（这种记号方法就像 Unix shell 的文件名 PATTERN 一样）。例如，\dt int*会显示名称以 int 开始的表。但是如果被放在双引号内，*和?就会失去这些特殊含义而变成普通的字符。

包含一个点号（.）的 PATTERN 被解释为一个 schema 名称模式后面跟上一个对象名称模式。例如，\dt foo*.*bar*会显示名称以 foo 开始的 schema 中所有名称包括 bar 的表。如果没有出现点号，那么模式将只匹配当前 schema 搜索路径中可见的对象。同样，双引号内的点号会失去其特殊含义并且变成普通的字符。

高级用户可以使用字符类等正则表达式记法，如[0-9]可以匹配任意数字。所有的正则表达式特殊字符都按照《开发指南》中的 POSIX 正则表达式所说的工作。以下字符除外：

- .会按照上面所说的作为一种分隔符。
- *会被翻译成正则表达式记号.*。
- ?会被翻译成.。
- $则按字面意思匹配。

根据需要，可以通过书写?、(*R*+|)、(*R*|)和 *R*?来分别模拟 PATTERN 字符.、*R**和 *R*?。$不需要作为一个正则表达式字符，因为 PATTERN 必须匹配整个名称，而不是像正则表达式的常规用法那样解释（换句话说，$会被自动地追加到 PATTERN 上）。如果不希望该 PATTERN 的匹配位置被固定，可以在开头或者结尾写上*。注意在双引号内，所有的正则表达式特殊字符会失去其特殊含义并且按照其字面意思进行匹配。另外，在操作符名称 PATTERN 中（即\do 的 PATTERN 参数），正则表达式特殊字符也按照字面意思进行匹配。

3.6 常见问题处理

连接性能问题

- 数据库内核执行初始化语句较慢导致的性能问题。

  此种情况定位较难，可以尝试使用 Linux 的跟踪命令：strace。

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

  此时便会在屏幕上打印出数据库的连接过程。比如较长时间停留在下面的操作上：

```
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}])
```

此时便可以确定是数据库执行"SELECT VERSION()"语句较慢。

在连接上数据库后，便可以通过执行“explain performance select version()”语句来确定初始化语句执行较慢的原因。更多信息请参考《开发指南》中的“SQL 执行计划介绍”章节。

另外还有一种场景不太常见：由于数据库 CN 所在机器的磁盘满或故障，此时所查询等受影响，无法进行用户认证，导致连接过程挂起，表现为假死。解决此问题清理数据库 CN 的数据盘空间便可。

- TCP 连接创建较慢问题。

  此问题可以参考上面的初始化语句较慢排查的做法，通过 strace 跟踪，如果长时间停留在：

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

  或者

  ```
  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)
  ```

  那么说明客户端与数据库端建立物理连接过慢，此时应当检查网络是否存在不稳定、网络吞吐量太大的问题。

创建连接故障

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

  此问题一般是指定了不可达的地址或者端口导致的。请检查-h 参数与-p 参数是否添加正确。

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

  此问题一般是输入了错误的用户名和密码导致的，请联系数据库管理员，确认用户名和密码的正确性。

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

  此问题是由于环境中使用的 libpq.so 的版本与 gsql 的版本不匹配导致的，请通过“ldd gsql”命令确认当前加载的 libpq.so 的版本，并通过修改 LD_LIBRARY_PATH 环境变量来加载正确的 libpq.so。

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

  此问题是由于环境中使用的 libpq.so 的版本与 gsql 的版本不匹配导致的（也有可能是环境中存在 PostgreSQL 的 libpq.so），请通过“ldd gsql”命令确认当前加载的 libpq.so 的版本，并通过修改 LD_LIBRARY_PATH 环境变量来加载正确的 libpq.so。

- 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?

  此问题是由于网络连接故障造成。请检查客户端与数据库服务器间的网络连接。如果发现从客户端无法 PING 到数据库服务器端，则说明网络连接出现故障。请联系网络管理人员排查解决。

```
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.
  此问题是由于用户不具备访问该数据库的权限，可以使用如下方法解决。
  a. 使用管理员用户 dbadmin 连接数据库。

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

  b. 赋予该用户访问数据库的权限。
  GRANT CONNECT ON DATABASE postgres TO user1;

说明

实际上，常见的许多错误操作也可能产生用户无法连接上数据库的现象。如用户连接的数据库不存在，用户名或密码输入错误等。这些错误操作在客户端工具也有相应的提示信息。

```
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.
  此问题是由于系统连接数量超过了最大连接数量。请联系数据库 DBA 进行会话连接数管理，释放无用会话。
  关于查看用户会话连接数的方法如表 3-24。
  会话状态可以在视图 PG_STAT_ACTIVITY 中查看。无用会话可以使用函数 pg_terminate_backend 进行释放。

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

其中 pid 的值即为该会话的线程 ID。根据线程 ID 结束会话。

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

显示类似如下信息，表示结束会话成功。

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

表3-24 查看会话连接数

描述	命令

描述	命令
查看指定用户的会话连接数上限。	执行如下命令查看连接到指定用户 USER1 的会话连接数上限。其中-1 表示没有对用户 user1 设置连接数的限制。 `SELECT ROLNAME,ROLCONNLIMIT FROM PG_ROLES WHERE ROLNAME='user1';` ` rolname | rolconnlimit` `---------+--------------` ` user1 | -1` `(1 row)`
查看指定用户已使用的会话连接数。	执行如下命令查看指定用户 USER1 已使用的会话连接数。其中，1 表示 USER1 已使用的会话连接数。 `SELECT COUNT(*) FROM V$SESSION WHERE USERNAME='user1';` ` count` `-------` `  1` `(1 row)`
查看指定数据库的会话连接数上限。	执行如下命令查看连接到指定数据库 postgres 的会话连接数上限。其中-1 表示没有对数据库 postgres 设置连接数的限制。 `SELECT DATNAME,DATCONNLIMIT FROM PG_DATABASE WHERE DATNAME='postgres';` ` datname | datconnlimit` `----------+--------------` ` postgres | -1` `(1 row)`
查看指定数据库已使用的会话连接数。	执行如下命令查看指定数据库 postgres 上已使用的会话连接数。其中，1 表示数据库 postgres 上已使用的会话连接数。 `SELECT COUNT(*) FROM PG_STAT_ACTIVITY WHERE DATNAME='postgres';` ` count` `-------` `  1` `(1 row)`
查看所有用户已使用会话连接数。	执行如下命令查看所有用户已使用的会话连接数。 `SELECT COUNT(*) FROM V$SESSION;` ` count` `-------` `  10` `(1 row)`

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

gsql 在向数据库发起连接的时候，会有 5 分钟超时机制，如果在这个超时时间内，数据库未能正常的对客户端请求进行校验和身份认证，那么 gsql 会退出当前会话的连接过程，并报出如上错误。

一般来说，此问题是由于连接时使用的-h 参数及-p 参数指定的连接主机及端口有误（即错误信息中的 xxx 部分），导致通信故障；极少数情况是网络故障导致。要排除此问题，请检查数据库的主机名及端口是否正确。

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

  同时，检查 CN 日志中出现类似如下日志“ FATAL: cipher file "/data/coordinator/server.key.cipher" has group or world access”，一般是由于数据目录或部分关键文件的权限被误操作篡改导致。请参照其他正常实例下的相关文件权限，修改回来便可。

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

  目标 CN 的 pg_hba.conf 里配置了当前客户端 IP 使用"gss"方式来做认证，该认证算法不支持用作客户端的身份认证，请修改到"sha256"后再试。

说明

- 请不要修改 pg_hba.conf 中数据库集群主机的相关设置，否则可能导致数据库功能故障。
- 建议业务应用部署在数据库集群之外，而非集群内部。

其他故障

- 出现因“总线错误”（Bus error）导致的 core dump 或异常退出

  一般情况下出现此种问题，是进程运行过程中加载的共享动态库（在 Linux 为.so 文件）出现变化；或者进程二进制文件本身出现变化，导致操作系统加载机器的执行码或者加载依赖库的入口发生变化，操作系统出于保护目的将进程杀死，产生 core dump 文件。

  解决此问题，重试便可。同时请尽可能避免在升级等运维操作过程中，在集群内部运行业务程序，避免升级时因替换文件产生此问题。

说明

此故障的 core dump 文件的可能堆栈是 dl_main 及其子调用，它是操作系统用来初始化进程做共享动态库加载的。如果进程已经初始化，但是共享动态库还未加载完成，严格意义上来说，进程并未完全启动。

4 Data Studio 数据库集成开发工具

4.1 Data Studio 简介

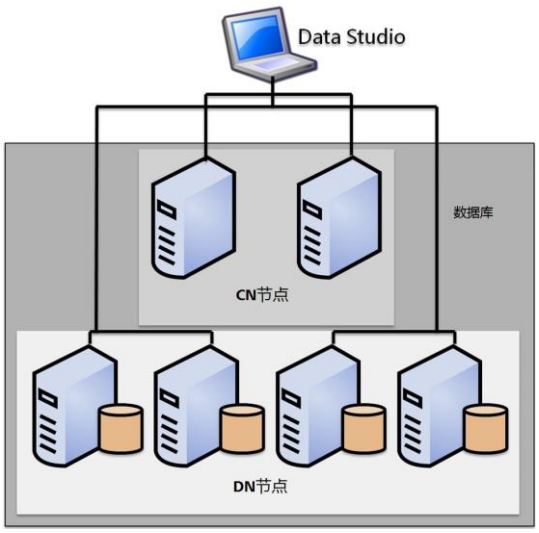
4.1.1 Data Studio 概述

Data Studio 通过提供图形化界面来展示数据库的主要功能，简化了数据库开发和应用构建任务。

数据库开发人员可以使用 Data Studio 所提供的特性，创建和管理数据库对象（数据库对象包含数据库、模式、函数、存储过程、表、序列、列、索引、约束条件、视图等），执行 SQL 语句/SQL 脚本，编辑和执行 PL/SQL 语句，以及导入和导出表数据。

数据库开发人员可在 Data Studio 中通过单步进入、单步退出、单步跳过、继续、终止调试等操作调试并修复 PL/SQL 代码中的缺陷。

数据库及 Data Studio 的操作环境如下图所示。



4.1.2 约束和限制

使用 Data Studio 的约束和限制包含以下所有内容。

对象浏览器过滤树

该过滤树不显示过滤结果数量以及过滤状态。

字符编码

当查看的 SQL 语句、DDL、对象名称或数据中包含中文时，在操作系统支持 GBK 的前提下，Data Studio 客户端字符编码需设置为 GBK。更多信息，请参见会话设置。

连接管理

在“**新建连接**”和“**编辑连接**”窗口的“**高级**”页签的**包含/不包含**字段中，逗号被视为一个分隔符。因此，**包含/不包含**字段不支持包含逗号的模式名称。

数据库表

- 在表创建向导的“**索引**”页签中，列表视图中的所选列在删除后无法保持原有排序。
- 操作完成后，如果 Data Studio 窗口不是当前操作系统的活动窗口，则仅当 Data Studio 窗口变为活动状态时才会显示消息对话框。
- 4.16.7.8 编辑表数据中的操作存在以下限制：
  - 不支持在“**编辑表数据**”页签中输入表达式值。
  - 在 Data Studio 中，仅能编辑获取的记录。
  - 编辑表的过滤条件时，不会高亮 HTML 标签中的搜索内容，如“<”,“&”，或“>”。
  - 包含一个“&”的单元格不会在提示信息中显示。包含两个连续的“&”的单元格会在提示信息中显示为一个“&”。
  - 光标不会停留在新增行。用户需单击需要编辑的单元格。

函数/过程

在“**SQL 终端**”或“**创建函数/过程**”向导创建的函数/过程须以“/”结尾，表示函数/过程的结尾。函数/过程随后输入的语句结尾如果没有“/”，该语句会被视为单条查询，执行过程中可能会报错。

通用

- 在编辑区域一次最多可打开 100 个页签。页签的显示取决于主机的可用资源。
- 数据库对象名最多可包含 64 个字符（仅限文本格式），数据库对象包括数据库、模式、函数、存储过程、表、序列、约束条件、索引和视图。但在 Data Studio 的表达式和说明中使用的字符数没有限制。
- 在 Data Studio 已登录的实例上最多可打开 300 个结果页签。
- 如果“**对象浏览器**”和“**搜索对象**”窗口中加载了大对象，则“**对象浏览器**”中对象展开的速度可能会变慢，同时 Data Studio 也可能会无法响应。
- 对于包含数据的单元格，如果数据超出了可显示区域，调整单元格宽度可能导致 Data Studio 无法响应。

- 表的单元格最多可显示 1000 个字符，超出部分显示为“...”。
  - 如果用户从表或“**结果**”页签的单元格复制数据到任意编辑器（如 SQL 终端/PLSQL 源编辑器、记事本或任意外部编辑器应用），将会粘贴全部数据。
  - 如果用户从表或“**结果**”页签的单元格复制数据到一个可编辑的单元格（本单元格或其他单元格），该单元格仅显示 1000 个字符，并将超出部分显示为“...”。
  - 导出表或“**结果**”页签数据时，导出的文件将包含全部数据。

安全

Data Studio 在首次连接时验证 SSL 连接参数。在后续连接中，Data Studio 不再验证 SSL 连接参数。如果勾选了“**启用 SSL**”，打开新连接时，该连接会使用同样的 SSL 连接参数。

说明

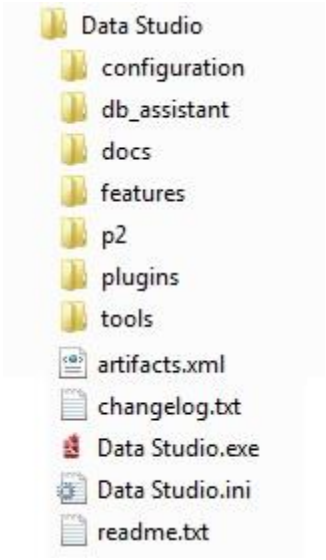
- Data Studio 连接时如果未勾选“**启用 SSL**”默认是未加密的。
- SSL 连接中，如果安全文件被损坏，Data Studio 将无法继续进行任何数据库操作。如果要修复该问题，请删除对应配置文件所在文件夹下的安全文件夹，然后重启 Data Studio。

SQL 终端

- 打开一个包含大量 SQL 语句的 SQL 文件，可能会出现“**内存不足**”错误。更多信息，请参见 4.25 故障处理。
- 对于“**SQL 终端**”页签中被注释掉的文本，Data Studio 不禁用自动建议和超链接功能。
- 如果模式名或表名中有空格或点（.），则不支持超链接。
- 如果对象名称中包含半角单引号（'）或双引号（"），则不支持自动建议功能。
- Data Studio 仅支持对简单的 SELECT 语句进行基本的格式化，对于复杂查询可能无法达到预期效果。

4.1.3 发布包结构

Data Studio 的发布包结构如下图所示。



文件夹/文件	说明
configuration	包含应用启动信息和所需 Eclipse 插件路径信息。
db_assistant	包含“SQL 助手”功能相关的文件。
docs	• 包含《Data Studio 用户手册.pdf》，本手册详细介绍了如何使用 Data Studio 工具。 • 包含在 Data Studio 中使用的开源软件的版权声明、许可证和书面邀约。
features	包含 Eclipse（如富客户端协议 GUI）和 Data Studio 特性。
p2	p2 包含的文件用于提供和管理基于 Eclipse 和 Equinox 的应用。
plugins	包含必须的 Eclipse 和 Data Studio 插件。
tools	包含 Data Studio 的依赖工具。
UserData/ • Autosave • *Logs/* • *Preferences/* • *Profile/* – *History/* • *Security/*	包含每个使用 Data Studio 的 OS 用户各自的文件夹。 **Autosave：**包含自动保存的查询和函数/过程信息。 **Logs：**包含 Data Studio.log 文件，该文件保存 Data Studio 所有操作的日志信息。 **Preferences：**包含 Preferences.prefs 文件，内容为自定义的首选项。 **Profile：**包含 connection.properties 文件、SQL 执行历史、Profiles.txt 文件，用于管理 Data Studio 中的连接信息。

文件夹/文件	说明
	Security：包含 Data Studio 安全管理所需文件。 说明 • User Data 文件夹在首个用户用 Data Studio 打开实例后创建。 • 日志文件夹、语言、内存设置、日志级别对所有用户生效。 • Data Studio 启动后，会创建日志文件夹、*Data Studio.log* 文件、Preferences 文件夹、Preferences.prefs 文件、Profile 文件夹、connection.properties 文件、Profiles.txt 文件和 security 文件夹。 • 如果 Data Studio.ini 文件中指定了日志文件夹路径，日志会在指定路径创建。 • 如果您因安全密钥被损坏，无法登录 Data Studio，请按如下步骤生成新的安全秘钥： 1. 从 Data Studio 文件夹下的 UserData 文件夹中删除 security 文件夹。 2. 重启 Data Studio。
artifacts.xml	包含产品编译版本信息。
changelog.txt	包含当前版本的详细变更信息。
Data Studio.exe/DataStudio.sh	支持连接服务器并执行各种操作，如管理数据库对象、编辑或执行 PL/SQL 程序。
Data Studio.ini	包含 Data Studio 工具运行时的配置信息。
readme.txt	包含当前版本的功能和修复的问题。

4.1.4 系统要求

本节介绍使用 Data Studio 的最低系统要求。

软件要求

操作系统要求

Data Studio 的操作系统配置要求如下表所示。

表4-1 支持操作系统及相应软件包

服务器	操作系统	支持版本
通用 x86 服务器	Microsoft 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)
泰山 ARM 服务器	NeoKylin	7.0

浏览器要求

Data Studio 的浏览器要求如下表所示。

操作系统	版本
Microsoft Windows	IE 11 及以上

其他软件要求

Data Studio 的软件配置要求如下表所示。

表4-2 Data Studio 软件要求

软件	规格
Java	推荐与操作系统位数对应的 Open JDK 1.8 版本
GTK	对于 Linux 操作系统，要求 GTK 最低版本为 GTK 2.24。
GNU libc	显示 DDL，导入 DDL，导出 DDL 和数据操作只支持 GN 系统 libc 2.17 以上的

软件	规格
	版本。

表4-3 支持的数据库版本

数据库	版本
GaussDB(DWS)	1.2.x 1.5.x 8.0.x 8.1.x 8.2.x

说明

保证最佳体验的情况下，推荐的最小屏幕分辨率是 1080 x 768。低于此分辨率，界面会异常。

4.2 安装配置 Data Studio

本节介绍使用 Data Studio 时要遵循的安装和配置步骤，以及配置服务器以调试 PL/SQL 函数的步骤。

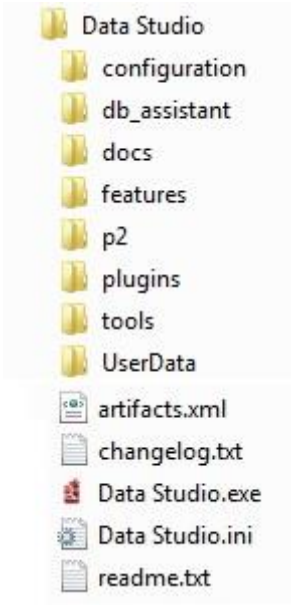
安装 Data Studio

解压安装包后即可运行 Data Studio。

执行以下步骤安装 Data Studio：

步骤 1 解压所需软件包（32 位或 64 位），分别放至 Program Files 或 Program Files（x86）文件夹中。如果用户需用其他文件夹，管理员应控制用户对该文件夹的访问权限。

解压后可以获取如下文件和文件夹：



步骤 2　定位并双击 Data Studio.exe，启动 Data Studio 客户端。

说明

User Data 文件夹在首个用户用 Data Studio 打开实例后创建。打开 Data Studio 时，如果出现任何错误，请参见 4.3 快速入门执行启动操作。

----结束

要创建新的数据库连接，请参见 4.8.2 添加连接。

配置 Data Studio

通过 Data Studio.ini 文件来配置 Data Studio：

说明

参数如有修改，需重新启动 Data Studio 方可查看。配置文件中添加的无效参数会被 Data Studio 忽略。如下描述的所有参数为可选参数。

下表为 Data Studio 相关的配置参数列表。

表4-4 配置参数

参数	说明	取值范围	默认值
-startup	加载 Data Studio 所需的.jar 文件。各版本所需的文件不同。	不适用	*plugins/org.eclipse.equinox.launcher_1.3.100.v20150511-1540.jar*
--launcher.library	加载 Data Studio 所需的库。各版本所需的库不同。	不适用	取决于所使用安装包，可能为 *plugins/org.eclipse.equinox.launcher.win3*

参数	说明	取值范围	默认值
			2.win32.x86_1.1.300.v20150602-1417 或 *plugins/org.eclipse.equinox.launcher.win32.win32.x86_64_1.1.300.v20150602-1417*
-clearPersistedState	删除当前用户界面上的所有缓存内容，重新加载 Data Studio。	不适用	不适用 说明 建议增加该参数。
-consoleLineCount	“消息”窗口显示的最大行数。	1-5000	1000
-logfolder	创建日志文件夹。用户可指定日志文件的保存路径，如果使用默认值“.”，则在 *Data Studio\User data\<用户名>\logs* 创建文件夹。有关详情，请参见“设置日志文件创建位置”。	不适用	-
-loginTimeout	定义创建连接的等待时间，单位为秒。在该参数指定的时长内，Data Studio 会不断尝试连接数据库。如果超时，会提示超时或连接失败。	不适用	180
-data	会话的实例数据位置。	不适用	@none
@user.home/MyAppWorkspace	启动 Data Studio 时，在此位置创建 Eclipse 工作区。 其中，@user.home 指的是 C:/Users/<username>路径。 Eclipse 日志文件保存在以下位置： @user.home/MyAppWorkspace/.metadata	不适用	不适用
-detailLogging	日志记录错误消息。如果设为 True，则日志记录所有错误消息。如果设为 False，则日志仅记录 Data Studio 中明确指定的错误消息。有关详情，请参见“控制故障和错误日志”。 默认情况下不添加此参	True/False	False

参数	说明	取值范围	默认值
	数，如需记录日志，可以手动设置此参数。		
-logginglevel	基于指定的值创建日志文件。如果该值设为取值范围外的任意值或为空，则使用默认值 WARN。有关详情，请参见“不同日志级别类型”。 默认情况下不添加此参数，如需记录日志，可以手动设置此参数。	FATAL、ERROR、WARN、INFO、DEBUG TRACE、ALL 和 OFF	WARN
-focusOnFirstResult	自动定位“结果”页签。 如果设为 false，则自动定位到最新打开的“结果”页签。 如果设为 true，则禁用自动定位功能。	True/False	False
说明 • 以上所有参数必须配置在-vmargs 参数之前。 • -startup 和--launcher.library 参数必须分别配置为第一和第二个参数。			
-vmargs	虚拟机参数的起始位置。 说明 -vmargs 参数必须配置于配置文件的末尾。	不适用	不适用
-vm <file name (javaw.exe) with relative path to Java executable>	定义文件名（javaw.exe）和 Java 的相对路径。	不适用	不适用
-Dosgi.requiredJavaVersion	运行 Data Studio 所需的最低 Java 版本。请勿修改该参数值。	不适用	1.5 说明 推荐的 Java 版本为 1.8.0_141。
-Xms	Data Studio 消耗的初始堆空间。该值必须是 1024 的倍数，大于 40 MB，且小于或等于-Xmx。在值的末尾加上字母 k 或 K 可表示千字节，加上 m 或 M 表示可表示兆字节，g 或 G 可	不适用	-Xms40m

参数	说明	取值范围	默认值
	表示千兆字节。例如： -Xms40m -Xms120m 有关详情，请参见 Java 文档。		
-Xmx	Data Studio 消耗的最大堆空间。可根据可用的 RAM 空间调整该值。在值的末尾加上字母 k 或 K 可表示千字节，加上 m 或 M 表示可表示兆字节，g 或 G 可表示千兆字节。例如： -Xmx1200m -Xmx1000m 有关详情，请参见 Java 文档。	不适用	-Xmx1200m
-OLTPVersionOldST	用于用户配置老版本的 OLTP 版本，用户可以通过登录 gsql，运行 SELECT VERSION()，将获取到的版本号更新 ini 文件中的 OLTPVersionOldST 参数值。	-	-
-OLTPVersionNewST	用于用户配置新版本的 OLTP 版本，用户可以通过登录 gsql，运行 SELECT VERSION()，将获取到的版本号更新 ini 文件中的 OLTPVersionNewST 参数值。	-	-
-testability	开启可测试性需求特性。在该版本中，该功能启用后： • 用户可使用“Ctrl+空格”快捷键复制最近触发的自动建议操作。 • 在选择了“包含 ANALYZE 结果”的场景下，“执行计划和开销”将以树形样式和图形化样式展示。 该参数默认不可用，需手	True/False	False

参数	说明	取值范围	默认值
	动添加。		
-Duser.language	定义 Data Studio 的语言设置。在语言设置更改后添加此参数。	zh/en	不适用
-Duser.country	定义 Data Studio 的国家/地区设置。在语言设置更改后添加此参数。	CN/IN	不适用
-Dorg.osgi.framework.bundle.parent=ext	指定 boot delegation 使用的类加载器。	boot/app/ext	boot
-Dosgi.framework.extensions=org.eclipse.fx.osgi	指定框架扩展名称列表。框架扩展 bundle 是系统 bundle（org.eclipse.osgi）的 fragment。作为 fragment，用户可以提供使用该框架的其他类。	不适用	不适用

说明

- 用户不允许修改以下设置：

  Dorg.osgi.framework.bundle.parent=ext

  Dosgi.framework.extensions=org.eclipse.fx.osgi

- 如果用户收到 SocketException : Bad Address: Connect 消息：

  则用户需检查客户端是否通过 IPv6 或 IPv4 协议建立到服务器的连接。用户还可以根据通过在.ini 文件中配置以下参数来建立连接：

  -Djava.net.preferIPv4Stack=true

  -Djava.net.preferIPv6Stack=false

  表 4-5 列举支持的通信场景。

  第一行和第一列分别代表尝试通信的不同节点类型，x 表示相关节点可以互相通信。

表4-5 通信场景

节点	仅 V4	V4/V6	仅 V6
仅 V4	x	x	无通信可能
V4/V6	x	x	x
仅 V6	无通信可能	x	x

设置日志文件创建位置

步骤 1　打开 Data Studio.ini 文件。

步骤 2　用-logfolder 指定日志文件路径。

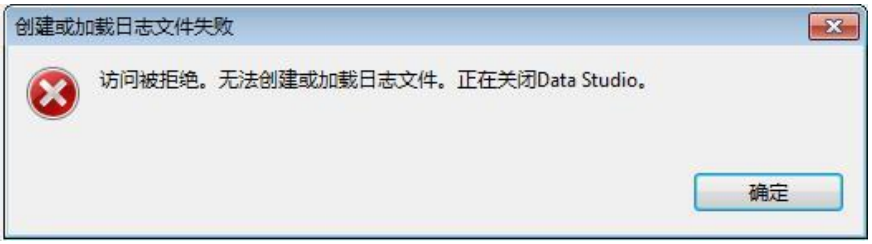
例如：

-logfolder=*c:\test1*

示例中，Data Studio.log 文件创建在 c:\test1\<*用户名*>\logs 路径下。

说明

如果用户没有 Data Studio.ini 文件中指定路径的访问权限，则 Data Studio 会关闭，并弹框显示以下信息。



----结束

出现以下情况时，Data Studio.log 文件在 Data Studio\User Data\<*用户名*>\logs 文件夹中创建：

- Data Studio.ini 文件中没有指定路径。

  例如：-logfolder=.
- 提供的路径不存在。

说明

有关服务器日志详情，请参见服务器手册。

可使用任何文本编辑器打开并查看 Data Studio.log 文件。

控制故障和错误日志

基于程序参数控制错误、异常或者 throw-able 的堆栈运行详情。该参数在 Data Studio.ini 中配置。

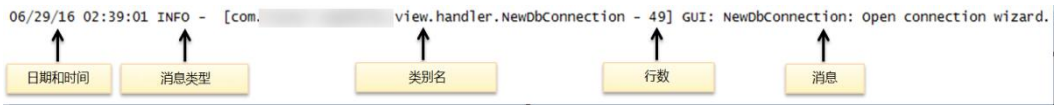
-detailLogging=false

如果标志值为‘true’，错误、异常或者 throw-able 的堆栈运行详情将记录在日志文件中。

如果标志值为‘false’，错误、异常或者 throw-able 的堆栈运行详情将不会记录在日志文件中。

日志消息描述

日志消息描述如下图所示：



Data Studio.log 文件大小达到最大值 10000 KB 时，系统会自动创建一个文件并另存为 Data Studio.log.1。Data Studio.log 中的日志将存储在 Data Studio.log.1 文件中。当 Data Studio.log 文件再次达到最大值，系统继续自动创建一个文件并另存为 Data Studio.log.2。最新日志持续写入 Data Studio.log 文件。以此类推，此过程将一直持续，直到 Data Studio.log.5 文件达到最大值，该循环重新开始。Data Studio 将删除最早的日志文件，即 Data Studio.log.1。例如，Data Studio.log.5 重命名为 Data Studio.log.4，Data Studio.log.4 重命名为 Data Studio.log.3，以此类推。

说明

如需启用服务器日志文件的性能日志记录功能，需启用配置参数 log_min_messages，且设置为 data/postgresql.conf 配置文件中的 debug1，即 log_min_messages = debug1。

不同日志级别类型

Data Studio.log 文件中所显示的不同类型的日志级别如下：

- TRACE：相比 DEBUG 级别，TRACE 级别提供更为详细的信息。
- DEBUG：DEBUG 级别指粒状信息事件，对调试应用程序最为有用。
- INFO：INFO 级别指着重显示应用进程的消息。
- WARN：WARN 级别指潜在的有害情况。
- ERROR：ERROR 级别指错误事件。
- FATAL：FATAL 级别指事件造成应用终止。
- ALL：ALL 级别指启用所有日志级别。
- OFF：OFF 级别指禁用所有日志级别和 ALL 级别相反。

说明

- 如果用户输入无效的日志级别值，日志级别会设置为 WARN。
- 如果用户未指定日志级别，日志级别会设置为 WARN。

日志记录会输出高于或等于其日志级别的所有消息。

标准 log4j 级别的顺序如下：

表4-6 日志级别

-	FATAL	ERROR	WARN	INFO	DEBUG	TRACE
OFF	x	x	x	x	x	x
FATAL	√	x	x	x	x	x
ERROR	√	√	x	x	x	x

-	FATAL	ERROR	WARN	INFO	DEBUG	TRACE
WARN	√	√	√	x	x	x
INFO	√	√	√	√	x	x
DEBUG	√	√	√	√	√	x
TRACE	√	√	√	√	√	√
ALL	√	√	√	√	√	√
√ - 创建日志文件 x- 不创建日志文件						

4.3 快速入门

本节介绍 Data Studio 的启动步骤。

前置条件

使用 StartDataStudio.bat 文件检查操作系统、 Java 和 Data Studio 的版本。

步骤 1 在 4.1.3 发布包结构压缩文件内，点击跳转到 Tools 文件夹，双击 StartDataStudio.bat 执行文件并检查 Java 版本兼容性。

批量文件检查版本兼容性并打开 Data Studio，或根据安装的操作系统、Java 和 Data Studio 版本，显示相应的信息。

如果安装的 Java 版本低于 1.8，可能会弹出错误消息。

批量文件检查如下场景，用于确认 Data Studio 的操作系统和 Java 版本：

DS 安装（32/64 位）	操作系统（位）	Java（位）	结果
32	32	32	打开 Data Studio
32	64	32	打开 Data Studio
32	64	64	弹出错误消息
64	32	32	弹出错误消息
64	64	32	弹出错误消息
64	64	64	打开 Data Studio

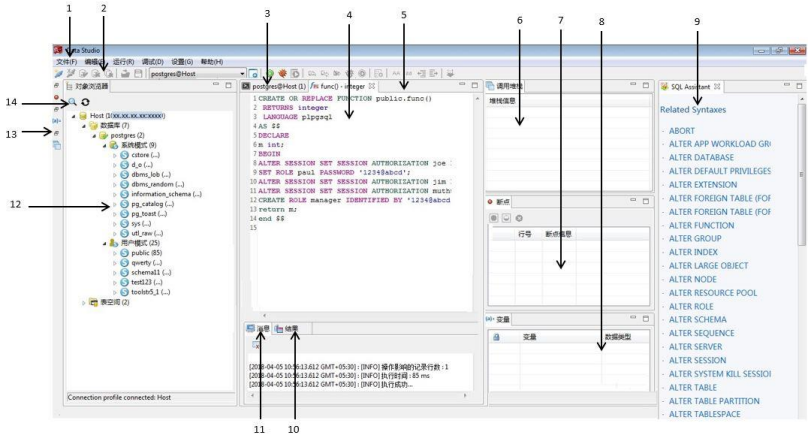
----结束

4.4 Data Studio 用户界面

本节介绍 Data Studio 用户界面。

Data Studio 主界面包括：

1. 主菜单：提供使用 Data Studio 的基本操作。
2. 工具栏：提供常用操作入口。
3. “**SQL 终端**”页签：在该窗口，可以执行 SQL 语句和函数/过程。
4. “**PL/SQL Viewer**”页签：显示函数/过程信息。
5. 编辑区域用于进行编辑操作。
6. “**调用堆栈**”窗格：显示执行栈。
7. “**断点**”窗格：显示所有设置过的断点。
8. “**变量**”窗格：显示变量及其变量值。
9. “**SQL Assistant**”页签显示“SQL 终端”和“PL/SQL Viewer”页签中输入信息的建议或参考。
10. “**结果**”页签：显示所执行的函数/过程或 SQL 语句的结果。
11. “**消息**”页签：显示进程输出。显示标准输入、标准输出和标准错误。
12. “**对象浏览器**”窗格：显示数据库连接的层级树形结构和用户有权访问的相关数据库对象。除公共模式外，所有默认创建的模式均分组在“**系统模式**”下，用户模式分组在相应数据库的“**用户模式**”下。
13. “**最小化窗口窗格**”：用于打开“**调用堆栈**”、“**断点**”和“**变量**”窗格。该窗格仅在“**调用堆栈**”、“**断点**”、“**变量**”窗格中的一个或多个窗格最小化时显示。
14. 搜索工具栏用于在“**对象浏览器**”窗格中搜索对象。



4.5 Data Studio 菜单

4.5.1 “文件”菜单

“**文件**”菜单包含了数据库连接选项。在主菜单中单击“**文件**”，按下“Alt+F”打开“**文件**”菜单。

功能	按钮	快捷键	说明
新建连接		Ctrl+N	在“**对象浏览器**”和“**SQL 终端**”窗格中，建立新的数据库连接。
删除连接		-	从“**对象浏览器**”窗格中删除已选择的数据库连接。
打开连接		-	连接到数据库。
断开连接		Ctrl+Shift+D	断开和指定数据库的连接。
断开所有连接		-	断连某一连接下所有数据库。
打开		Ctrl+O	在“**SQL 终端**”中加载 SQL 查询。
保存		Ctrl+S	将“**SQL 终端**”中的 SQL 脚本保存到 SQL 文件中。
另存为		CTRL+ALT+S	将“**SQL 终端**”中的 SQL 脚本保存到新的 SQL 文件中。
退出	-	Alt+F4	退出 Data Studio 工具并断开连接。 **说明** 未保存的内容将会丢失。

关闭 Data Studio

执行以下步骤关闭 Data Studio：

步骤 1 单击  。

或者选择“**文件>退出**”。

系统显示“**退出应用程序**”退对话框，提示用户进行选择。

步骤 2 根据需要单击按钮。

- **强制退出**：退出前不保存历史执行 SQL 信息。

说明

若单击“**强制退出**”，未保存的历史执行 SQL 可能丢失。

- **标准退出**：退出前将未保存的历史执行 SQL 保存到磁盘。
- **取消**：不退出应用程序。

----结束

4.5.2“编辑”菜单

“**编辑**”菜单支持“**PL/SQL Viewer**”和“**SQL 终端**”页签中的剪切、复制、粘贴、格式化、全选、查找和替换、搜索对象等操作。按下“Alt+E”打开“**编辑**”菜单。

功能	快捷键	说明
剪切	Ctrl+X	剪切选中的文本。
复制	Ctrl+C	复制选中的文本或修饰的对象名称。
粘贴	Ctrl+V	粘贴选中的文本或修饰的对象名称。
格式化	Ctrl+Shift+F	格式化所有的 SQL 语句和函数/过程。
全选	Ctrl+A	在“**SQL 终端**”选中所有的文本。
查找和替换	Ctrl+F	在“**SQL 终端**”查找替换文本。
搜索对象	Ctrl+Shift+S	在连接的数据库中搜索对象。
撤销	Ctrl+Z	撤销上一步操作。
重做	Ctrl+Y	恢复上一步操作。
大写	Ctrl+Shift+U	将所选文本改为大写
小写	Ctrl+Shift+L	将所选文本改为小写
转到行	Ctrl+G	跳转到“**SQL 终端**”或“**PL/SQL Viewer**”页签的特定行。
行注释/取消行注释	Ctrl+/	单独注释/取消注释每个选

功能	快捷键	说明
		中的行
块注释/取消块注释	Ctrl+Shift+/	注释/取消注释所有选中的行或整段内容

复制

复制功能可用于从“**对象浏览器**”窗格中复制对象。

复制对象如下：

表4-7

对象类型	复制格式
函数/过程	模式.对象名称(参数名称 参数类型,...）
数据库	对象名称
模式	对象名称
列	对象名称
约束	对象名称
分区名称	对象名称
其他	模式.对象名称

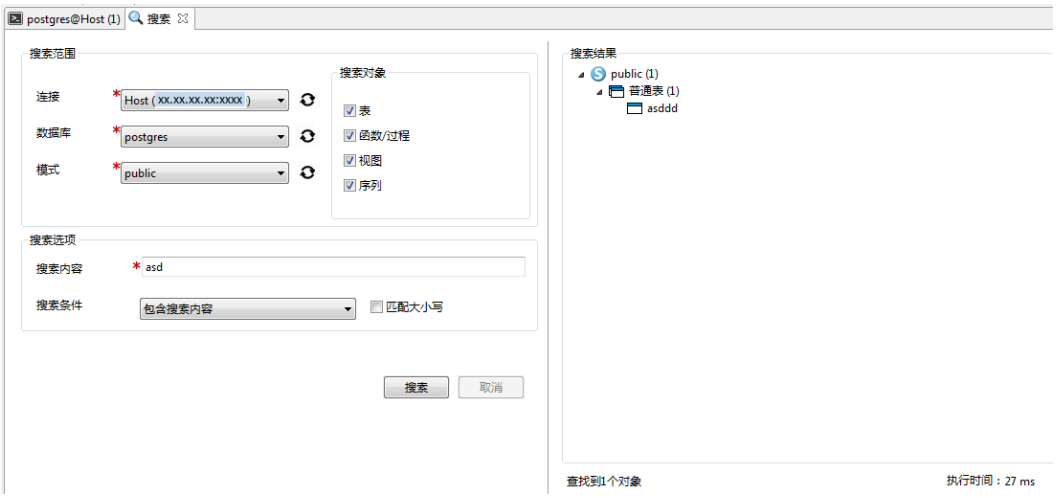
搜索对象

可使用“**搜索对象**”选项从“**对象浏览器**”窗格中按搜索条件搜索对象。搜索方法：

从菜单中选择“**编辑** > **搜索对象**”或从”**对象浏览器**“工具栏单击图标  进行搜索。显示的搜索结果为树型结构，与对象浏览器中的对象类似。右键菜单中除“**刷新**”之外的操作，均可对搜索结果中的对象执行。经过删除、模式设置和重命名等操作修改后的对象，在页面刷新之后仅可从主对象浏览器查看。右键菜单中对群组名称（如表、模式和视图等）的操作选项无法对搜索结果中的对象执行。只有用户有权访问的对象才能被搜索到。用户无权访问的对象不会出现在“搜索范围”中。

说明

在“**搜索**”窗口，单击对象类型末尾的刷新选项，可以查看新增加的对象。



支持的搜索条件选项

搜索选项	搜索行为
包含搜索内容	显示包含搜索内容的搜索文本。
以搜索内容为开头	显示以搜索内容开头的搜索文本。
全字匹配	以被搜索字符精准匹配搜索文本。
正则表达式	使用正则表达式的搜索文本会在对象浏览器中搜索其形式满足条件的文本。可在“搜索条件”中选择“正则表达式”进行搜索。详情请参阅 POSIX 正则表达式规则。 例如， • 输入^a 可查找所有以字母 a 开头的对象。 • 输入^[^A-Za-z]+$可查找不包含字母的对象。 • 输入^[^0-9]+$可查找所有不包含数字的对象。 • 输入^[a-t][^r-z]+$可查找所有以 a~t 之间字母开头，且不包含 r~z 之间字母的对象。 • 输入^e.*a$可查询所有以字母 e 开头并以字母 a 结尾的对象。 • 输入^[a-z]+$并选择“**匹配大小写**”，可查询只包含小写字母的对象。 • 输入^[A-Z]+$并选择“**匹配大小写**”，可查询只包含大写字母的对象。 • 输入^[A-Za-z]+$ 并选择“**匹配大小写**”,可查询包含大小写字母的对象。 • 输入^[A-Za-z0-9]+$并选择“**匹配大小写**”,可查询包含大小写字母和数字的对象。 • 输入^".*"$，可查询引号内所有对象。

下划线（_）和百分比符号（%）搜索

搜索值	搜索行为
_	含下划线（_）的搜索文本中，下划线被视为单个字符的通配符。这不适用于正则表达式、以搜索内容为开头和全字匹配。 例如： • 输入_ed 可查询以单个字符开头的所有对象，后跟“ed”。 • 输入 D_t_e 可查询含有字符“d”的所有对象，后依次跟单个字符、字符“t”、单个字符和字符“e”。
%	含英文百分号（%）的搜索文本中，该百分号被视为多个字符的通配符。这不适用于正则表达式、以搜索内容为开头和全字匹配。 例如： • 输入_%ed 可查询含有“ed”字符的所有对象。 • 输入 D%t%e 可查询含有字符“d”的所有对象，后依次跟任意数量的字符、字符“t”、任意数量的字符和字符“e”。

勾选“匹配大小写”后执行搜索，将查找与搜索文本大小写匹配的内容。

4.5.3 “运行”菜单

“**运行**”菜单提供在“**PL/SQL Viewer**”页签中执行数据库操作以及在“**SQL 终端**”页签中执行 SQL 语句的选项。按下“Alt+R”打开“**运行**”菜单。

功能	按钮	快捷键	说明
执行		Ctrl+E	启动 normal 模式下执行指定函数/过程。 在“**结果**”页签中显示结果。 在“**消息**”页签显示执行的动作信息。
编译/执行声明		Ctrl+Enter	编译函数/过程。 在“**SQL 终端**”页签启动执行 SQL 语句。
新增页签编译/执行语句		Ctrl+Alt+Enter	保留当前页签，在新增页签中执行语句。 如果选择“**保留当前结果集**”，则该

功能	按钮	快捷键	说明
			功能不可用。
终止		Shift+Esc	终止正在执行的查询。 在“**结果**”页签中显示结果。 在“**消息**”页签显示执行的动作信息。

4.5.4 “调试”菜单

“**调试**”菜单支持“**PL/SQL Viewer**”和“**SQL 终端**”页签中的调试操作。按下“Alt+D”打开“**调试**”菜单。

功能	按钮	快捷键	说明
调试		Ctrl+D	启动调试流程。
继续步骤		F9	继续调试。
终止调试		F10	终止调试。
单步进入		F7	进入流程。
单步跳过		F8	跳过流程。
单步退出		Shift+F7	跳出流程。

4.5.5 “设置”菜单

“**设置**”菜单包含语言变更选项。按下“Alt+G”打开“**设置**”菜单。

功能	快捷键	说明
语言	-	修改 Data Studio 的界面语言。
首选项	-	修改 Data Studio 的偏好设置。

设置界面语言

界面语言为英文时，执行如下步骤将界面语言修改为中文。

步骤 1 选择“**设置** > **语言** > **(zh_CN)中文（简体）（C）**”。

显示“**重启 Data Studio**”对话框。

说明

保存所有数据，然后重新设置语言。

步骤 2 单击“**是**”。

所有连接关闭，准备重启。

如果选择“**否**”，即使 Data Studio 重启也不会改变当前语言。

----结束

4.5.6 “帮助”菜单

“**帮助**”菜单提供 Data Studio 用户手册及版本信息。按下“Alt+H”打开“**帮助**”菜单。

功能	快捷键	说明
用户手册	F1	打开 Data Studio 用户手册。
关于	-	显示 Data Studio 当前版本及版权信息。

说明

- 请参阅 https://java.com/en/download/help/path.xml 以设置 Java Home 路径。

4.6 Data Studio 工具栏

工具栏如下图所示。

工具栏提供如下操作功能：

- 4.8.2 添加连接
- 4.8.5 删除连接
- 4.9.3 连接到数据库
- 4.9.4 断开连接
- 4.9.2 断开所有连接
- 4.20.3 打开并保存 SQL 脚本
- 4.20.3 打开并保存 SQL 脚本
- 4.20.1 打开多个“SQL 终端”页签
- 4.20.1 打开多个“SQL 终端”页签

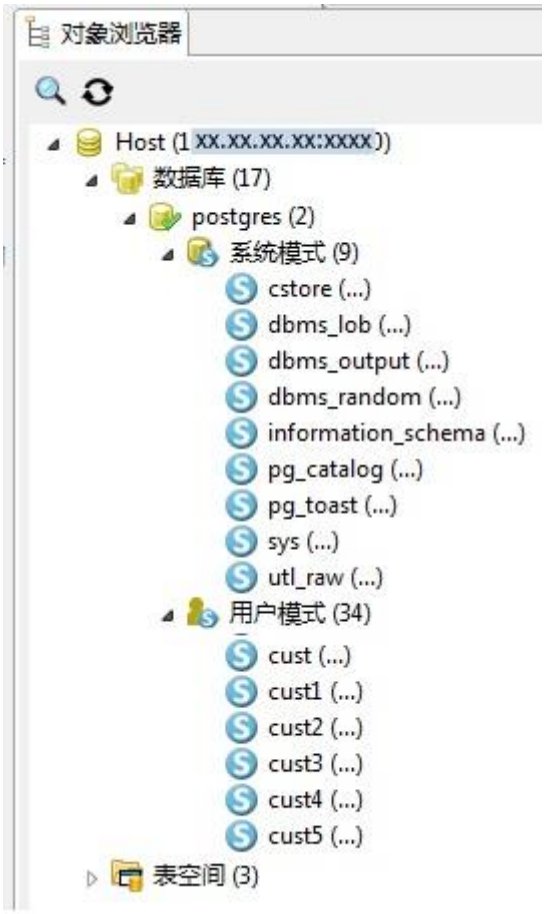
- 4.15.6 执行函数/过程
- 4.14 调试 PL/SQL 函数
- 4.12 编辑函数/过程
- 4.14.3 控制执行
- 4.14.3 控制执行
- 4.14.3 控制执行
- 终止调试
- 继续调试
- 4.20.8 查看执行计划和开销
- 4.20.9 图形化查看执行计划和开销
- 4.20.6 SQL 查询格式化
- 大写
- 小写
- SQL Assistant

4.7 Data Studio 右键菜单

本节介绍 Data Studio 的右键菜单。

“对象浏览器”窗格

“**对象浏览器**”窗格如下图所示。



右键单击连接名称，可以选择“**重命名连接**”、“**编辑连接**”、“**删除连接**”、“**属性**”和“**刷新**”选项。

菜单项	快捷键	说明
重命名连接	-	重命名连接
编辑连接	-	修改连接详细信息
删除连接	-	删除存在的数据库连接
属性	-	查看连接详细信息
刷新	F5	刷新连接

右键单击“**数据库**”，可以选择“**创建数据库**”、“**断开所有连接**”和“**刷新**”选项。

菜单项	快捷键	说明
创建数据库	-	在此连接下创建一个新的数据库
断开所有连接	-	断开此连接下的所有数据库

菜单项	快捷键	说明
刷新	F5	刷新数据库分组

右键单击连接中的数据库，可以选择“**断开连接**”、“**打开新的终端**”、“**属性**”和“**刷新**”选项。

菜单项	快捷键	说明
断开连接	Ctrl+Shift+D	断开和数据库的连接
打开新的终端	Ctrl+T	在此连接下打开新的终端
属性	-	显示数据库属性
刷新	F5	刷新数据库

右键单击断连的数据库，可以选择“**打开连接**”、“**重命名**”和“**删除**”选项。

菜单项	快捷键	说明
打开连接	-	连接数据库
重命名	-	重命名数据库
删除	-	删除数据库

右键单击“**系统模式**”，可选择“**刷新**”选项。

菜单项	快捷键	说明
刷新	F5	刷新函数/过程。

右键单击“**用户模式**”，可选择“**创建模式**”、“**授权/撤销权限**”和“**刷新**”选项。

菜单项	快捷键	说明
创建模式	-	创建新模式
授权/撤销权限	-	为用户模式组授权或撤销权限
刷新	F5	刷新模式

右键单击模式名称，可以选择“**导出 DDL**”、“**导出 DDL 和数据**”、“**重命名**”、“**删除**”、“**授权/撤销权限**”、“**刷新**”选项。

菜单项	快捷键	说明
导出 DDL	-	导出模式的 DDL
导出 DDL 和数据	-	导出模式的 DDL 和数据
重命名	-	重新命名模式
删除	-	删除模式
授权/撤销权限	-	为模式授权或撤销权限
刷新	F5	刷新模式

右键单击“**函数/过程**”，可以选择“**创建 PL/SQL 函数**”、“**创建 PL/SQL 过程**”、“**创建 SQL 函数**”、“**创建 C 函数**”、“**授权/撤销权限**”、“**刷新**”选项。

菜单项	快捷键	说明
创建 PL/SQL 函数	-	创建 PL/SQL 函数
创建 PL/SQL 过程	-	创建 PL/SQL 过程
创建 SQL 函数	-	创建 SQL 函数
创建 C 函数	-	创建 C 函数
授权/撤销权限	-	为函数/过程授权或撤销权限
刷新	F5	刷新函数/过程

右键单击“**普通表**”，可以选择“**创建普通表**”、“**创建分区表**”、“**授权/撤销权限**”和“**刷新**”选项。

菜单项	快捷键	说明
创建普通表	-	创建普通表
创建分区表	-	创建分区表
授权/撤销权限	-	为表授权或撤销权限
刷新	F5	刷新表

右键单击“**视图**”，可以选择“**创建视图**”、“**授权/撤销权限**”和“**刷新**”选项。

菜单项	快捷键	说明
创建视图	-	创建视图
授权/撤销权限	-	为视图授权或撤销权限

菜单项	快捷键	说明
刷新	F5	刷新视图

右键单击“**PL/SQL Viewer**”，可以选择“**剪切**”、“**拷贝**”、“**粘贴**”、“**全选**”、“**行注释/取消行注释**”、“**块注释/取消块注释**”、“**编译**”、“**执行**”、“**添加到监视器**“、”**使用回滚进行调试**“和“**调试**”选项。

右键单击选项	快捷键	说明
剪切、拷贝、粘贴	Ctrl+X、 Ctrl+C、 Ctrl+V	剪切板选项
全选	Ctrl+A	在“PL/SQL Viewer”页签选择内容
行注释/取消行注释	-	单独注释/取消注释每个选中的行
块注释/取消块注释	-	注释/取消注释所有选中的行或整段内容
编译	-	编译函数/过程
执行	-	执行函数/过程
添加到监视器	-	添加变量到监视器窗口
使用回滚进行调试	-	调试函数/过程，并在调试完成之后回滚
调试	-	调试函数/过程

右键单击“**SQL 终端**”，可以选择“**剪切**”、“**拷贝**”、“**粘贴**”、“**全选**”、“**执行语句**”、“**打开**”、“**保存**”、“**查找和替换**”、“**执行计划**”、“**另存为**”、“**格式化**”、“**取消**”、“**行注释/取消行注释**”以及“**块注释/取消块注释**”选项。

右键单击选项	快捷键	说明
剪切、拷贝、粘贴	Ctrl+X, Ctrl+C, Ctrl+V	剪切板选项
全选	-	选中所有文本
执行语句	-	执行查询
打开	-	打开文件
保存	-	保存查询
查找和替换	-	在“SQL 终端”页签查找并替换文本
执行计划	-	执行查询
行注释/取消行注释	Ctrl+/	单独注释/取消注释每个选中的行

右键单击选项	快捷键	说明
块注释/取消块注释	Ctrl+Shift+/	注释/取消注释所有选中的行或整段内容
取消	-	取消执行
另存为	CTRL+ALT+S	保存查询到新文件
格式化	CTRL+SHIFT+F	根据查询中配置的规则格式化 SQL 语句

右键单击“**消息**”，可以选择“**拷贝**”、“**全选**”和“**清除**”选项。

右键单击选项	快捷键	说明
拷贝	Ctrl+C	复制文本
全选	Ctrl+A	选中所有文本
清除	-	清除文本

4.8 连接信息

4.8.1 概述（连接信息）

Data Studio 启动后，默认打开“创建数据库连接”对话框。要执行数据库操作，Data Studio 需连接至少一个数据库。

输入连接参数，创建 Data Studio 到数据库服务器的连接。将光标悬停在连接名称上，可查看数据库信息。

说明

所有必选参数均需要填写。必选参数用星号（*）标识。

4.8.2 添加连接

执行以下步骤创建数据库连接：

步骤 1 在主菜单中选择“**文件** > **新建连接**”，或

单击工具栏上的或按“Ctrl+N”连接到数据库服务器，弹出“**新建/选择数据库连接**”对话框。

说明

建立连接时，如果首选项文件损坏或首选项值无效，会显示如下错误信息，提示用户首选项值无效，并恢复默认值。单击“确定”完成建立新数据库连接的操作

步骤 2 该连接对话框的左侧列表中显示已有连接信息和服务器信息。

说明

服务器信息在连接成功后方可显示。

- 双击连接名称，可自动填充“**名称**”、“**主机名**”和“**端口号**”等连接参数。

说明

如果任何现有连接信息的密码或密钥损坏，那么无论使用哪个连接，均需要手动填写连接密码。

- 若单击  ，在数据库不同连接状态下会出现不同提示信息：
  - 如果数据库连接已激活，会弹出“**确认删除连接**”对话框。单击“**是**”断开所有数据库连接。
  - 如果数据库连接未激活，会弹出“**删除连接**”对话框。
- 如果未选择连接名，直接单击  ，会显示对话框，提示用户至少选择一条连接信息。

步骤 3 设置如下参数，创建数据库连接：

说明

- 单击“清除”可清除“创建数据库连接”对话框中的所有字段。
- 使用快捷键（Ctrl+V）在“新建/选择数据库连接”窗口中粘贴数据。Data Studio 的对话框中无法使用右键菜单选项。

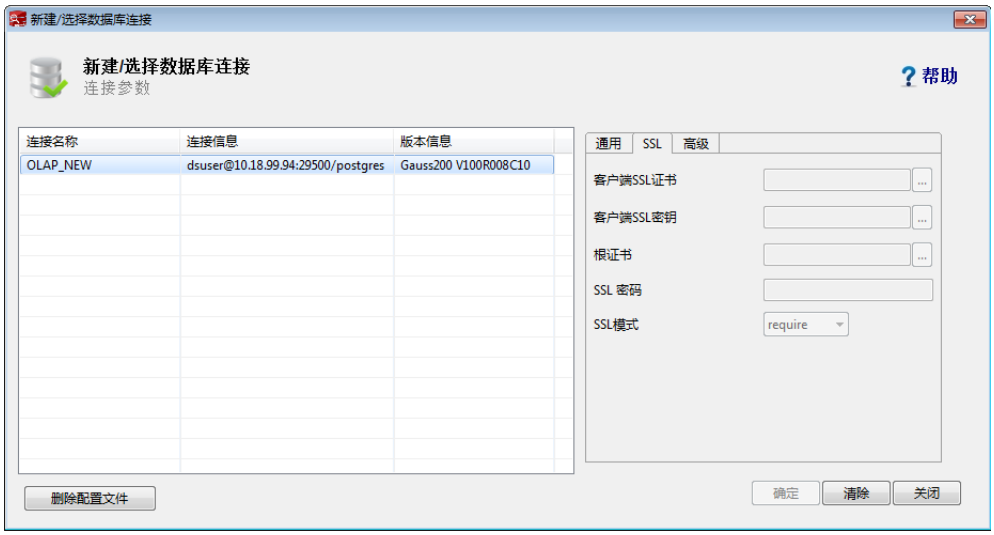
字段名称	说明	举例
数据库类型	数据库类型	-
名称	连接名称	My_Connection_DB
主机	主机 IP 地址（IPv4）或数据库域名 说明 • 如果域名长度大于 25 个字符，则域名将不会完整显示。 例如：*test1(db.dws...com:25xxx)* • 一旦建立连接，将鼠标悬停在连接名称上将显示服务器 IP 地址和版本。 • 如果此字段中的条目含有带三个分隔符（.）的数字格式，则该条目会被视为 IP 地址。不符合该格式的条目会被视为域名。 • 域名必须满足以下条件: • 以字母开头。 • 可包含字母、数字、连字符（-）和	db.dws.mycloud.com 10.xx.xx.xx

字段名称	说明	举例
	英文句点（.）。不允许出现其他任何特殊字符。 • 不得出现空格或制表符。 • 长度不能超过 253 个字符，在两个句点之间最多可出现 63 个字符。	
端口号	端口地址	8000
数据库	数据库名称。	gaussdb
用户名	连接所选数据库的用户名	-
密码	连接数据库的用户名密码。此密码文本以掩码显示。	-

- 从“**保存密码**”中选择一个选项。可选项包括：
  - “**永久保存**”：退出数据库后仍然保存密码。首次建立连接时，此选项将不可用。要隐藏或查看该下拉选项，请参见“密码”小节。
  - “**仅当前会话**”：仅在当前会话中保存密码。
  - “**不保存**”：不保存密码。如果选择该选项，Data Studio 会在用户进行如下操作时要求输入密码：
    - 4.9.1 创建数据库
    - 4.9.5 重命名数据库
    - 4.14 调试 PL/SQL 函数
    - 4.20.10 使用 SQL 终端
- “**启用 SSL**”默认选中。

步骤 4 执行以下步骤启用 SSL：

1. 选中“**启用 SSL**”。
2. 单击“**SSL**”页签。



3. 设置以下信息，提供以下文件以使用安全连接。请参见 4.24.7 SSL 证书。

   - 选择“**客户端 SSL 证书**”，单击 ，选择客户端 SSL 证书。
   - 选择“**客户端 SSL 密钥**”，单击 并选择客户端 SSL 密钥。
   - 如需选择“**根证书**”，单击 并选择根证书。
   - 可从“**SSL 模式**”中选择 SSL 模式。有关各模式的详情，请参见下表。

说明

- 如果将“SSL 模式”设为 verify-ca 或 verify-full，则必须选择根证书。
- Data Studio 首次访问 gs-dump 特性时,会弹框要求输入客户端秘钥。

SSL 模式	说明
require	如果选择 require，则不会验证证书有效性，因为所使用的 SSL factory 无需验证。
verify-ca	如果选择 verify-ca，则会使用相应的 SSL factory 检查 CA 是否正确。
verify-full	如果选择 verify-full，则会使用相应的 SSL factory 检查 CA 和数据库是否正确。

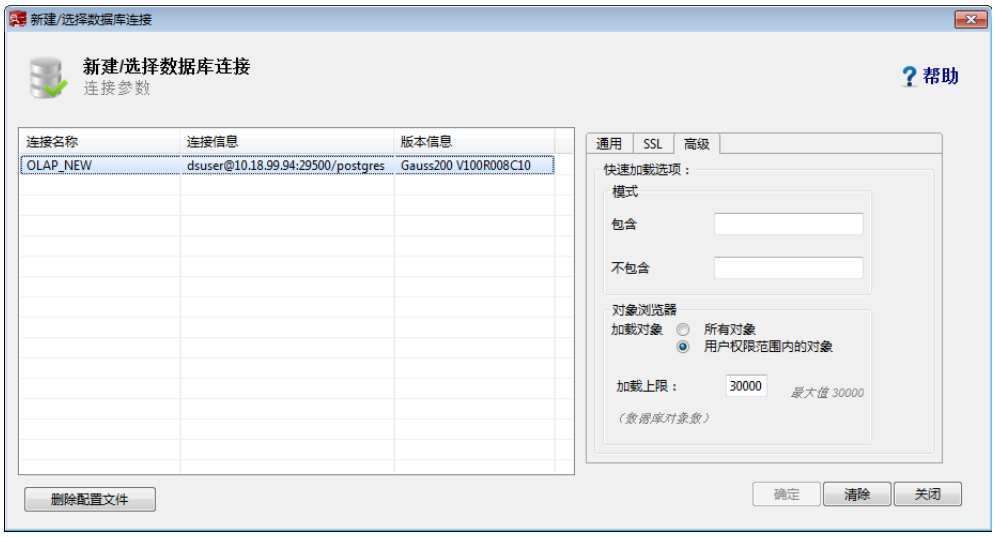
说明

- 选择“客户端 SSL 证书”和“客户端 SSL 密钥”，可使用安全连接从 Data Studio 中导出 DDL 和数据。
- 如果为“客户端 SSL 证书”和“客户端 SSL 密钥”选择了无效的文件，将导致导出失败。有关详情，请参见 4.25 故障处理。
- 如果取消选中“启用 SSL”复选框并继续操作，则会弹出“连接安全告警”对话框。要设置是否显示该安全告警，请参见安全免责声明。

- “继续”：单击“继续”，继续使用不安全的连接。
- “取消”：单击“取消”并启用 SSL。
- “不再显示”：如果勾选该字段，当前登录的 Data Studio 实例在后续连接时，不再显示“连接安全告警”对话框。
- 有关详情，请参见服务器手册。

步骤 5 按照以下步骤设置“**快速加载选项**”：

1. 单击“**高级**”选项卡。



2. 建立连接时，在“**包含**”字段中输入模式名称（使用逗号作为分隔符），以优先加载这些选项。
3. 建立连接时，在“**不包含**”字段中输入模式名称，以避免优先加载这些选项。
4. 为“**加载对象**”选择以下任意一个值：
   - “**所有对象**”：加载所有对象。
   - “**用户权限范围内的对象**”：仅加载用户有权访问的对象。请参阅表 4-29 查看对象浏览器中列出的对象所需的最低访问权限。

说明

默认选择“用户权限范围内的对象”。

5. 在“**加载上限**”字段中输入加载限制。允许的最大值为 30,000。该值为数据库对象个数。

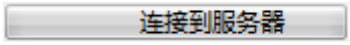
说明

- 如果“包含”字段中输入的模式中对象类型（表、视图等）的数量大于“加载上限”中的值，则将只加载该模式的父对象。这意味着含有大于 3 个参数的列、约束、索引、函数等子对象将不会被加载。
- 对“包含”和“不包含”列表中提供的模式名称进行验证。
- 如果无法访问“包含”字段中指定的模式，则连接期间会显示该模式的错误消息。
- 如果无法访问“不包含”字段中指定的模式，则建立连接后，模式不会在“对象浏览器”中加载。

步骤 6 单击“**确定**”建立连接。

状态栏显示已完成操作的状态。

Data Studio 连接数据库时，状态栏显示连接状态，如下图所示：



一旦建立连接，“**对象浏览器**”窗格中会显示所有模式。

说明

- 密码失效的情况下您也可登录到 Data Studio，但系统会提示您某些操作可能无法正常工作。请参阅密码过期获取详细信息。
- 如要取消连接，请参阅“取消连接”。
- Postgres 模式名不在“对象浏览器"窗格显示。

----结束

取消连接

执行以下步骤取消连接：

步骤 1 单击“**取消**”。

显示“**取消连接**”对话框。

步骤 2 单击“**是**”。

显示消息确认会话框。

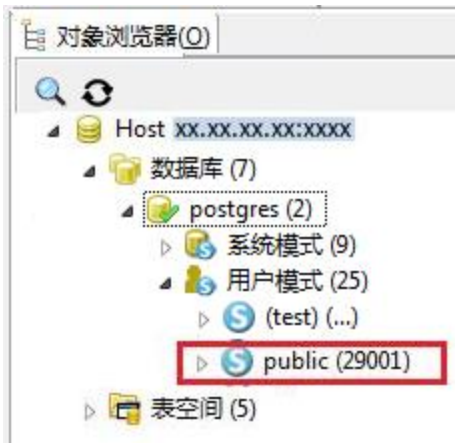
步骤 3 单击“**确认**”。

----结束

懒加载

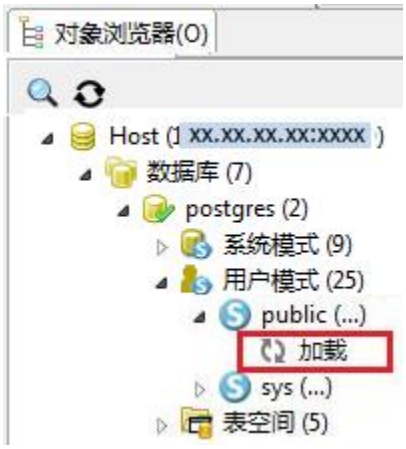
懒加载功能仅在用户有需要时加载对象。

连接到数据库时，仅加载 search_path 路径下保存的模式子对象，如下所示：



未加载的模式显示为“*模式名*（…）”。

要加载子对象，请展开模式。模式展开时，会显示模式下的对象正在加载中：



说明

如果有对象正在加载，且用户尝试加载未加载的对象，则会弹出消息告知用户有对象正在加载中。未加载对象旁的 ▷ 会消失。如要加载该对象，在对象或数据库级别进行刷新，该按钮会重新显示。

展开模式来加载和查看子对象。“对象浏览器”一次只能加载一个模式下的子对象。

如果建立连接后修改了 search_path，该修改在重新连接数据库后才会生效。自动建议适用于用户有权访问的所有模式对象的关键字、数据类型、模式名称、表名称、视图和表别名。

“对象浏览器”窗格 1 分钟内最多可加载 50000 个对象。

数据库连接超时时间默认为 3 分钟（180 秒）。如果在此时间段内未能连接成功，会显示超时错误。

可以在 Data Studio.ini 文件中设置 loginTimeout。该文件的路径为 Data Studio\Data Studio.ini。

说明

当用户登录 Data Studio 时，系统会自动加载 pg_catalog。

4.8.3 重命名连接

执行以下步骤重命名数据库连接：

步骤 1 在“**对象浏览器**”窗格中，右键单击连接名称，然后选择“**重命名连接**”。

工具会显示“**重命名连接**”对话框，提示您输入新的连接名称。

步骤 2 输入新的连接名称。单击“**确定**”重命名连接。

状态栏显示已完成操作的状态。

说明

新的连接名称必须唯一，否则重命名操作将失败。

----结束

4.8.4 编辑连接

执行以下步骤编辑数据库连接属性：

步骤 1 在“**对象浏览器**”窗格中，右键单击连接名称，然后选择“**编辑连接**”。

若要编辑活跃连接，需要先关闭该连接，然后重新打开设置了新属性的连接。工具会显示连接重置警告。

弹出“**编辑连接**”对话框。

步骤 2 单击“**确定**”继续，或单击“**取消**”退出操作。

说明

“名称”字段无法修改。

步骤 3 编辑连接参数。有关参数详情，请参见 4.8.2 添加连接。

步骤 4 单击“确定”保存更新后的连接信息。

说明

- 可单击“清除”清除“编辑连接”对话框中的所有字段。
- 如果未修改任何连接参数就单击“确定”，会显示对话框，提示用户未保存更改。在修改连接参数后，会显示对话框消息。
- 在密码失效的情况下用户可登录到 Data Studio，但系统会提示某些操作可能无法正常工作。请参阅密码过期获取详细信息。
- 关于如何取消连接，请参阅“取消连接”。

如果未启用 SSL，工具会显示“**连接安全告警**”对话框。

步骤 5 单击“**继续**”以继续使用不安全的连接，或单击“**取消**”返回到“**编辑连接**”对话框并启用 SSL。

说明

如果勾选“不再显示”字段，当前登录的 Data Studio 实例在后续连接时，不再显示“连接安全告警”对话框。

工具会显示“**确认删除连接**”对话框，询问用户是否确认删除已编辑连接的数据库。

步骤 6 单击“是”继续更新连接信息，并重新连接到已更新参数的连接。

状态栏显示已完成操作的状态。

----结束

4.8.5 删除连接

执行以下步骤删除数据库连接：

步骤 1 右键单击连接名称，选择“**删除连接**”删除该连接。

Data Studio 弹出确认对话框。

步骤 2 单击“**是**”删除服务器连接。

状态栏显示已完成操作的状态。

此操作将从“**对象浏览器**”中删除与当前服务器的连接。任何未保存的数据将会丢失。

----结束

4.8.6 查看连接属性

执行以下步骤查看连接的属性：

步骤 1 右键单击连接，选择“属性”。

状态栏显示已完成操作的状态。

Data Studio 显示所选连接的属性。

说明

如果为已创建的连接修改了属性，则需要再次打开连接方可查看更新后的属性。

----结束

4.8.7 刷新数据库连接

执行以下步骤刷新数据库连接：

步骤 1 在“**对象浏览器**”窗格中，右键单击连接名称并选择“**刷新**”，或按”F5“刷新数据库。

状态栏显示已完成操作的状态。

----结束

完成刷新数据库所需时间完全取决于数据库中存在的对象数量。因此，建议在大规模数据库中根据需要执行此操作。

- 右键单击连接名称，选择“**刷新**”，刷新整个连接。在刷新的过程中，整个连接将更新为服务器上的最新内容。
- 右键单击函数/过程，选择“**刷新**”，该模式下的所有函数/过程及表将被刷新。在刷新的过程中，所有函数/过程及表将更新为服务器上的最新内容。

  如果存储过程在刷新前已经从数据库中删除，该过程仅当执行刷新操作时从“**对象浏览器**”中删除。
- 右键单击特定函数/过程，选择“**刷新**”，刷新该函数/过程。刷新过程中，特定函数/过程将更新为服务器上的最新内容。
- 如果刷新整个数据库或连接，search_path 中模式下的所有子对象和用户已展开的模式会重新加载。
- 如果重新连接数据库，仅会加载 search_path 中保存的模式对象。先前展开的对象不会加载。
- 不能同时刷新数据库及其下的多个对象。

导出/导入连接详细信息

Data Studio 可以使您从连接对话框中导出/导入连接详细信息，以供将来参考。

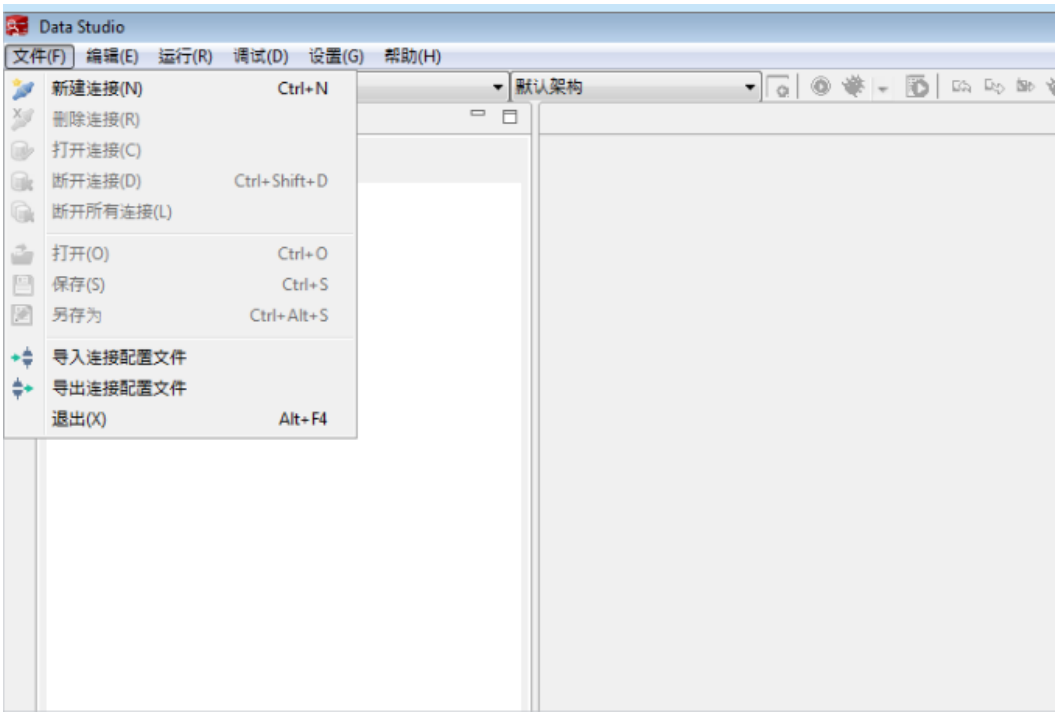
可导出如下字段：

- SSL 模式
- 连接名称
- 服务器 IP
- 服务器端口
- 数据库名称
- 用户名
- clSSLCertificatePath
- clSSLKeyPath
- profileId
- rootCertFilePathText
- connctionDriverName
- schemaExclusionList
- schemaInclusionList
- loadLimit
- privilegeBasedObAcess
- databaseVersion
- savePrdOption
- dbType
- version

执行如下步骤导入/导出连接配置文件：

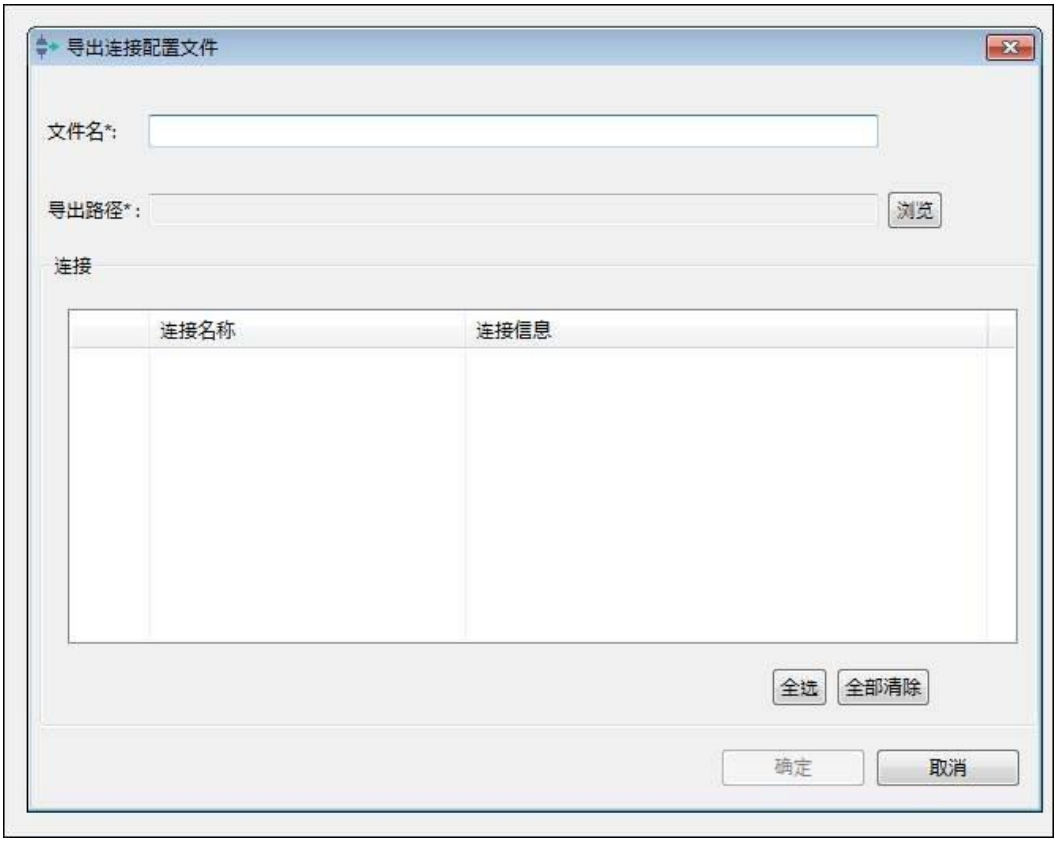
步骤 1 单击 Data Studio 菜单栏上的“**文件**”。

显示如下窗口：



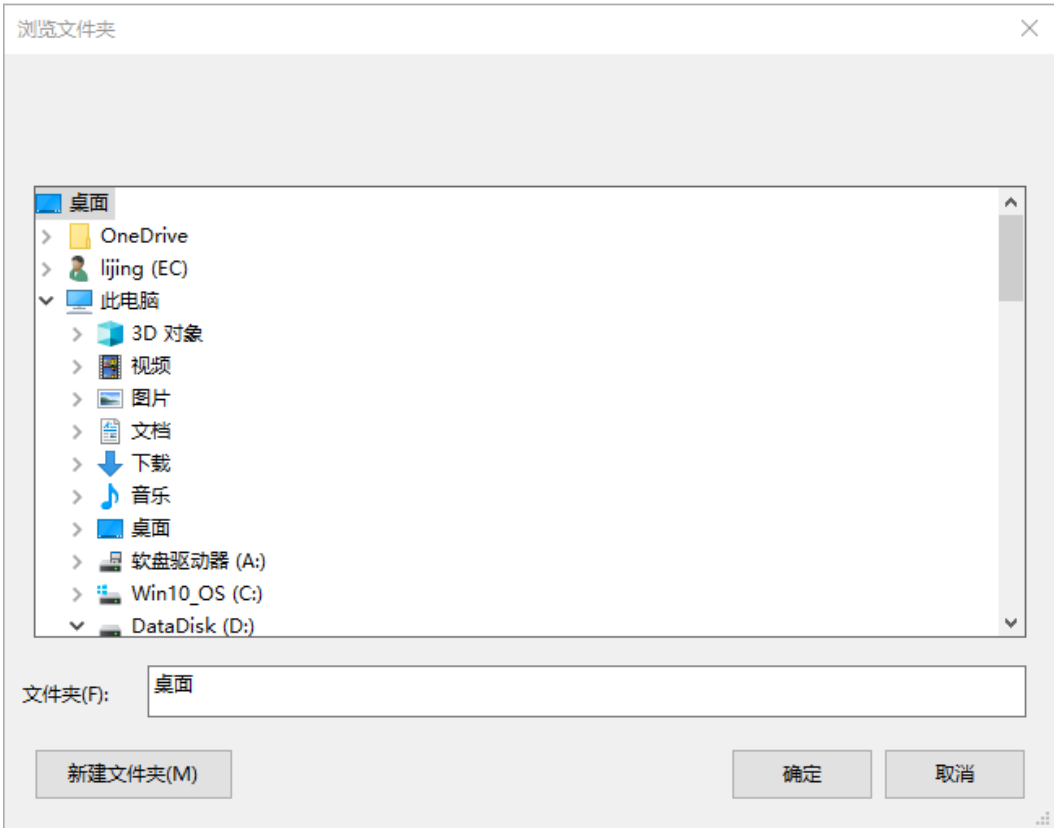
步骤 2 选择“**导出连接配置文件**”导出配置文件。

显示“**导出连接配置文件**”窗口。用户可在该窗口选择需要导出的连接。

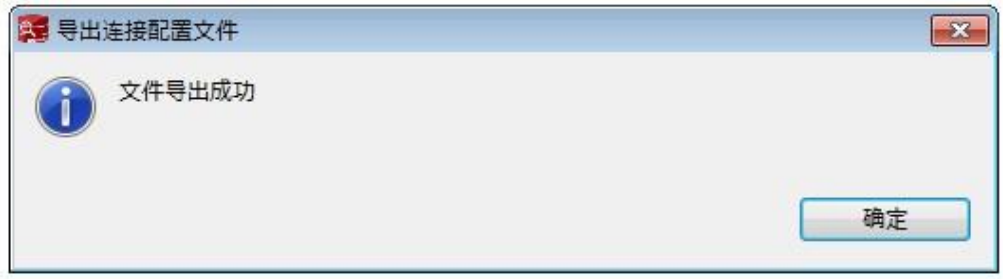


选择要导出的连接，然后输入将保存导出的连接的文件名，单击“**确定**”。

选择文件的保存路径，单击“**确定**”。

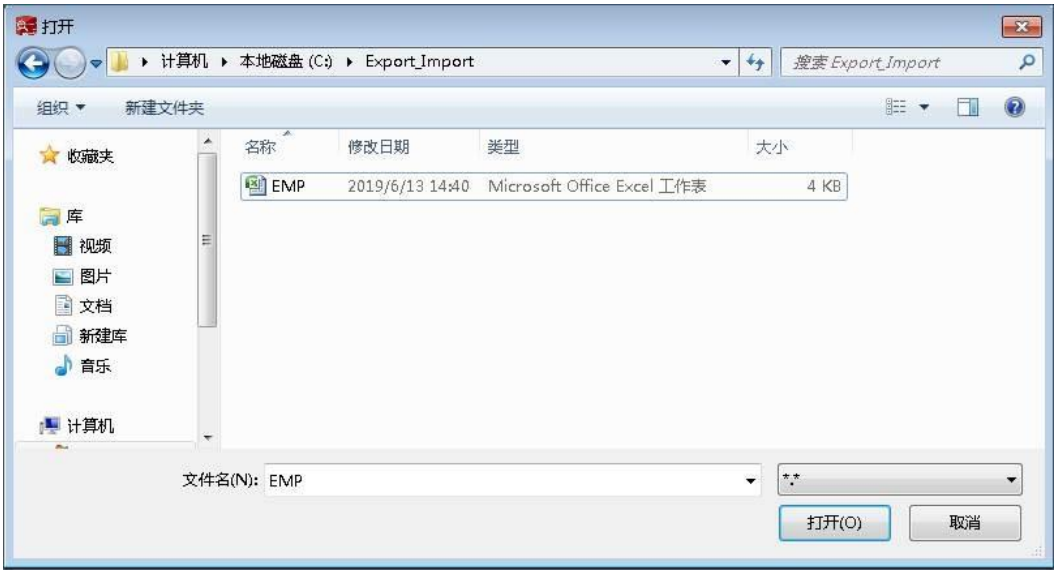


连接导出成功后，会弹出如下对话框。



步骤 3 选择“**导入连接配置文件**”导入连接配置文件。

步骤 4 选择需要导入的文件，单击“**打开**”。



如果要导入的连接与现有连接之间存在匹配，则会弹出如下对话框：

- 替换：导入的连接配置文件将替换为现有配置文件。
- 复制，但保留两个文件：导入的连接配置文件将被重命名。
- 不复制：现有的连接配置文件将保持不变。
- 所有冲突执行该操作：所有匹配都将重复此操作。

按要求选择需要的选项，单击“**确定**”。

----结束

说明

“密码”和“SSL 密码”字段将不会被导出。

4.9 数据库管理

4.9.1 创建数据库

关系型数据库包含一组表，这些表可以依照数据关系模型来操作。关系型数据库包含一组数据对象，用于存储、管理和访问数据对象，包括表、视图、索引、函数等。

执行以下步骤创建数据库：

步骤 1 在“**对象浏览器**”窗格中右键单击“**数据库**”组，选择“**创建数据库**”。弹出“**创建数据库**”对话框，提示您提供创建数据库所需信息。

说明

只有当至少存在一个已连接的数据库时才能执行此操作。

步骤 2 输入数据库名称。命名规则的有关详情，请参见服务器手册。

步骤 3 从“**数据库编码**”中选择所需编码字符集类型。

数据库还支持 UTF-8、GBK、SQL_ASCII 和 LATIN1 编码字符集。使用其他编码字符集创建数据库可能导致操作出错。

步骤 4 勾选“**连接到该数据库**”，然后单击“**确定**”。

状态栏显示已完成操作的状态。

“**对象浏览器**”显示所创建的数据库。服务器上系统相关模式自动添加到新的数据库。

说明

密码失效的情况下您也可登录到 Data Studio，但系统会提示您某些操作可能无法正常工作。请参阅密码过期获取详细信息。

----结束

取消连接

执行以下步骤取消连接：

步骤 1 双击状态栏打开“**进度视图**”标签。

步骤 2 在“**进度视图**”标签内，单击 。

步骤 3 在“**取消操作**”对话框中，单击“**是**”。

状态栏显示被取消的操作状态。

----结束

4.9.2 断开所有连接

通过断开连接功能可以断开某一连接下的所有数据库。

执行如下步骤断连某一连接下的所有数据库：

步骤 1 在“**对象浏览器**”窗格中，右键单击数据库组，选择“**断开所有连接**”。该操作将断开某一连接下所有数据库。

说明

该操作仅能在已连接的数据库上执行。

Data Studio 弹出确认对话框，确认是否断连该连接下所有数据库。

步骤 2 点击“**是**”断开连接。

状态栏显示已完成操作的状态。

Data Studio 自动填充最近一次与服务器成功建立连接的所有连接参数设置（“密码”除外）。因此，再次创建连接时，仅需要在连接向导中输入密码。

----结束

4.9.3 连接到数据库

本节介绍如何连接到数据库。

按照以下步骤连接数据库：

步骤 1 在“**对象浏览器**”窗格，右键数据库名称，然后选择“**连接到数据库**”。

说明

该操作仅能在断连的数据库上执行。

连接到数据库。

状态栏显示已完成操作的状态。

说明

- 密码失效的情况下您也可登录到 Data Studio，但系统会提示您某些操作可能无法正常工作。请参阅密码过期获取详细信息。
- 如何取消连接，请参阅取消连接获取详细信息。

----结束

4.9.4 断开连接

本节介绍如何断开数据库连接。

按照如下步骤断开数据库连接。

步骤 1 在“**对象浏览器**”窗格，右键单击数据库名称，选择“**断开连接**”。

说明

该操作只能在主库执行。

Data Studio 弹出确认对话框。

步骤 2 单击“**是**”断开连接。

数据库连接断开。

状态栏显示已完成操作的状态。

----结束

4.9.5 重命名数据库

执行如下步骤重命名数据库。

步骤 1 在“**对象浏览器**”窗格中右键单击数据库名称，在菜单中选择“**重命名**”。

说明

该操作仅能在断连的数据库上执行。

弹出“**重命名数据库**”对话框，提示用户提供重命名数据库所需信息。

步骤 2 输入新数据库名称。勾选“**连接到该数据库**”，然后单击“**确定**”。

Data Studio 提示确认该操作。

步骤 3 在确认对话框中，单击“**确定**”重命名数据库。

状态栏显示已完成操作的状态。

“**对象浏览器**”显示重命名后的数据库。

说明

如何取消连接，请参阅取消连接获取详细信息。

----结束

4.9.6 删除数据库

可单独或批量删除数据库。要进行批量删除，详情请参见 4.21.2 批量删除对象。

执行以下步骤删除数据库：

步骤 1 在“**对象浏览器**”窗格中，右键单击数据库并选择“**删除数据库**”。

说明

该操作仅能在断连的数据库上执行。

Data Studio 弹出确认窗口，确认是否删除数据库。

步骤 2 单击“**确定**”删除该数据库。

弹出消息和状态栏显示已完成操作的状态。

----结束

4.9.7 查看数据库属性

按照如下步骤查看数据库属性：

步骤 1 右键数据库并选择“**属性**”。

说明

该操作仅能在已连接的数据库上执行。

状态栏显示已完成操作的状态。

Data Studio 显示所选数据库的属性。

说明

如果修改了已经打开的数据库的属性，则可刷新并重新打开数据库的属性，以在同一窗口中查看更新后的信息。

----结束

4.10 模式管理

4.10.1 概述

本节介绍如何使用数据库模式。所有系统模式均在“**系统模式**”下分组，用户模式在“**用户模式**”下。

4.10.2 创建模式

在关系型数据库技术中，模式提供数据库对象的逻辑分类，一个模式可能包含如下数据库对象：函数/过程、表、视图、序列和索引。

执行如下步骤定义模式：

步骤 1 在“**对象浏览器**”窗格中，右键单击“**用户模式**”组，选择“**创建模式**”。

说明

对于“系统模式”组，仅能执行刷新操作。

步骤 2 输入模式名称，单击“**确定**”。仅在数据库连接为 active 时，可创建模式。

状态栏显示已完成操作的状态。

“**对象浏览器**”窗格中显示所创建的模式。

----结束

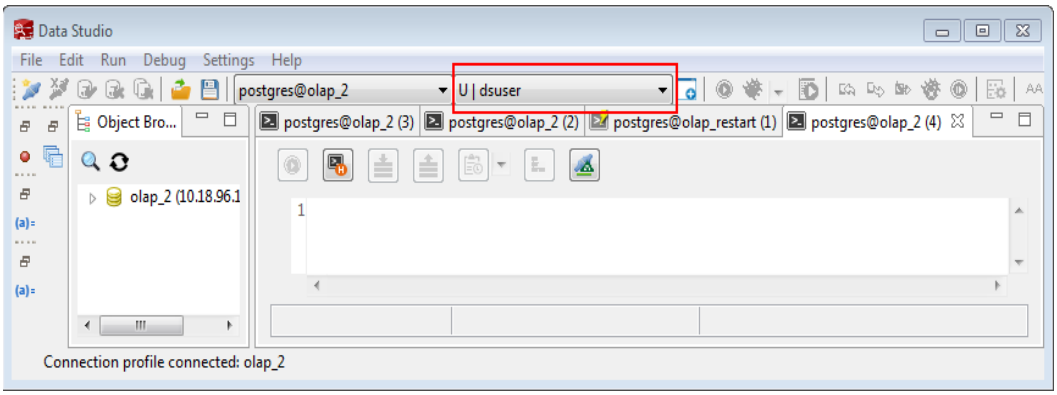
可对模式执行如下操作：

- 重命名模式（有关详情，请参见 4.10.5 重命名模式）
- 删除模式（有关详情，请参见 4.10.8 删除模式）

- 导出 DDL（有关详情，请参见 4.10.3 导出模式 DDL）
- 导出 DDL 和数据（有关详情，请参见 4.10.4 导出模式 DDL 和数据）
- 授权/撤销权限（有关详情，请参见 4.10.7 授权/撤销权限）
- 刷新模式：右键单击模式名称，选择“**刷新**”，将刷新该模式下所有对象。

显示默认模式

Data Studio 可以在工具栏中显示用户的默认模式。



当从 SQL 终端执行未提及模式名称的 create 查询时，将在用户的默认模式下创建相应的对象。

当在 SQL 终端中执行 select 查询而不提及模式名称时，将搜索默认模式以查找这些对象。

Data Studio 启动时，默认模式会被设置为<username>，公共模式具有相同的优先级。

如果在下拉列表中选择了另一个模式，则此模式被设置为默认模式，并覆盖之前的设置。

所选模式设置为数据库（从数据库下拉列表中选择）中所有活动连接的默认模式。

说明

此功能不适用于 OLTP 数据库。

4.10.3 导出模式 DDL

可通过导出 DDL 导出该模式下函数/过程、表、序列和视图的 DDL。

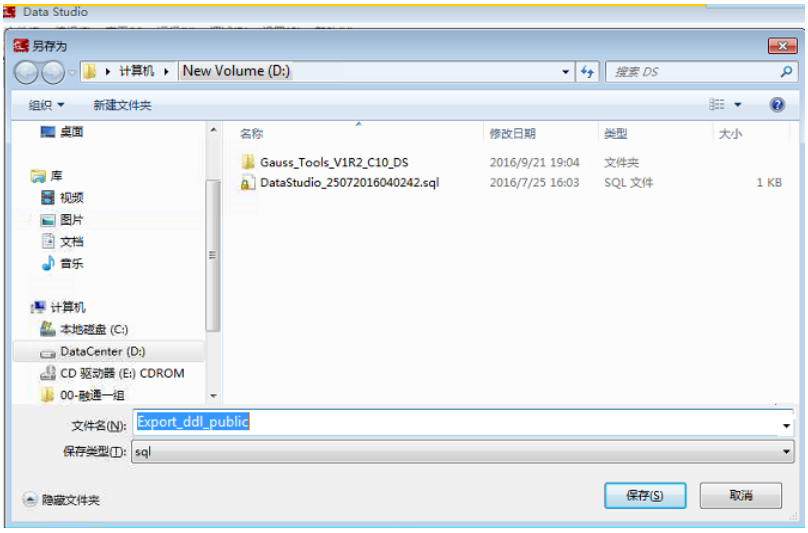
执行以下步骤导出模式 DDL：

步骤 1 在“**对象浏览器**”窗格中，右键单击所选模式，选择“**导出 DDL**”。

Data Studio 显示“**Data Studio 安全免责声明**”对话框。用户可关闭此对话框。详情请参见安全免责声明。

步骤 2 单击“**确定**”。

Data Studio 显示“**另存为**”对话框。



步骤 3 在“**另存为**”对话框中，选择定义的保存位置，单击“**保存**”。状态栏会显示操作进度。

说明

- 要终止导出操作，双击状态栏，打开“进度视图”页签，单击。有关详情，请参见取消导出表数据操作。
- 如果文件名包含 Windows 中文件名不支持的字符，则文件名的名称会与模式名称不同。
- 要执行该操作，需要 Microsoft Visual C Runtime 文件（msvcrt100.dll）。详情请参阅 4.25 故障处理。

“**导出完成**”对话框和状态栏显示已完成操作的状态。

数据库编码	文件编码	支持导出 DDL
UTF-8	UTF-8	是
	GBK	是
	LATIN1	是
GBK	GBK	是
	UTF-8	是
	LATIN1	否
LATIN1	LATIN1	是
	GBK	否
	UTF-8	是

说明

可选择并导出多个对象的 DDL。批量导出章节列举了不支持导出 DDL 的对象。

----结束

4.10.4 导出模式 DDL 和数据

通过导出模式的 DDL 和数据，可导出该模式下的如下内容：

- 函数/过程的 DDL
- 表的 DDL 和数据
- 视图的 DDL
- 序列的 DDL

执行以下步骤导出模式的 DDL 和数据：

步骤 1 在“**对象浏览器**”窗格中，右键单击所选模式，选择“**导出 DDL 和数据**”。

Data Studio 显示“**Data Studio 安全免责声明**”对话框。

用户可关闭此对话框。详情请参见安全免责声明。

步骤 2 单击“**确定**”。

Data Studio 显示“**另存为**”对话框。

步骤 3 在“**另存为**”对话框中，选择定义和数据的保存位置，单击“**保存**”。状态栏会显示操作进度。

说明

- 要终止导出操作，双击状态栏，打开“进度视图”页签，单击。有关详情，请参见取消导出表数据操作。
- 如果文件名包含 Windows 中文件名不支持的字符，则文件名的名称会与模式名称不同。
- 要执行该操作，需要 Microsoft Visual C Runtime 文件（msvcrt100.dll）。详情请参阅 4.25 故障处理。

“**导出完成**”对话框和状态栏显示已完成操作的状态。

数据库编码	文件编码	支持导出 DDL
UTF-8	UTF-8	是
	GBK	是
	LATIN1	是
GBK	GBK	是
	UTF-8	是
	LATIN1	否
LATIN1	LATIN1	是
	GBK	否

数据库编码	文件编码	支持导出 DDL
	UTF-8	是

说明

可选择并导出多个对象的 DDL 和数据。批量导出章节列举了不支持导出 DDL 和数据的对象。

----结束

4.10.5 重命名模式

执行如下步骤重命名模式：

步骤 1 在“**对象浏览器**”窗格中，右键单击模式并选择“**重命名**”。

步骤 2 输入模式名称，单击“**确定**”。

“**对象浏览器**”窗格显示重命名后的模式。

状态栏显示已完成操作的状态。

----结束

4.10.6 支持序列 DDL

Data Studio 支持用户进行显示序列 DDL 和导出序列 DDL 操作，包括“**显示 DDL**”，“**导出 DDL**”，“**导出 DDL 和数据**”。

执行如下步骤进行相关操作：

步骤 1 在“**对象浏览器**”窗格中，右键单击“**序列**”下的任一对象。

显示菜单选项。

步骤 2 选择“**显示 DDL**”选项查看 DDL 语句。

选择“**导出 DDL**”选项导出 DDL 语句。

选择“**导出 DDL 和数据**”选项导出 DDL 和 select 语句。

如下图所示：

说明

只有序列所有者或系统管理员，或具有序列的 select 权限的用户，才能进行上述操作。

----结束

4.10.7 授权/撤销权限

执行以下步骤授权/撤销权限：

步骤 1 右键单击模式组并选择“**授权/撤销权限**”。

弹出“**授权/撤销权限**”对话框。

步骤 2 打开“**选择对象**”页签，选择待授权/撤销权限的对象，并单击“**下一步**”。

步骤 3 打开“**选择权限**”页签，从“**角色**”中选择对应角色。

步骤 4 在“**选择权限**”页签，勾选“**授予**”或“**撤销**”。

步骤 5 在“**选择权限**”页签，勾选或取消勾选相关权限。

在“**SQL 预览**”页签，可以查看根据以上输入自动生成的 SQL 查询。

步骤 6 单击“**完成**”。

----结束

4.10.8 删除模式

可单独或批量删除模式。要进行批量删除，详情请参见 4.21.2 批量删除对象。

执行如下步骤删除模式：

步骤 1 在“**对象浏览器**”窗格中右键单击模式，选择“**删除模式**”。Data Studio 弹出确认窗口。

步骤 2 单击“**确定**”。该模式从“**对象浏览器**”中删除。

弹出消息和状态栏显示已完成操作的状态。

----结束

4.11 创建函数/过程

执行如下步骤创建函数、过程和 SQL 函数：

步骤 1 在“**对象浏览器**”窗格中，右键单击待创建 PL/SQL 过程的指定模式下的“**函数/过程**”，按照要求选择“**创建 PL/SQL 函数**”、“**创建 SQL 函数**”、“**创建 PL/SQL 过程**”或“**创建 C 函数**”。

Data Studio 在新页签中显示所选模板。

步骤 2 添加函数/过程，右键单击页签，选择“**编译**”，或选择“**运行 > 编译/执行声明**”；或按“Ctrl+Enter”编译该过程。

Data Studio 弹出“**创建函数/存储过程成功**”对话框，并在“**对象浏览器**”中显示新函数/过程。单击“**确定**”关闭“**NewObject()**”页签，并将调试对象添加到“**对象浏览器**”。

如果在执行期间丢失连接，请参阅执行 SQL 查询获取有关重新连接选项的具体信息。

步骤 3 过程名称旁的星号（*），表示该过程不被编译或添加到“对象浏览器”。

刷新（按下“F5”）后，“对象浏览器”中会显示新添加的调试对象。

说明

- C 函数不支持调试操作。
- 弹出消息显示已完成操作的状态。状态栏将不显示此操作的状态。

----结束

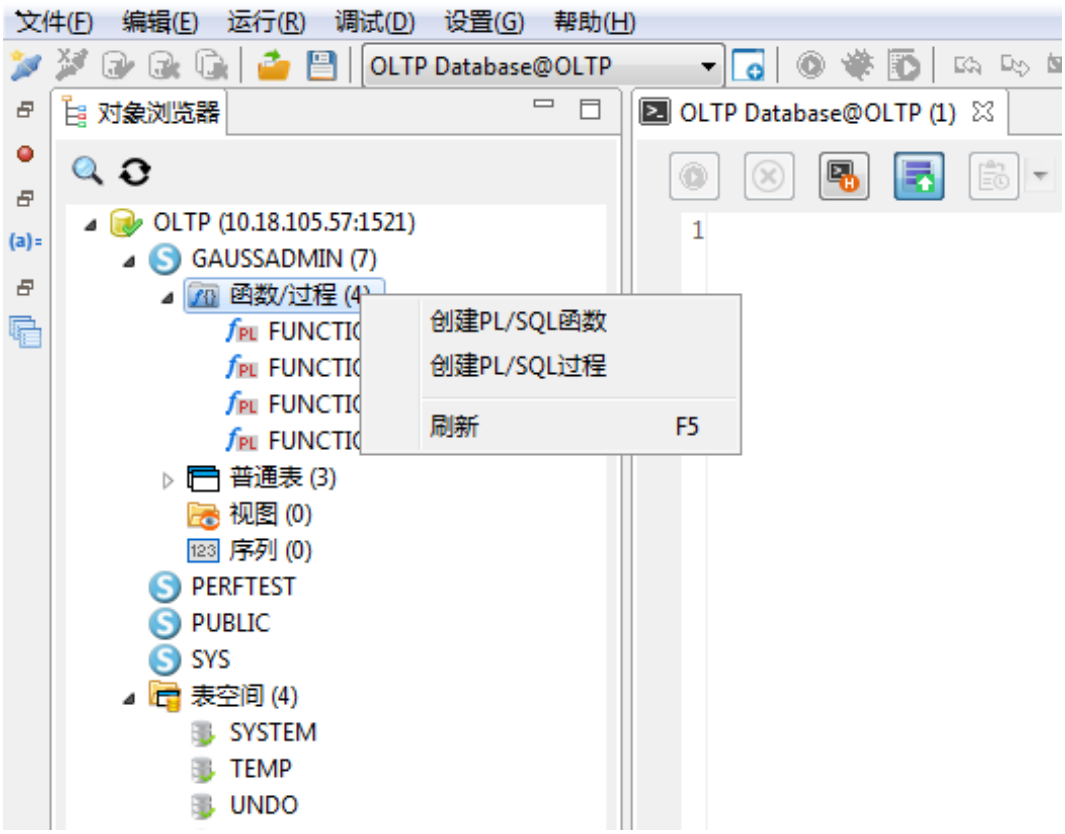
函数编译支持

当用户从模板获取或通过编辑已有对象生成一个新的 PL/SQL 对象时，系统打开新的页签显示该对象。

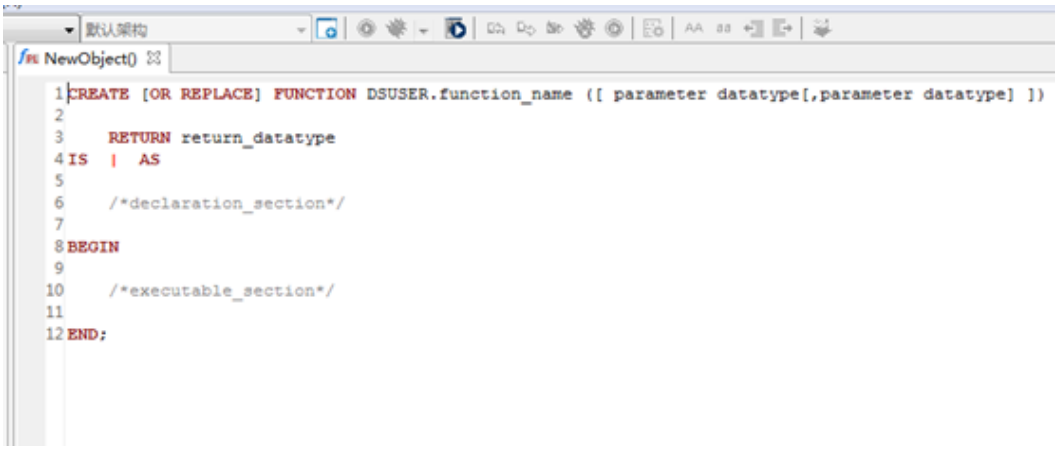
执行以下步骤为创建函数提供编译支持：

步骤 1 在对象浏览器中选中“**函数/过程**”。

步骤 2 右键单击“**函数/过程**”，弹出如下菜单：



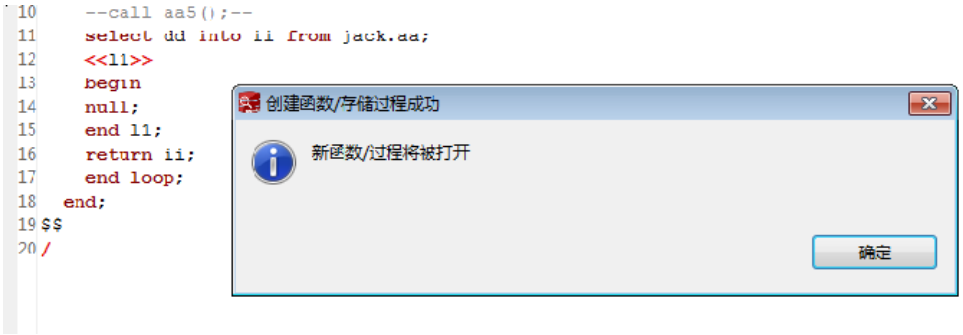
步骤 3　选择“**创建 PL/SQL 函数**”。系统打开新的页签。



步骤 4　编辑代码。

步骤 5　在页签空白处右键单击，弹出如下菜单：

步骤 6 选择“**编译**”。弹出如下提示信息：



系统打开新页签，显示该函数。

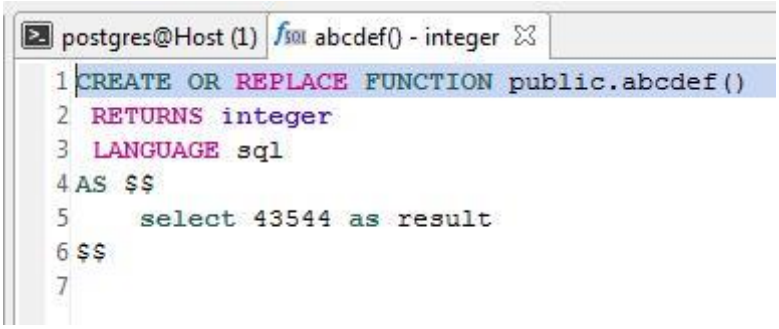
----结束

4.12 编辑函数/过程

执行以下步骤编辑函数/过程或 SQL 函数：

步骤 1 在“**对象浏览器**”中双击所需过程/函数或 SQL 函数，或右键单击函数/过程或 SQL 函数并选择“**查看源**”。用户必须刷新“**对象浏览器**”才能查看最新 DDL。

“PL/SQL Viewer”页签显示所选函数/过程或 SQL 函数。



拥有相同模式、名称和输入参数的函数/过程或 SQL 函数一次只能打开一个。

步骤 2 编辑或更新后，您可以编译并执行该 PL/SQL 程序或 SQL 函数。有关详情，请参见 4.15.6 执行函数/过程。

在编译前若执行函数/过程或 SQL 函数，会显示一条“**改变源代码**”的提示。

步骤 3 单击“**是**”，编译并执行 PL/SQL 函数/过程。

“**消息**”对话框显示已完成操作的状态。

如果在执行期间丢失连接，请参阅执行 SQL 查询获取有关重新连接选项的具体信息。

步骤 4 编译函数/过程或 SQL 函数后，刷新“**对象浏览器**”（按下“F5”）查看更新后信息。

----结束

4.13 授权/撤销权限（函数/过程）

执行以下步骤授权/撤销权限：

步骤 1 右键单击函数/过程组并选择“**授权/撤销权限**”。

弹出“**授权/撤销权限**”对话框。

步骤 2 打开“**选择对象**”页签，选择待授权/撤销权限的对象，并单击“**下一步**”。

弹出“**选择权限**”页签。

步骤 3 从“**角色**”中选择对应角色。

步骤 4 勾选“**授予**”或“**撤销**”。

步骤 5 勾选或取消勾选相关权限。

“**SQL 预览**”页签显示根据以上输入自动生成的 SQL 查询。

步骤 6 单击“**完成**”。

----结束

说明

本特性仅在 OLAP 支持，OLTP 中不支持。

4.14 调试 PL/SQL 函数

4.14.1 调试 PL/SQL 函数概述

调试操作期间，如果连接丢失，但对象浏览器中仍存在该数据库连接，则“**连接错误**”对话框中会存在如下选项：

- “**是**”：重建连接并重启调试操作。
- “**否**”：断开对象浏览器中的数据库连接。

说明

SQL 语言函数不支持调试操作。

4.14.2 使用断点

本节包含如下内容：

- 使用“断点”窗格
- 设置或添加断点
- 启用或禁用断点
- 删除断点
- 修改源代码
- 使用断点测试 PL/SQL 程序

断点用于暂停其所在行中的 PL/SQL 程序的执行，可用断点控制执行并调试过程。

- 设置并启用断点后，PL/SQL 程序会在该断点所在的行停止执行，此时用户可以进行其他调试操作。Data Studio 支持以下断点操作：
  - 为某行设置或创建断点。
  - 启用或禁用某行的断点。
  - 删除某行的断点。
- 禁用断点后，PL/SQL 程序不会在断点处暂停执行。

运行 PL/SQL 程序时，程序会在设置断点的每一行暂停执行。此时，Data Studio 中检索当前程序状态信息，如程序变量的值。

执行如下步骤调试 PL/SQL 程序：

步骤 1 在需要 PL/SQL 程序暂停执行的行设置断点。

步骤 2 启动调试会话。

到达断点所在行时，监视调试窗格中应用程序的状态，然后继续执行程序。

步骤 3 关闭调试会话。

----结束

Data Studio 的工具栏中有调试选项，可以帮助您执行可调试对象。

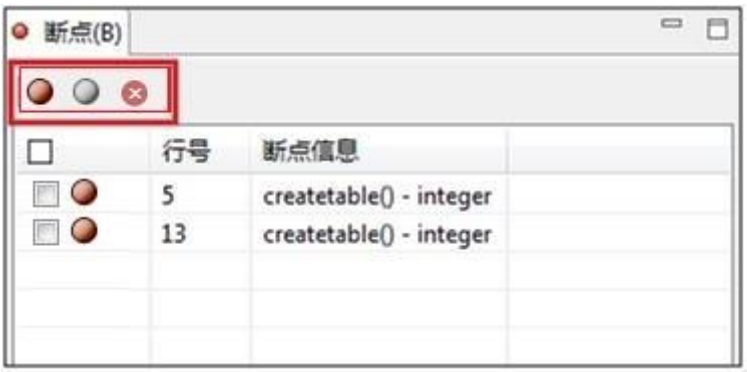
使用“断点”窗格

在“**断点**”窗格中可查看和管理已有断点。可从最小化窗口窗格中单击断点按钮打开“**断点**”窗格。

“**断点**”窗格会列出每一个断点所在行的行号以及调试对象的名称。

在“**断点**”窗格中单击☐，可启用或者禁用所有断点。在“**断点**”窗格中，勾选断点复选框，单击 、 或 ，进行断点启用、禁用或者删除操作。

在“PL/SQL Viewer”窗格，双击所需断点信息行，定位该断点。



说明

- 禁用断点后，程序不会在该断点处暂停执行，但该断点仍会保留（以备将来启用）。
- 删除的断点无法恢复。
- 按下“Alt+Y”可复制“断点”窗格内容。

设置或添加断点

执行以下步骤在某行中添加断点：

步骤 1 打开需要添加断点的 PL/SQL 函数。

步骤 2 在“PL/SQL Viewer”窗格中，双击行号字段左侧的断点标尺，设置断点，启用断点标志 [ ] 表示操作成功。

说明

如果函数在调试过程中不会间断或停止执行，则为其设置的断点不会生效。

----结束

启用或禁用断点

设置断点后，可在“**断点**”窗格顶部勾选其对应的复选框，单击“**断点**”窗格顶部的  ，暂时禁用该断点。“PL/SQL Viewer”和“**断点**”窗格中，禁用的断点会灰化显示[]。若要启用已禁用的断点，勾选其对应的复选框，单击  。

删除断点

用户可删除不再使用的断点。其方法与断点创建的方法相同。

“PL/SQL Viewer”页签中，打开待删除断点所在的函数，双击断点启用图标[]，将断点删除。

也可以利用上述方法，在“**断点**”页签进行断点启用或禁用操作。

修改源代码

调试过程中，如果用户修改了从服务器获取的源代码，并继续进行调试，Data Studio会提示错误：

建议刷新该对象，再次执行调试操作。

说明

如果用户修改了从服务器获取的源代码，且在未设置断点的情况下执行或调试了该代码，Data Studio 会显示服务器中源代码的执行结果。建议在进行调试或者执行前，进行刷新操作。

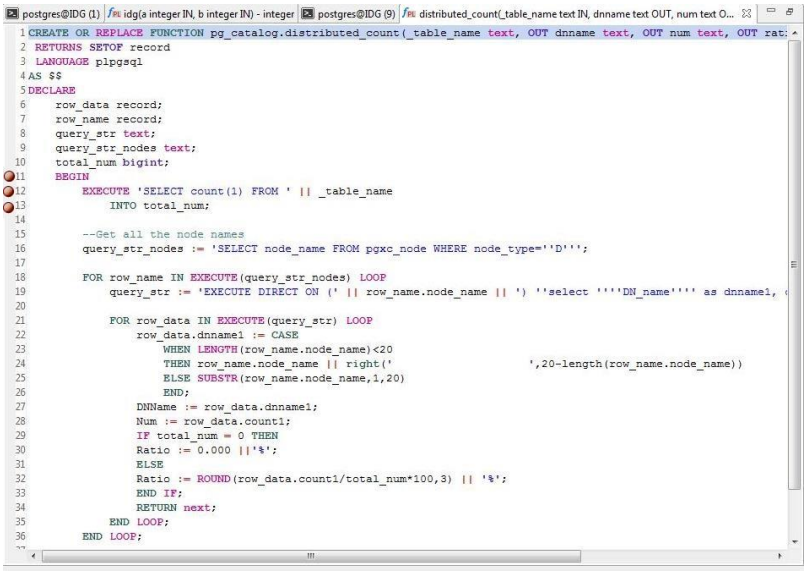
使用断点测试 PL/SQL 程序

执行以下步骤使用断点测试 PL/SQL 程序：

步骤 1 打开 PL/SQL 程序，在要调试的行创建断点。

例如：

11、12 和 13 行



步骤 2 单击，按下“Ctrl+D”，或在“**对象浏览器**”中右键单击 PL/SQL 程序，选择“**调试**”。在弹出的“**调试函数/过程**”对话框中，输入参数信息。

说明

如果没有输入参数，则“调试函数/过程”对话框不会弹出。

步骤 3 输入信息，单击“**确定**”。对于 varchar 和 date 数据类型，参数值需加半角单引号（'）；对于 numeric 数据类型，参数值可以不加半角单引号。

要将参数值设为 NULL，需输入 NULL 或 null。

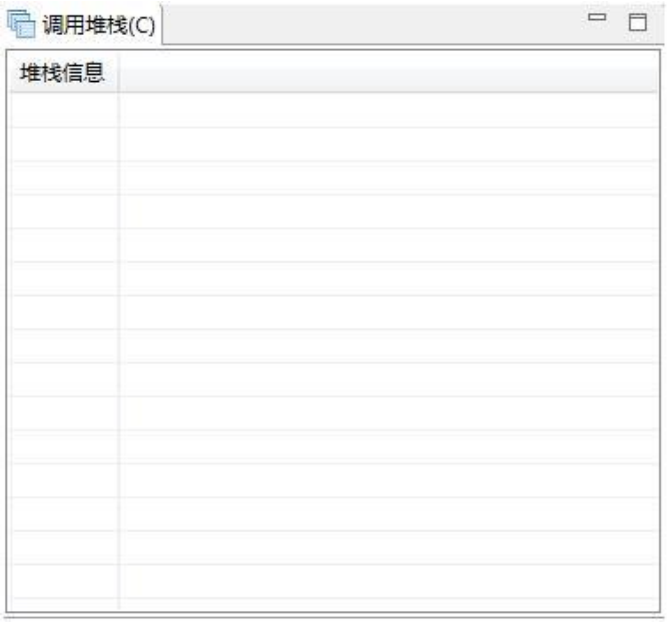
单击“**调试**”，可以看到箭头指向断点所在行。箭头所指的行号，即为继续执行程序时的起始行号。

postgres@Host (1) | func_long_src_code() - integer

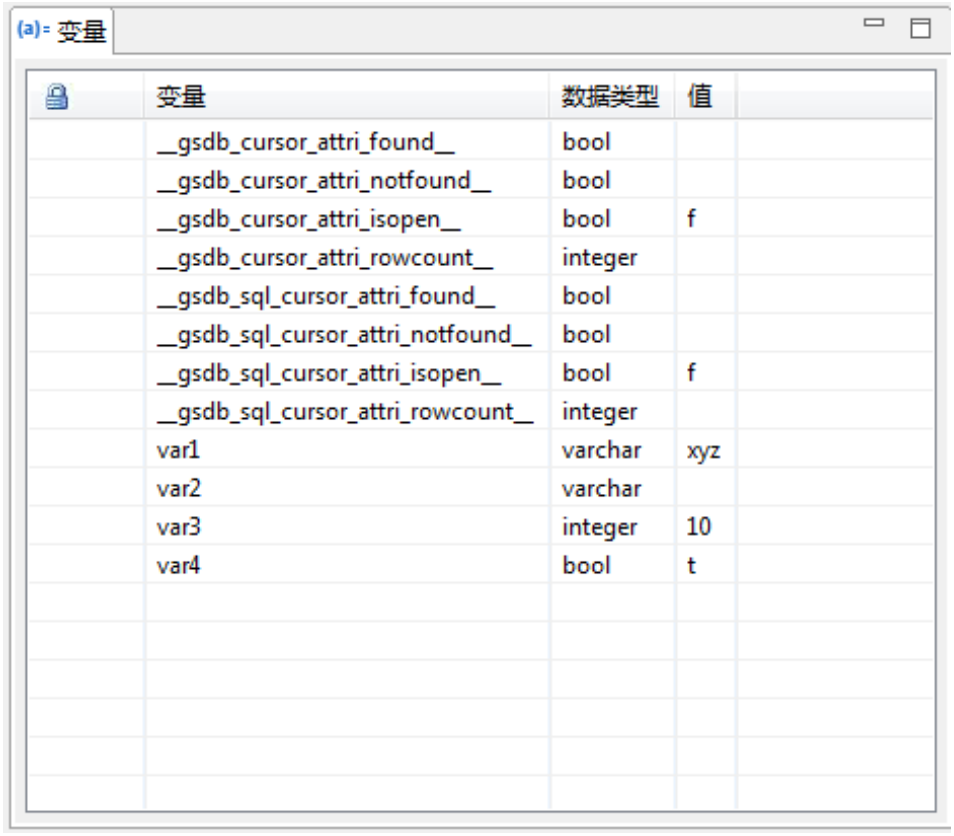
```
1 CREATE OR REPLACE FUNCTION public.func_long_src_code()
2  RETURNS integer
3  LANGUAGE plpgsql
4 AS $$ DECLARE  result INTEGER;
5 ab varchar(27);
6 bc varchar(27);
7     a INTEGER;
8
9 BEGIN
10     a := 10;
11 a := a+2;
12  a := a+2;
13  a := a+2;
14  a := a+2;
15  a := a+2;
16     a := a+2;
17  a := a+2;
18  a := a+2;
19  a := a+2;
20 ab:='abcdefghijklmnopqrstuvwxyz';
21 bc:=ab;
22 ab:='abcasddfsdfksdfksdklflkla';
23 a := a+2;
24 a := a+2;
25 a := a+2;
26 a := a+2;
```

可通过三种方式终止调试：在工具栏中单击按钮；按下“F10”；或在“**调试**”菜单中选择“**终端调试**”。调试完成后，函数会继续正常执行，不会在任何断点暂停。

“**调用堆栈**”和“**变量**”窗格会填充信息。



“**变量**”窗格显示当前的变量值。将鼠标悬停在函数/过程中的变量上，也会显示当前的变量值。



用户可单步跳入、单步退出、或单步跳过代码。详细信息请参见 4.14.3 控制执行。

步骤 4 单击继续执行至下一断点（如有）。“**结果**”页签中显示执行 PL/SQL 程序的结果，“**调用堆栈**”和“**变量**”窗格将被清除。可单击“**结果**”页签上的复制页签内容。

可通过以下方式删除断点：

- 再次双击断点，将其从 PL/SQL Viewer 中删除。
- 选中断点对应的复选框，在“**断点**”页签中单击。

----结束

改变“变量”窗格位置

该功能支持改变“**变量**”窗格极其列的位置。用户可以将“**变量**”窗格移动至如下位置：

- “**SQL 助手**”页签旁边
- 终端页签旁边
- “**对象浏览器**”页签旁边
- “**结果**”页签旁边
- “**断点**”页签旁边

- “**调用堆栈**”页签旁边
- “**对象浏览器**”页签下方

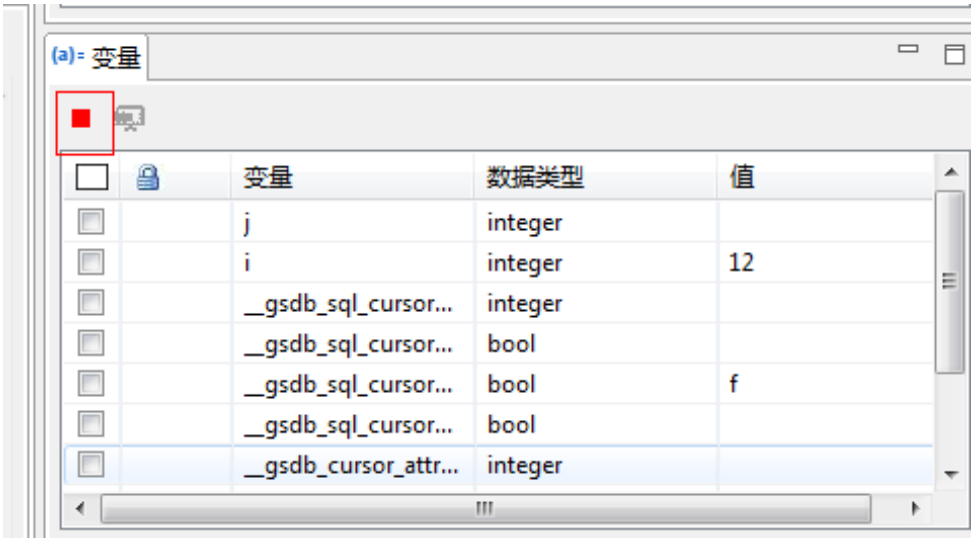
说明

无论“变量”窗格是否在调试过程中被改变位置，调试结束后，窗格都会被最小化。若“变量”窗格被拖动至终端页签或“结果”页签，调试结束后，应手动最小化窗格。窗格位置改变后将保持不变。

启用/屏蔽系统变量

“**变量**”窗格默认显示系统变量。必要时，用户可以屏蔽系统变量。

步骤 1 单击“**变量**”窗格中的按钮，如下图红框中所示，即可屏系统变量。



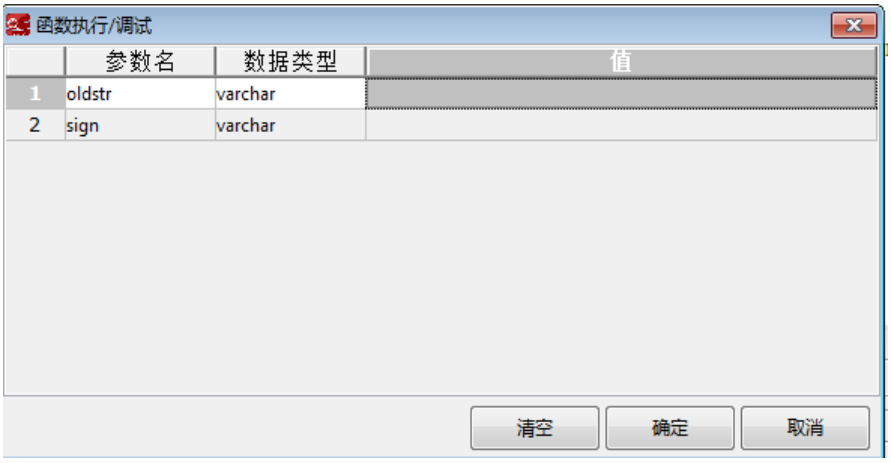
按钮默认开启，即默认启用系统变量。

----结束

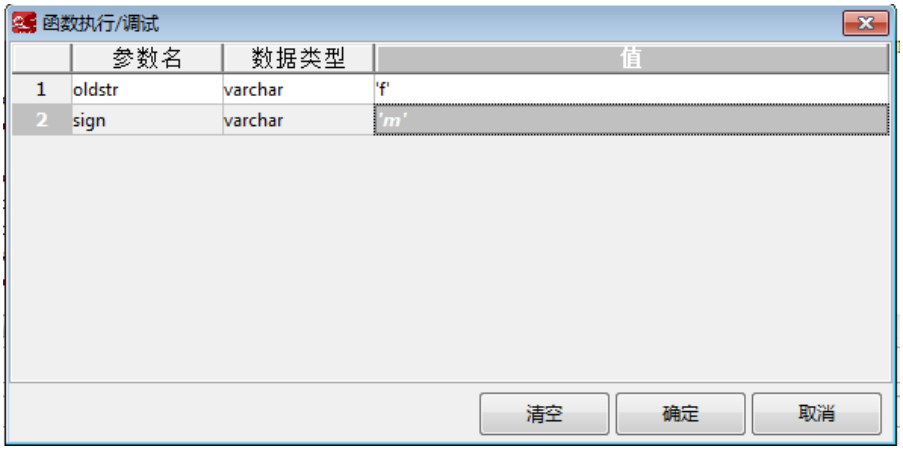
显示缓存参数

调试或执行 PL/SQL 函数或过程期间，相同参数的值会在下次操作中被直接使用。

执行 PL/SQL 对象时，显示如下窗口。



首次执行时，参数值为空，您可根据需要输入参数值。



单击“**确定**”，参数值将被缓存。下次查询时，执行/调试期间会显示上次缓存的相同参数的值。

说明

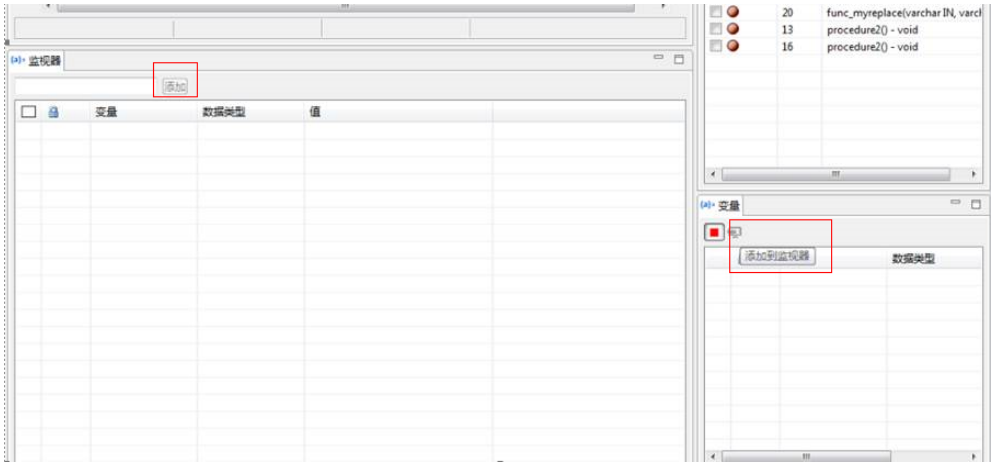
某一连接被删除后，缓存的所有参数值都会被清除。

“监视器”窗格中显示变量

调试期间，Data Studio 会在“**监视器**”窗口显示正在被监控的变量。

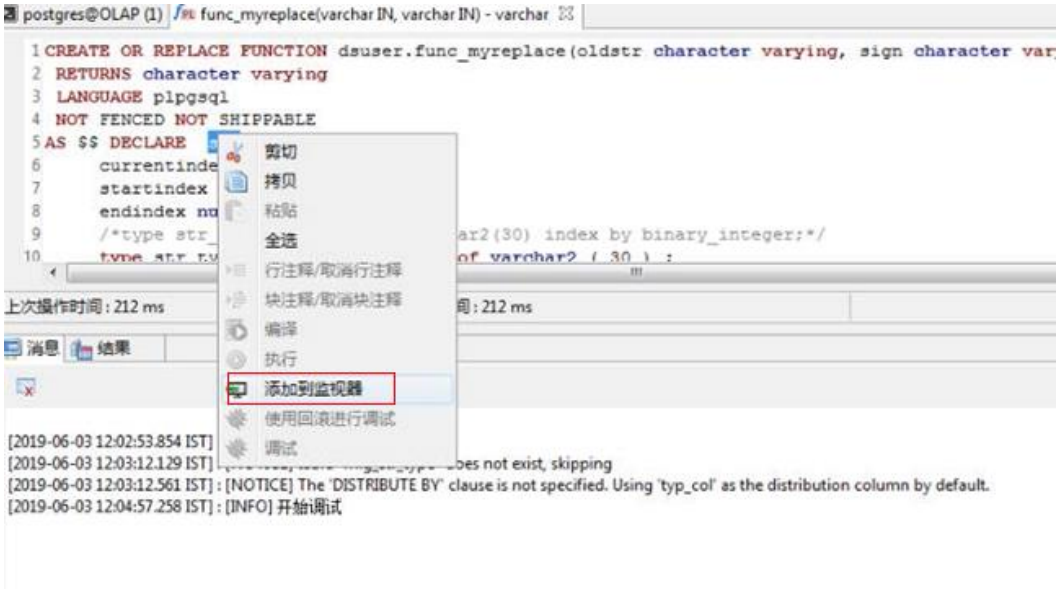
在“**监视器**”窗格中，必须以如下方法添加变量：

- 从“**变量**”窗格中添加所选变量并右键单击。
- 从“**变量**”窗格中选择变量，然后单击“**变量**”窗格工具栏中的按钮进行添加。



如果变量处于被监控状态，“**监视器**”窗格中的值会随“**变量**”窗格中值的变化而变，反之亦然。

- 调试函数/过程期间，在编辑器中右键单击变量，将变量添加至“**监视器**”窗格：



在 Data Studio 窗口中，“**监视器**”窗格可被拖拽至任何位置。

调试期间显示变量的游标信息

在 Data Studio 中调试 PL/SQl 函数期间，将光标放在变量上方，则会显示该变量的信息。

调试中支持回滚/提交

Data Studio 支持在调试完成后，提交/回滚 PL/SQL 查询结果。

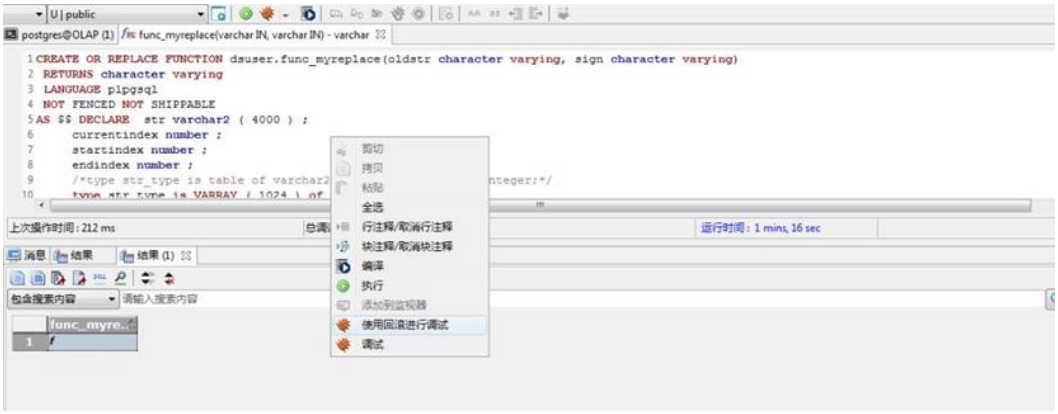
- 如果启用了“**使用回滚进行调试**”选项，则调试后获取的 PL/SQL 执行结果不会保存在数据库中。
- 如果禁用了“**使用回滚进行调试**”选项，则调试后获取的 PL/SQL 执行结果被提交到数据库中。

按如下步骤启用回滚功能：

步骤 1 PL/SQL 调试期间，检查“**使用回滚进行调试**”复选框并启用回滚功能。

或者

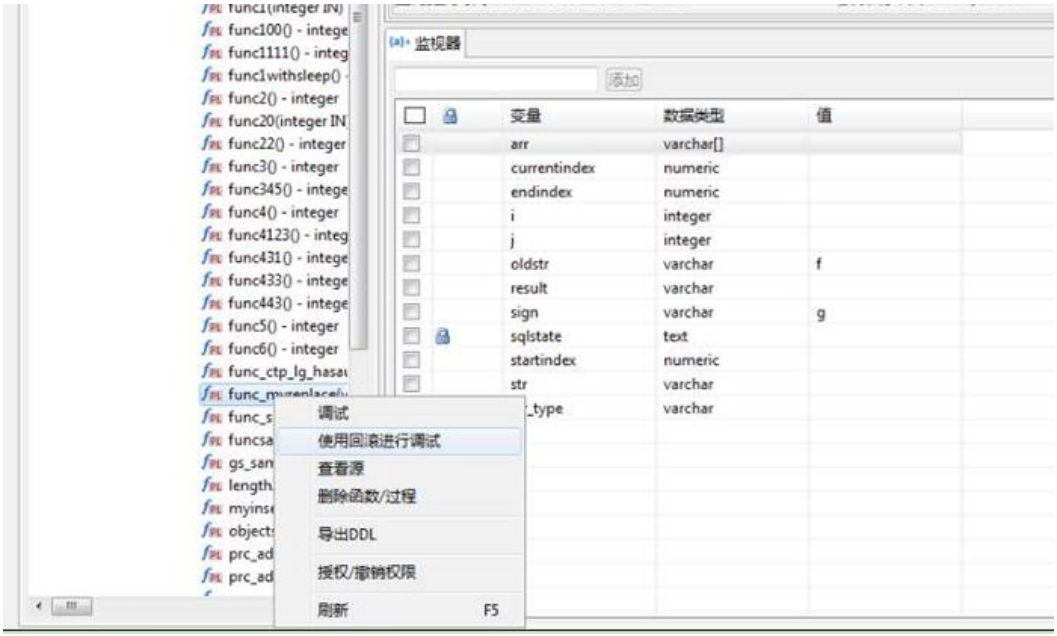
右键单击执行 PL/SQL 函数的 SQL 终端窗格。



调试完成后，勾选“**使用回滚进行调试**”启用回滚功能。

或者

在“**对象浏览器**”窗格的“**函数/过程**”模块下右键单击任一 PL/SQL 函数启用回滚功能。



----结束

4.14.3 控制执行

本节包含如下内容：

- 开始调试
- 单步调试 PL/SQL 函数
- 继续执行
- 查看调用堆栈

开始调试

在“**对象浏览器**”窗格中选择需调试的函数。单击工具栏中的（或使用前文提到的其他方法）开始调试。如果没有设置断点，或者设置的断点无效，则不会停止任何语句、进行调试操作，而仅会执行对象并显示结果（如有）。

单步调试 PL/SQL 函数

调试执行函数时，用户可使用工具栏中的单步调试命令。通过进行单步控制，可逐行调试程序。如果进行单步操作时遇到断点，则该单步操作会停止，程序也会暂停执行。

“单步”是指一次执行一条语句。单步执行一条语句后，可以在其他调试窗口中查看执行结果。

说明

一次最多可以弹出 100 个“PL/SQL Viewer”页签。如果打开的页签超过 100 个，则调用函数页签关闭。例如，如果已打开 100 个页签，且调试对象调用了新调试对象，则 Data Studio 会关闭调用函数并打开新的调试对象。

单步进入

在选择“**调试**”菜单中选择单步进入、按或者按 F7，都可以逐条语句执行代码。

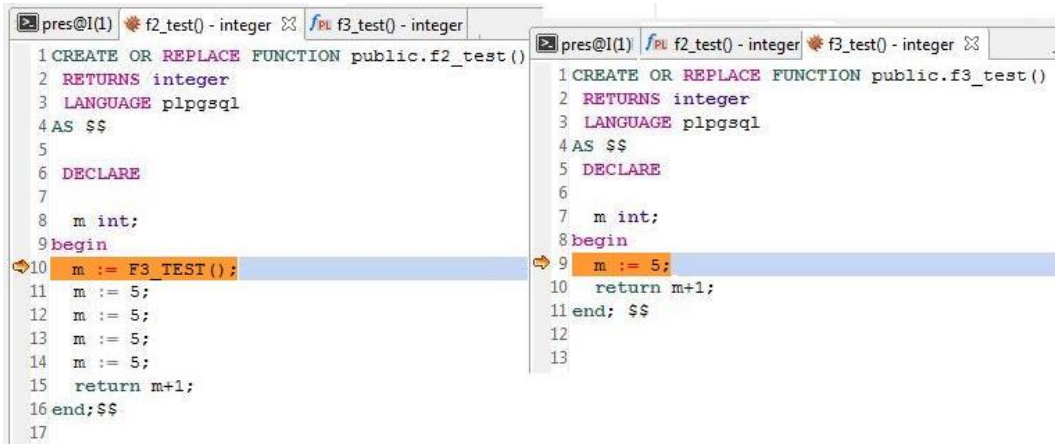
当单步跳入一个函数，Data Studio 执行当前语句，然后进入调试模式，调试位置由左侧标尺区域中的箭头表示。如果执行语句调用另一个函数，Data Studio 将单步跳入该函数。一旦函数中的所有语句执行完成，Data Studio 将跳回其调用函数的下一语句。

单步进入(F7)，进入下一语句如果按“**继续**”，PL/SQL 代码执行将继续。

例如：

跳入第 8 行时，进入 m := F3_TEST();，即进入 f3_test()中的第 9 行。持续单步进入（F7），进入每一行，从而执行 f3_test()中的所有语句。一旦该函数中的所有语句执行完毕，Data Studio 跳回 f2_test()中的第 10 行。

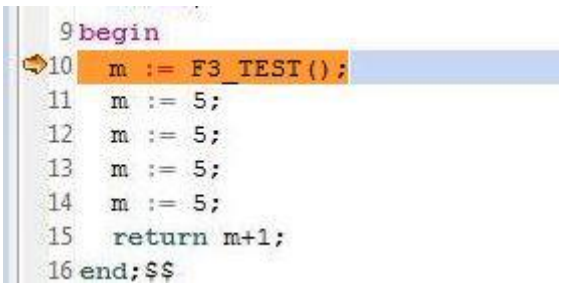
以函数名为标题的页签中，当前调试对象用星号（*）标记。



单步跳过

单步跳过与单步进入相同，除非调用了另一个函数，否则不会跳入该函数。该函数将运行，进入当前函数中的下一个语句。F8 是单步跳过的快捷键。如果该调用函数内设置了断点，单步跳过将进入该函数，并命中该断点。

下面的例子中，如在第 10 行单击“**单步跳过**”，Data Studio 将运行 f3_test()进程。



光标移动到 f2_test()中的下一语句，即 f2_test()中的第 11 行。

```
 9 begin
10   m := F3_TEST();
11   m := 5;
12   m := 5;
13   m := 5;
14   m := 5;
15   return m+1;
16 end;$$
```

用户熟悉了该函数的工作方式，并确保函数的执行不会影响正在调查的问题后，可以单步跳过该函数。

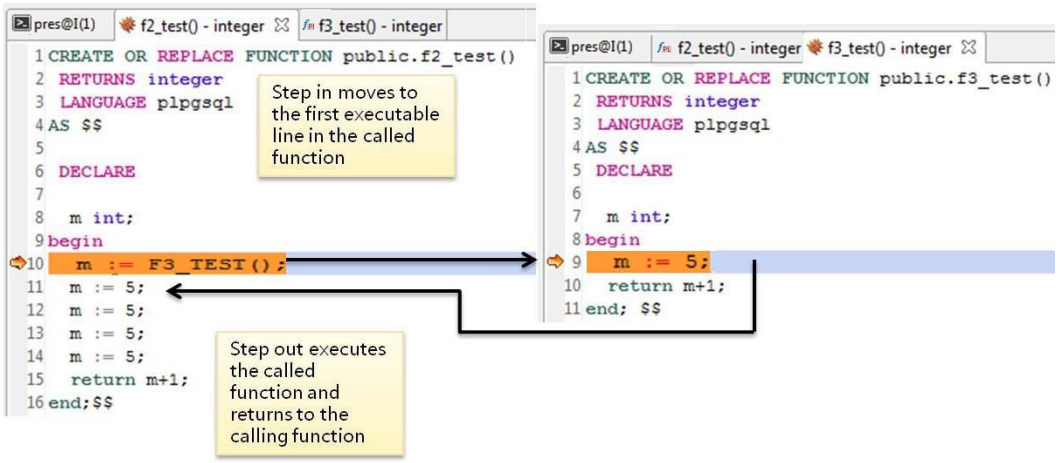
说明

同单步进入该行一样，单步跳过不包含函数调用的一行代码执行该行。

单步退出

子程序单步退出，继续该函数的执行，在该过程返回其调用函数后，暂停执行，确定该函数的剩余部分无需调试后，可单步退出该段函数。然而，如果该函数其余部分设置了断点，在返回调用函数前，该断点将被命中。

单步跳过和单步退出过程都会执行函数。单步退出操作的快捷键是“Shift+F7”。



上图所示例子中，

- 选择“**调试** > **单步进入**”单步进入 f3_test()。
- 选择“**调试** > **单步退出**”单步退出 f3_test()。

继续执行

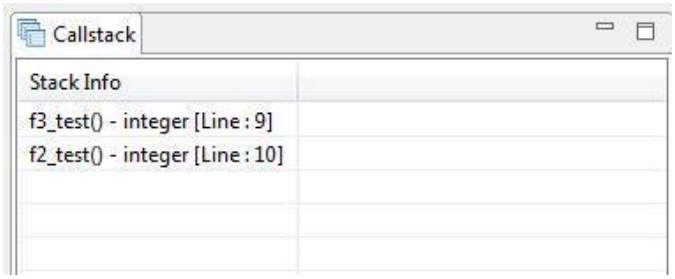
调试进程在某一位置停止时，可在“**调试**”菜单中选择“**继续**”（F9）或者单击工具栏中的继续 PL/SQL 函数执行。

查看调用堆栈

“**调用堆栈**”窗格展示调用过程时的过程链。可从最小化窗口窗格中打开“**调用堆栈**”窗格。最近的过程在顶部，最早的在底部，每个程序名的末尾是该过程的当前行号。

双击“**调用堆栈**”窗格中的函数名，通过“**调用堆栈**”窗格在多个函数中导航。例如，当 f2_test()调用 f3_test()的第 10 行时，调试指针将指向调用函数中的首个可执行行（即上一个例子中的第 9 行）。

这种情况下，“**调用堆栈**”窗格如下图所示：



说明

按下“Alt+J”复制“调用堆栈”窗格内容。

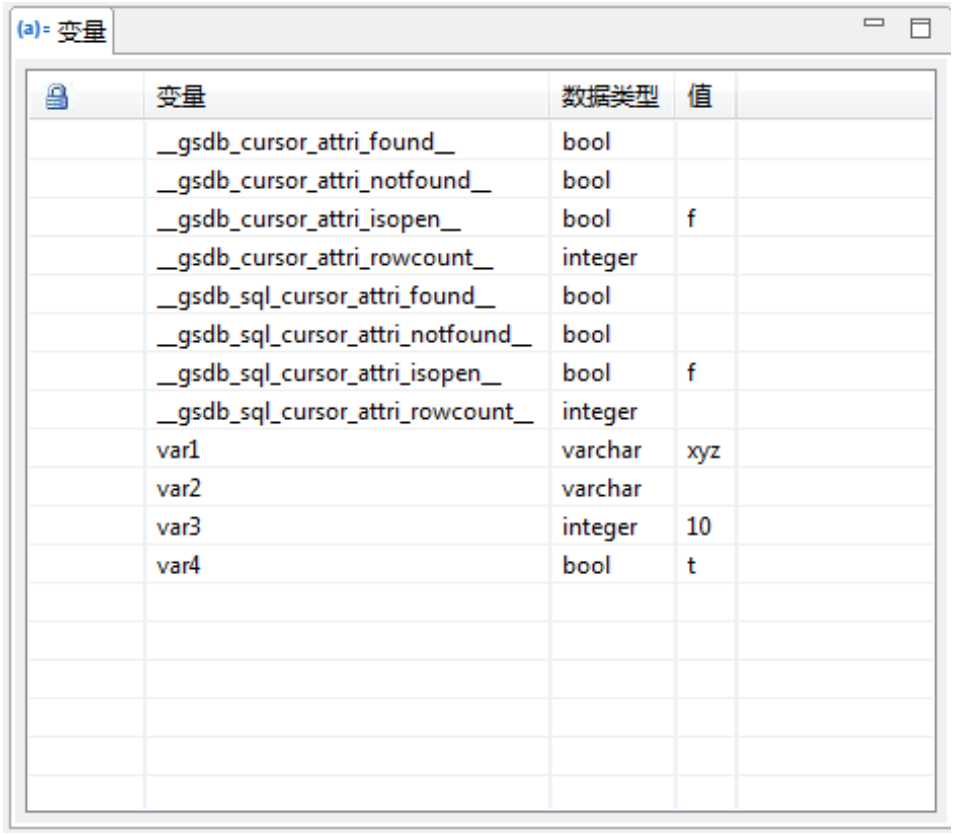
4.14.4 查看调试信息

使用 Data Studio 时，可通过一些调试窗口查看调试信息。本节介绍可用于检查调试信息的操作：

- 变量操作
- 查看结果

变量操作

“**变量**”窗格用于监视信息或估算值。可从最小化窗口窗格中打开“**变量**”窗格。通过这个窗格，可以估算或者修改 PL/SQL 过程中的变量或参数。执行代码时，一些本地变量可能被修改。



说明

使用键盘上的 Alt+K 键复制“变量”窗口内容。

双击该变量的对应行，在运行时间内手动修改变量值。

单击“**变量**”窗格中的“**变量**”、“**数据类型**”或者“**值**”列，对值进行分类。例如，将百分比变量从 5 改为 15，双击“**变量**”窗口中的对应行弹出“**设置变量值**”对话框，提示您输入变量值，单击“**确定**”。

如要设置 NULL 为变量值，在“**值**”列中输入 NULL 或 null。

如果变量为只读，该变量旁会用标示。

用户不能更新这些变量。被称为常数的变量不会以只读格式出现在“**变量**”窗格中，但是更新该变量时，会出现错误。

说明

- “变量”窗格中，如果输入的是'NULL'或 NULL，参数值显示为 NULL 同样适用于字符串数据类型。
- 用 Data Studio 将值设为变量后，那么该变量的值与由 gsql 中执行的“选择表达式”语句返回的值相同。

设置/展示变量	说明
设置 NULL 值	1. 在“变量”窗格中双击一个变量值。 弹出对话框。

设置/展示变量	说明
	2. 设置值为空。
设置字符串值	设置字符串值如下： • 要设置为 abc，则输入 abc。 • 要设置为 Master's Degree，则输入 Master''s Degree。 • 要将变量设置为文本（NULL），在“变量”窗格中设置 NULL。
设置 Boolean 值	Boolean 值 t 或 f 上加单引号。将 *t* 设成一个 boolean 变量，则在 Variable 窗格中输入*'t'*。
显示变量值	如果变量值是 NULL 文本，则显示为 *NULL*。 如果变量值是 NULL，则显示为空。 如果变量值为字符串，比如 abc，则显示为 abc。

查看结果

“**结果**”页签显示 PL/SQL 调试会话的结果，在页签顶部会展示出对应过程名。只有出现执行 PL/SQL 程序的结果时，“**结果**”页签才会自动弹出。

可以单击“**结果**”页签上的图标复制页签内容。详情请参见 4.20.10 使用 SQL 终端。

说明

- “结果”页签中的工具指导最多可以展示 10 行，每行最多包含 80 个字符。
- 如果执行查询的结果是 NULL，则显示为<NULL>。
- 表格数据中的 Tab 字符（ASCII 009）不会显示在“结果”、“查看表数据”和“属性”窗口中。复制/导出的数据中会正确包含 Tab 字符。工具提示中也会正确显示 Tab 字符。

4.15 使用函数/过程

4.15.1 概述

本节介绍如何在 Data Studio 中使用函数/过程和 SQL 函数。

说明

对于以下操作，Data Stutio 支持 PL/pgSQL 和 SQL 语言。

- 4.11 创建函数/过程

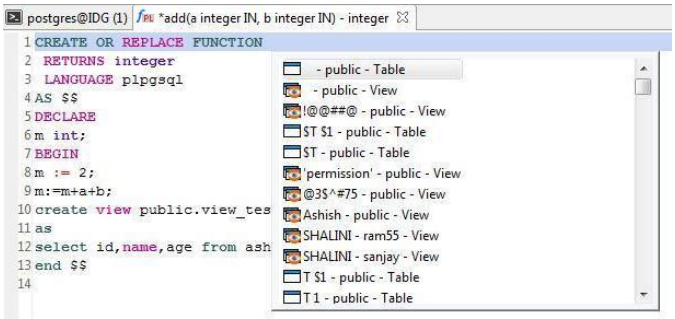
- 4.12 编辑函数/过程
- 4.15.3 导出函数/过程 DDL
- 4.15.5 删除函数/过程

4.15.2 在“PL/SQL Viewer”页签中选择数据库对象

Data Studio 在“PL/SQL Viewer”页签中显示建议列表，提供建议的模式名、表名、列名、视图、序列和函数/对象。

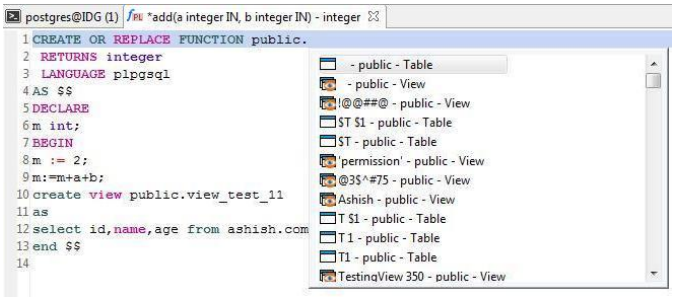
执行以下步骤选择数据库对象：

步骤 1 按下“Ctrl+空格”，输入所需数据库父对象名。列表内容会随输入的数据库对象名进行调整。数据库对象列表显示连接到“SQL 终端”页签的数据库的所有对象。



步骤 2 按下键盘上、下键移动至要选择的数据库父对象，然后按下回车键或双击选择数据库父对象。

步骤 3 按下“.”键列出所有数据库子对象。



步骤 4 按下键盘上、下键移动至要选择的数据库子对象，然后按下回车键或双击选择数据库子对象。

选择后，数据库子对象会添加至数据库父对象后（带英文句号“.”）。

说明

- 自动建议也适用于所有用户有权访问的模式对象的关键字、数据类型、模式名、表名、视图和表别名，方式同上。

  以下查询示例中指定了别名对象：

```
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>
```

- 如下场景下，自动建议可能会在终端显示“正在加载”：
- 当对象数量超过“加载上限”字段中出现的值时，不会加载这些对象。详情请参见 4.8.2 添加连接。
- 当要加载的对象已经添加在“不包含”列表项中时，不会加载这些对象。
- 从服务器上获取到对象时存在时延。
- 如果不同场景下存在名称相同的对象，则自动建议时会同时显示这些对象的子对象。

  例如：

  如果有两个模式分别名为 public 和 PUBLIC，则会显示这两个模式的所有子对象。

----结束

4.15.3 导出函数/过程 DDL

执行如下步骤导出函数/过程 DDL：

步骤 1 在“**对象浏览器**”窗格中，右键单击所选函数/过程，选择“**导出 DDL**”。

Data Studio 显示“**Data Studio 安全免责声明**”对话框。

步骤 2 单击“**确定**”。

Data Studio 显示“**另存为**”对话框。

步骤 3 在“**另存为**”对话框中，选择 DDL 的保存位置，单击“**保存**”。状态栏会显示操作进度。

说明

- 要终止导出操作，双击状态栏，打开“进度视图”页签，单击。有关详情，请参见取消导出表数据操作。
- 如果文件名包含 Windows 中文件名不支持的字符，则文件名的名称会与模式名称不同。
- 要执行该操作，需要 Microsoft Visual C Runtime 文件（msvcrt100.dll）。详情请参阅 4.25 故障处理。
- 可以选择多个对象导出 DDL，导出 DDL 操作不支持的对象列表请参见批量导出章节。

“**导出完成**”对话框和状态栏显示已完成操作的状态。

数据库编码	文件编码	支持导出 DDL
UTF-8	UTF-8	是
	GBK	是
	LATIN1	是

数据库编码	文件编码	支持导出 DDL
GBK	GBK	是
	UTF-8	是
	LATIN1	否
LATIN1	LATIN1	是
	GBK	否
	UTF-8	是

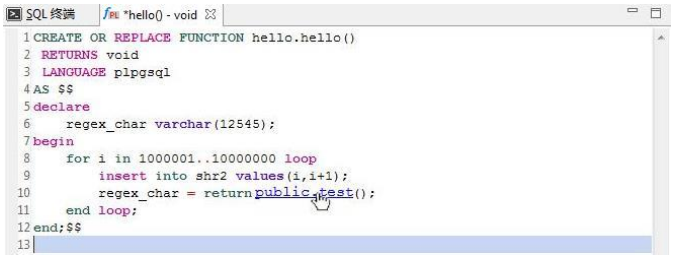
----结束

4.15.4 在“PL/SQL Viewer”页签中查看对象属性

可在 Data Studio 中查看表属性和函数/过程和 SQL 函数。

执行如下步骤查看表属性：

步骤 1 按下 Ctrl 键，同时将光标移动到表名上。



步骤 2 单击高亮显示的表名。Data Studio 显示所选表的属性。

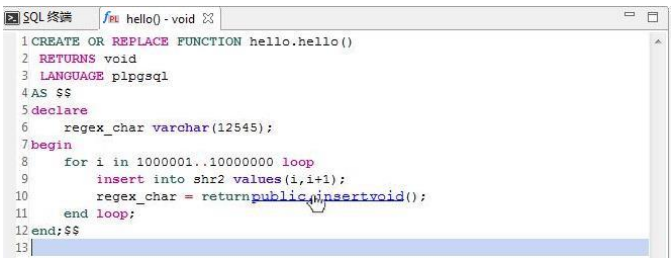
说明

表属性为只读。

----结束

执行如下步骤查看函数/过程或 SQL 函数：

步骤 1 按下 Ctrl 键，同时将光标移动到函数/过程名或 SQL 函数名上。

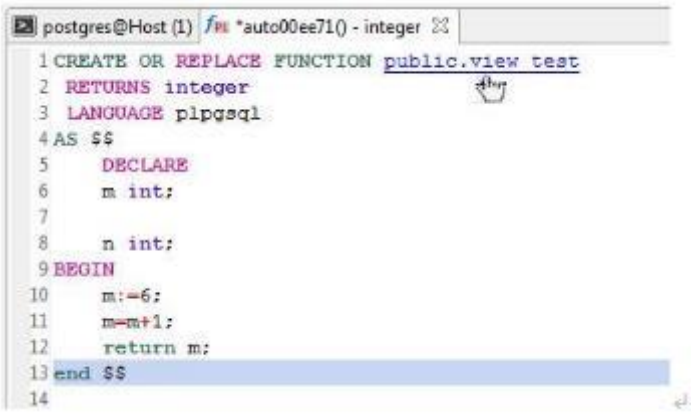


步骤 2 单击突出显示的函数/过程名或 SQL 函数名。Data Studio 在“PL/SQL Viewer”页签中显示所选函数/过程或 SQL 函数。

----结束

执行如下步骤查看对象 DDL：

步骤 1 按下 Ctrl 键查看对象 DDL 名称。



步骤 2 单击高亮显示的“**查看对象 DDL**”名称。Data Studio 根据您的选择在一个新页签显示查看对象 DDL。

----结束

4.15.5 删除函数/过程

可单独或批量删除函数/程序。要进行批量删除，详情请参见 4.21.2 批量删除对象。

执行如下步骤删除函数/过程或 SQL 函数对象：

步骤 1 在“**对象浏览器**”窗格中右键单击函数/过程对象，选择“**删除函数/过程**”。

步骤 2 如需批量删除，则选中两个或多个函数/过程对象，选择“**删除对象**”。

步骤 3 Data Studio 提示确认该操作窗口中，单击“**是**”完成该操作。

状态栏显示已完成操作的状态。

----结束

4.15.6 执行函数/过程

连接数据库后，所有的存储函数/过程和触发器将自动移动到“**对象浏览器**”窗格。可使用 Data Studio 执行 PL/SQL 程序或 SQL 函数

说明

- 如过程前后包含空白行，该过程发送服务器前将被修改，且从服务器获取后显示源信息时，在 Data Studio 将再次修改。

- 执行 Data Studio 上的任何过程时，输入需和 gsql 客户端保持一致。Data Studio 中如未输入，NULL 值将作为输入值。

  例如：

  -执行带字符串的过程，data 为取值。

  -执行带日期的过程，取值如下：to_date('2012-10-10', 'YYYY-MM-DD')。
- 参数类型为 OUT 和 INOUT 的函数/过程不能直接执行。
- Data Studio 不执行带未知数据类型参数的函数。

右键单击“**对象浏览器**”窗格中的函数/过程，可执行如下操作：

- 刷新程序，从服务器获取最新程序
- 执行函数/过程或 SQL 函数
- 调试 PL/SQL 函数
- 删除调试对象

执行 PL/SQL 程序或 SQL 函数

执行如下步骤执行 PL/SQL 程序或 SQL 函数：

步骤 1 双击打开 PL/SQL 程序或 SQL 函数。每个调试对象都会在新页签中打开。Data Studio 中最多可打开 100 个页签。

步骤 2 在工具栏单击，或从菜单栏选择“**运行** > **执行**”。

还可右键单击“PL/SQL Viewer”页，选择“**执行**”。

步骤 3 弹出的“**调试函数/过程**”对话框提示您输入信息。

说明

如无输入参数，“调试函数/过程”对话框不会弹出。这时，将执行 PL/SQL 程序。如有结果，显示在“结果”窗口。

步骤 4 在“**调试函数/过程**”对话框中输入信息，单击“**确定**”。

设置 NULL 为参数值，输入 NULL 或 null。

- 如果提供的值不以单引号开始，则 Data Studio 在该值前后添加单引号，完成设置。
- 如果所提供的值以单引号带单引号，Data Studio 则不添加单引号，完成数据类型设置。

  例如：对于支持的数据类型等，执行查询如下：

  ```
  select func('1'::INTEGER);
  select func('1'::FLOAT);
  select func('xyz'::VARCHAR);
  ```
- 如已提供引号，需对引号进行转义。

  例如：如输入值为 ab'c，则需输入 ab"c。

“**结果**”页签显示执行 PL/SQL 程序的结果，及在“**消息**”页签显示执行的操作信息。可点击 复制“**结果**”页签内容。关于工具栏选项的详情，请参见 4.20.10 使用 SQL 终端。

如果在执行过程中丢失连接，请参阅执行 SQL 查询获取有关重新连接选项的具体信息。

----结束

4.15.7 授权/撤销权限

执行以下步骤授权/撤销权限：

步骤 1 右键单击函数/过程并选择“**授权/撤销权限**”。

弹出“授权/撤销权限”对话框。

步骤 2 参考 4.13 授权/撤销权限（函数/过程）来进行授权/撤销权限。

----结束

4.16 表（GaussDB(DWS)）

4.16.1 概述（管理表）

本节介绍如何高效地管理表。

说明

- 用户需填写所有必选参数方可完成操作。必选参数用星号（*）标识。

4.16.2 创建普通表

4.16.2.1 概述

本节介绍如何创建普通表。

表是由数据库管理员维护的逻辑结构，由行和列组成。可以从数据角度，将表定义为数据定义的一部分。定义表之前，需首先定义数据库和模式，本节将介绍如何利用 Data Studio 创建表。执行如下步骤在数据库中定义表：

步骤 1 在“**对象浏览器**”窗格中，右键单击“**普通表**”，选择“**创建普通表**”。

步骤 2 定义基本表信息，如表名称、表类型等。详情请参见提供基本信息。

步骤 3 定义列相关信息，如列名、数据类型模式、数据类型、列约束等。详情请参见定义列。

步骤 4 选择表数据分布信息详情请参见选择数据分布类型需修改。

步骤 5 为不同约束类型定义列约束。约束类型包含主键（PRIMARY KEY）、唯一（UNIQUE）、检查（CHECK）。详情请参见定义表约束。

步骤 6 定义表索引信息，如索引名称、访问方法等。详情请参见定义索引。

在“**SQL 预览**”页签，可以查看输入所自动生成的 SQL 查询。详情请参见 SQL 预览。

----结束

提供基本信息

在模式中创建表时，当前模式将作为待创建表的模式。创建过程中，需要执行如下若干步骤。创建普通表时，需提供以下信息：

步骤 1 填写“**表名**”。“**表名**”指定要创建的表名称。

说明

选择“区分大小写”复选框可保存“表名”字段中输入的文本的大小写。例如，如果输入的表名称“Employee”，则表名将创建为“Employee”。

创建表的模式名称显示在“**用户模式**”中。

步骤 2 从“**表存储方式**”中选择表存储方式。

步骤 3 选择“**表类型**”。“**表类型**”指定表类型。

- “**正常**”：如指定为“正常”，则创建一个标准的表。
- “**非正常**”：如指定为“非正常”，则创建一个无日志表。对无日志表写入数据时，将不记录到日志中，这样使无日志表的写入速度大大超过一般表的速度。尽管如此，无日志表也是不安全的：在冲突或异常关闭后，无日志表会被自动截断；另外，无日志表的内容不会被备份到备机，并且，创建无日志表的索引时，也不会自动录入日志。

步骤 4 选择选项：

- 如不存在相同名称的表，勾选“**如果不存在**”，创建该表。
- 勾选“**带 OIDS**”，为新创建的表分配 OID（对象标识符）。创建需要带 OID 的新表时，请选择该选项。
- 选择“**填充因子**”表填充因子取值范围为 10~100。默认值为 100，表示完全填充。

“**填充因子**”指定为较小值时，“INSERT”操作仅填充表页面到指定的百分比。表页面预留的剩余空间，将用于更新该页面的行。这样使得“UPDATE”操作可以在和原页面相同的页面上，放置更新后的行内容，比放置在不同的页面上更加高效。对于从未更新过表项的表来说，完全填充是最好的选择。但是对于更新规模较大的表，较小填充因子更加合适。TOAST 表不支持设置该参数。

步骤 5 在“**设置表描述**”框中填写表的简短描述。

步骤 6 提供完这些通用信息后，单击“**下一步**”定义表的列信息。

----结束

下表列出了“**一般**”中普通表支持的字段。

表4-8 支持的字段

字段	行存表	列存表	ORC 表
表类型	✓	✓	✗
如果不存在	✓	✓	✓
带 OIDS	✓	✗	✗
填充因子	✓	✗	✗

定义列

列定义了行中的信息单元，每一行为一个表项，每一列为应用于所有行的信息的分类。数据库添加表时，可以定义组成该数据库的列，列决定了表可以包含的数据类型。提供表的通用信息后，单击“**列**”页签定义列清单，每一列包含名称、数据类型和其他可选属性。

仅可在普通表中的已有列上进行如下操作：

- 移除列
- 编辑列
- 移动列

执行以下步骤定义表的列：

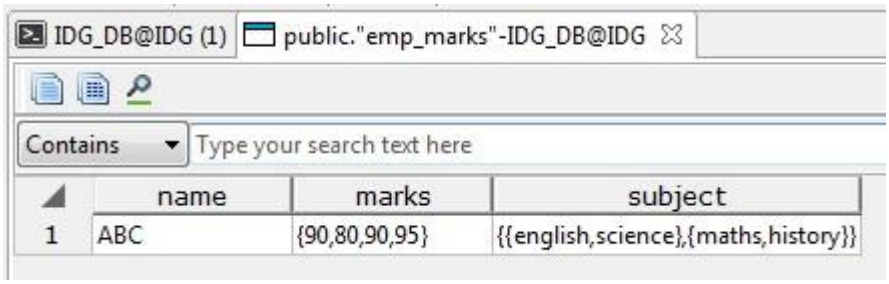
步骤 1 在“**列名**”区域填写列名称。“**列名**”指新表中待创建的列的名称，列名在表中唯一。

说明

勾选“区分大小写”复选框可保留“列名”字段中输入的文本的大小写。例如，如果输入的表名称为“Name”，则表名将创建为“Name”。

步骤 2 选择“**阵列维度**”。“**阵列维度**”指列的阵列尺寸。

例如：如果某列的数组维度定义为 integer []，该列数据会添加为单维数组。



上表的 marks 列创建为单维数据，subject 列创建为二维数据。

步骤 3 从“**数据类型**”中选择列的数据类型。例如，为整数值选择“bigint”。

对于复杂的数据类型：

- 从“**模式数据类型**”中选择模式。
- 从“**数据类型**”中选择对应的数据类型。该列表显示所选模式的表和视图。

说明

用户无法选择自定义数据类型。

步骤 4 在“**精度/大小**”字段选择所输入数据的类型。仅当数据类型可以用精度/尺寸来定义时，该选项可选。

步骤 5 在“**范围**”字段选择输入的数据类型的范围。

步骤 6 根据需要在“**列约束**”区域选择如下约束：

- “**非空**”：指该列不能包含空值。
- “**唯一**”：指列可能仅包含唯一值。
- “**默认**”：指定该列未定义值的情况下使用的默认值。
- “**检查**”：指生成 boolean 结果的表达式，该结果中，新增行或更新行的操作需成功执行。

步骤 7 如果要在“**创建普通表**”对话框的“**列**”页签添加列注释信息，则在“**设置列描述（最多 5000 个字符）**”文本框中进行输入，之后点击“**添加**”。此外，也支持通过单独的新增列窗口添加注释信息。完成之后，可以在普通表的属性窗口中查看该信息。

步骤 8 输入新增列的所有信息后，单击“**添加**”按钮也可以从清单中删除列或修改列顺序，定义完所有列后，单击“**下一步**”。

----结束

下表列出了“**列**”中普通表支持的字段。

表4-9 支持的字段

字段	行存表	列存表	ORC 表
阵列维度	√	x	x
模式数据类型	√	x	x
非空	√	√	√
默认	√	√	√
唯一	√	x	x
检查	√	x	x

移除列

执行以下步骤移除列：

步骤 1 选择要移除的列。

步骤 2 单击“**删除**”。

----结束

编辑列

执行以下步骤编辑列：

步骤 1 选择要编辑的列。

步骤 2 单击“**编辑**”。

步骤 3 编辑列的详细信息，单击“**更新**”保存更改。

说明

开始编辑列后，必须完成并保存更改才能进行其他操作。

----结束

移动列

可移动表格中列的位置。要移动某列，选中该列后单击“**向上**”或“**向下**”。

选择数据分布类型需修改

数据分布是指如何分布一个表或如何在多个数据节点间复制一个表。

分布类型如下表所示：

分布类型	说明
默认分布	分配默认分布类型。
REPLICATION	每一行复制到数据库集群的所有数据节点。
HASH	根据指定列的哈希值分布行。
RANGE	根据 range 值分布行。
LIST	根据 list 值分布行。

选择数据分布后，单击“**下一步**”。

下表列出了“**数据分布**”中普通表支持的选项。

表4-10 分布类型

分布类型	行存表	列存表	ORC 表
默认分布	√	√	x

分布类型	行存表	列存表	ORC 表
HASH	√	√	√
REPLICATION	√	√	x

定义表约束

创建约束为可选操作一个表有一个（且只能有一个）主键，建议创建主键。

在“**约束类型**”中可选择如下选项：

- 主键（“PRIMARY KEY”）
- 唯一（“UNIQUE”）
- 检查(“CHECK”)

主键

主键是行的唯一标识，包含一列或者多个列。

一个表只可指定一个主键，作为列约束或表约束主键约束应命名一组列，该组列不同于同一表定义的任何唯一约定命名的其他组的列。

在“**约束类型**”中选择“**PRIMARY KEY**”，然后填写“**约束名**”。在“**可用列**”清单中选择列，单击“添加”。如需多列主键，为其他列重复执行该步骤。

表的“**填充因子**”取值范围为 10~100。默认值为 100，表示完全填充。“**填充因子**”指定为较小值时，“INSERT”操作仅填充表页面到指定的百分比。表页面预留的剩余空间，将用于更新该页面的行。这样使得“UPDATE”操作可以在和原页面相同的页面上，放置更新后的行内容，比放置在不同的页面上更加高效。对于从未更新过表项的表来说，完全填充是最好的选择。但是对于更新规模较大的表，较小填充因子更加合适。TOAST 表不支持设置该参数。

“**延迟**”：勾选该复选框延迟该选项。

“**初始化延迟**”：勾选该复选框，在设置的默认时间点检查约束。

然后，在“**约束**”区域框单击“**添加**”。

可单击“**删除**”，删除“**约束**”中的主键。

各域的必选参数用星号（*）标识。

唯一

在“**约束类型**”中选择“**UNIQUE**”，然后填写“**约束名**”。

在“**可用列**”清单中选择列，单击“**添加**”。如需多列“UNIQUE”，为其他列重复执行该步骤添加第一列后，“UNIQUE”名自动从表名填入该名称支持修改。

“**填充因子**”：详情请参见主键小节。

“**延迟**”：详情请参见主键小节。

“**初始化延迟**”：详情请参见主键小节。

可单击“**删除**”删除“**约束**”中的“**UNIQUE**”。

各域的必选参数用星号标识。

检查

在“**约束类型**”中选择“**CHECK**”，然后填写“**约束名**”。

执行“INSERT”或者“UPDATE”操作，如果检查表达式错误，表数据不可更改。

双击“**可用列**”清单中的列，则“**检查表达式**”编辑行插入到当前光标位置。

然后，在“**约束**”区域框单击“**添加**”。也可单击“**删除**”，删除“**约束**”中的“**CHECK**”。各域的必选参数用星号标识。定义完所有列后，单击“**下一步**”。

下表列出了“**表约束**”中普通表支持的选项。

表4-11 约束类型

约束类型	行存表	列存表	ORC 表
CHECK	√	x	x
UNIQUE	√	x	x
PRIMARY KEY	√	x	x

定义索引

创建索引为可选操作。索引主要用于增强数据库性能。该操作建立指定表中指定列的索引，如需创建“**唯一索引**”，勾选该复选框。

在“**访问方式**”中选择要使用的索引方法名称。默认方法是 B-tree。

索引的“**填充因子**”指索引方法填充索引页面的百分比。“**访问方式**”为 B-trees 时，初次建立索引以及在右侧扩展索引（填写新的最大键值）时，叶子页面填充到该百分比如果后续完全填满，页面将拆分，这样会导致索引效率逐步衰减。B-trees 使用默认填充因子 90，也可以选择 10~100 范围内的整数。如果为静态表，填充因子 100，这样可以尽量减小索引的物理尺寸。对于更新量较大的表，推荐填充因子设置为较小值，这样可以尽量减少页面拆分的需求。其他索引方法使用的填充因子不同，但是比较类似。默认的填充因子随方法不同而不同。

可以直接提到索引的用户定义的表达式，也可以使用“可用列”创建索引在“**可用列**”列表中选择列，单击“**添加**”。如需多列索引，为其他列重复执行该步骤。

输入新增索引的所有信息后，单击“**添加**”按钮。

可单击“**删除**”删除清单中的索引定义完所有列后，单击“**下一步**”。

下表列出了“**索引**”中普通表所支持的字段/选项。

表4-12 支持的字段/选项

字段/选项	行存表	列存表	ORC 表
唯一索引	√	x	x
btree	√	√	x
gin	√	√	x
gist	√	√	x
hash	√	√	x
psort	√	√	x
spgist	√	√	x
填充因子	√	x	x
用户自定义表达式	√	x	x
部分索引	√	x	x

SQL 预览

Data Studio 按照“**创建普通表**”向导中输入的内容，生成一个 DDL 语句。

可以查看、选择、复制该查询，但是无法编辑。

- 要选择查询，按“Ctrl+A”，或单击右键，选择“**全选**”。
- 要复制所选查询，按“Ctrl+C”，或单击右键，选择“**复制**”。

单击“**完成**”创建表。单击“**完成**”后，生成的查询将被发送到服务器。错误会显示在对话框和状态栏中。

4.16.2.2 管理列

创建表后，可以在该表中添加新的列。仅在普通表中，可对当前列执行如下操作：

- 创建列
- 重命名列
- 切换为非空
- 删除列
- 设置列缺省值
- 更改数据类型

创建列

执行以下操作在当前表中添加一列：

步骤 1 右键单击表下的“**列**”，选择“**添加列**”。

弹出“**添加新列**”对话框，提示在新列中输入详细信息，单击“**添加**”。

步骤 2 完成操作后，该列添加到对应的表中。

Data Studio 在状态栏显示操作状态信息。

----结束

重命名列

执行以下操作重命名列：

步骤 1 右键单击列，选择“**重命名**”。

弹出“**重命名列**”对话框。

步骤 2 提示输入新名称，单击“**确定**”。状态栏显示操作状态。

----结束

切换为非空

执行如下操作设置或重置“**非空**”选项：

步骤 1 右键单击列，选择“**转换为非空值**”。

Data Studio 显示“**切换非空属性**”对话框。

步骤 2 单击“**确定**”完成该操作。状态栏显示操作状态。

----结束

删除列

执行以下步骤删除列：

步骤 1 右键单击所需列，选择“**删除**”，删除列。该列将从表中完全删除。

显示“**删除列**”对话框。

步骤 2 单击“**确定**”完成操作。Data Studio 在状态栏显示操作状态。

----结束

设置列缺省值

执行如下操作设置列的缺省值：

步骤 1 右键单击列，选择“**设置列缺省值**”。

弹出对话框显示当前缺省值（如果已设置），提示您提供缺省输入值。

步骤 2 输入值，单击“**确定**”。Data Studio 在状态栏显示操作状态。

----结束

更改数据类型

执行如下操作更改列数据类型：

步骤 1 右键单击所需列，选择“**更改数据类型**”。

弹出“**更改数据类型**”对话框。

说明

修改复杂数据类型时，现有数据类型将显示为“未知”。

步骤 2 选择“**模式数据类型**”和“**数据类型**”。根据选择，如“**精度/大小**”已启用，输入详细信息，单击“**确定**”。Data Studio 在状态栏显示操作状态。

----结束

4.16.2.3 管理约束

仅在普通表创建后，可执行如下操作：

- 创建约束
- 重命名约束
- 删除约束

创建约束

执行如下步骤添加新约束到当前表：

步骤 1 右键单击表下的“**约束**”，选择“**添加约束**”。

弹出“**添加新约束**”对话框，提示在新约束中输入信息。

步骤 2 输入“**约束名**”和“**选定的列**”，单击“**添加**”。完成操作后，该约束添加到表中。

Data Studio 在状态栏显示操作状态信息。

说明

如果“约束名”字段中已经提供了约束名称，则状态栏将显示该约束名称，否则将不显示该约束名。

----结束

重命名约束

执行如下步骤重命名约束：

步骤 1 右键单击约束，选择“**重命名**”。

弹出“**重命名约束**”对话框。

步骤 2 输入名称，单击“**确定**”。Data Studio 在状态栏显示操作状态信息。

----结束

删除约束

执行如下步骤删除约束：

步骤 1 右键单击表，选择“**删除**”。

显示“**删除约束**”对话框。

步骤 2 单击“**确定**”完成操作。Data Studio 在状态栏中显示操作的状态。

----结束

4.16.2.4 管理索引

可以在表中创建索引，方便更快更有效地查找数据。

创建表后，可以在该表中添加新索引。仅可对普通表的当前索引执行以下操作：

- 创建索引
- 重命名索引
- 修改填充因子
- 删除索引

创建索引

执行如下步骤添加新索引到当前表：

步骤 1 右键单击表下的“**索引**”，选择“**创建索引**”。

弹出“**创建索引**”对话框。

步骤 2 输入详细信息，单击“**创建**”或单击“**预览查询**”按钮预览建索引语句。“**可用列**”无需按序排列。从“**可用列**”移到“**可用列**”的项未排序，与表中列的顺序不相关。您可以使用箭头设置“**可用列**”中列的顺序。Data Studio 在状态栏显示操作状态信息。

----结束

重命名索引

执行如下步骤重命名索引：

步骤 1 右键单击索引，选择“**重命名**”。

弹出“**重命名索引**”对话框。

步骤 2 输入新名称，然后点击“**确定**”。Data Studio 在状态栏显示操作状态信息。

----结束

修改填充因子

执行如下步骤修改填充因子：

步骤 1 右键单击索引，选择“**更改填充因子**”。

弹出“**更改填充因子**”对话框。

步骤 2 选择填充因子，单击“**确定**”。Data Studio 在状态栏显示操作状态信息。

----结束

删除索引

执行以下步骤删除索引：

步骤 1 右键单击索引，选择“**删除**”。

显示“**删除索引**”对话框。

步骤 2 在确认对话框中，单击“**确定**”。Data Studio 将在状态栏中显示操作的状态。此操作将索引从表中删除。

说明

当表的最后一个索引删除后，“是否包含索引”字段的值可能显示为“TRUE”。在对表进行 Vacuum 操作后，此字段的值变为“FALSE”。

----结束

4.16.3 创建外表

刷新“**对象浏览器**”后，可查看在“SQL 终端”或任何其他工具中使用查询执行创建的外部表。

步骤 1 要查看新创建的外部表，在数据库、模式、或外表组级别单击右键并选择“**刷新**”。

说明

- GDS 外表在表名之前用  标示。
- HDFS 外表在表名之前用  标示。
- HDFS 分区外表在表名之前用  标示。

----结束

4.16.4 创建分区表

4.16.4.1 概述

分区是指根据特定方案将逻辑上的一个大表分成较小的物理片区。基于该逻辑的表称为分区表，物理片区称为分区。数据不存储在较大的逻辑分区表上，而是这些较小的物理分区上。

执行以下步骤在数据库中定义表：

步骤 1 在“**对象浏览器**”窗格中，右键单击“**普通表**”，选择“**创建分区表**”。

步骤 2 定义基本表信息，如表名称、表类型等。详情请参见基本信息。

步骤 3 定义列相关信息，如列名、数据类型模式、数据类型、列约束等。详情请参见定义列。

步骤 4 选择表数据分布信息。详情请参见更改分区顺序。

步骤 5 为不同约束类型定义列约束。约束类型包含主键（PRIMARY KEY）、唯一（UNIQUE）、检查（CHECK）。详情请参见定义表约束。

步骤 6 定义表索引信息，如索引名称、访问方法等。详情请参见定义索引。

步骤 7 定义表的分区信息，如分区名称、分区列、分区值等。详情请参见定义分区。

在“**SQL 预览**”页签，可以查看输入所自动生成的 SQL 查询。详情请参见 SQL 预览。

步骤 8 如果要在“**创建分区表**”对话框的“**列**”页签添加列注释信息，则在“**设置列描述（最多 5000 个字符）**”文本框中进行输入，之后点击“**添加**”。

----结束

基本信息

提供以下信息创建表：

有关填写以下字段的详情，请参见提供基本信息。

- 表名
- 模式
- 选项
- 表描述

有关填写其他字段的详情，请参见如下内容：

步骤 1 从“**表存储方式**”中选择表存储方式。

说明

如果表存储方式选择为 ORC，则会创建一个 HDFS 分区表。

步骤 2 在“**ORC 版本**”中输入 ORC 版本号。该字段仅适用于 HDFS 分区表。

步骤 3 提供有关表的基本信息后，单击“**下一步**”以定义表的列信息。

下表列出了每个分区表支持的字段：

表4-13 支持的字段

字段名	行分区	列分区	ORC 分区
表类型	x	x	x
如果不存在	√	√	√

字段名	行分区	列分区	ORC 分区
带 OIDS	x	x	x
填充因子	√	x	x

----结束

定义列

有关定义列的详情，请参见定义列。

下表列出了每个分区类型表的支持字段：

表4-14 支持的字段

字段名	行分区	列分区	ORC 分区
阵列维度	√	x	x
数据类型	√	x	x
非空	√	√	√
默认	√	√	√
唯一	√	x	x
检查	√	x	x

更改分区顺序

您可以根据表中的要求更改分区顺序。要更改顺序，请选择所需的分区，然后单击“**向上**”或“**向下**”。

SQL 预览

请参见 SQL 预览。

编辑分区

执行以下步骤编辑分区：

步骤 1 选择所需分区。

步骤 2 单击“**编辑**”。

步骤 3 根据需要编辑分区详细信息，然后单击“**更新**”以保存更改。

说明

用户完成编辑操作并保存更改后，方可继续其他操作。

----结束

删除分区

执行以下步骤删除分区：

步骤 1 选择所需分区。

步骤 2 单击“**删除**”。

----结束

定义分区

下表列出了各分区类型表支持的字段/选项：

表4-15 支持的字段/选项

字段/选项名	行分区	列分区	ORC 分区
分区类型	按范围	按范围	按值
分区名	√	√	x
分区值	√	√	x

按照以下步骤定义表的分区：

步骤 1 “**一般**”页签中，如果“**表存储方式**”选择为“**ROW**”或“**COLUMN**”，则“**分区类型**”区域会显示“**By Range**”。如果“**表存储方式**”选择为“**ORC**”，则“**分区类型**”区域会显示“**By Values**”。

步骤 2 从“**可用列**”区域选择用于定义分区的列，单击  。

该列会移动到“**分区列**”区域。

说明

- 如果“表存储方式”选择为“ROW”或“COLUMN”，则仅能选择一列用于分区。
- 如果“表存储方式”选择为“ORC”，则最多可选择四列用于分区。
- 最多可选择 4 列用于定义分区。

步骤 3 在“**分区名称**”中输入分区的名称。

步骤 4 单击“**分区值**”旁的  。

1. 在“**值**”列中输入要对表进行分区的值。
2. 单击“**确定**”。

步骤 5 输入分区的所有信息后，单击“**添加**”。

步骤 6 定义所有分区后，单击“**下一步**”。

----结束

您可以对行或列分区表的已有分区执行以下操作。以下操作不适用于 ORC 分区表：

删除分区

编辑分区

定义索引

有关索引定义详情，请参见定义索引。

表4-16 支持的字段/选项

字段/选项名	行分区	列分区	ORC 分区
唯一索引	√	x	x
btree	√	√	x
gin	√	√	x
gist	√	√	x
hash	√	√	x
psort	√	√	x
spgist	√	√	x
填充因子	√	x	x
用户自定义表达式	√	x	x
部分索引	√	x	x

定义表约束

有关定义表约束的详情，请参见定义表约束。

表4-17 支持的字段/选项

选项名	分区	列分区	ORC 分区
Check	√	x	x

选项名	分区	列分区	ORC 分区
Unique	√	x	x
Primary Key	√	x	x

选择数据分布

有关选择分布类型的详情，请参见选择数据分布类型。

表4-18 支持的字段/选项

选项名	行分区	列分区	ORC 分区
默认分布	√	√	x
Hash	√	√	√
Replication	√	√	x

4.16.4.2 管理分区

创建表后，可以添加/修改分区。还可对现有分区执行以下操作：

- 重命名分区
- 移除分区

重命名分区

执行以下步骤重命名分区：

步骤 1 右键单击所需分区，选择“**重命名分区**”。

弹出“**重命名分区**”对话框，提示用户为分区输入新名称。

步骤 2 输入新名称，单击“**确定**”。

Data Studio 在状态栏显示操作状态信息。

----结束

移除分区

执行以下步骤移除分区：

步骤 1 右键单击所需分区，选择“**删除分区**”。

弹出“**删除分区**”对话框。

步骤 2 单击“**确定**”。

该分区会从表中移除。Data Studio 将在状态栏中显示操作的状态。

----结束

4.16.5 授权/撤销权限 - 普通表/分区表

执行以下步骤授权/撤销权限：

步骤 1 右键单击普通表组并选择“**授权/撤销权限**”。

弹出“**授权/撤销权限**”对话框。

步骤 2 打开“**选择对象**”页签，选择待授权/撤销权限的对象，并单击“**下一步**”。

步骤 3 打开“**选择权限**”页签，从“**角色**”中选择对应角色。

步骤 4 在“**选择权限**”页签，勾选“**授予**”或“**撤销**”。

步骤 5 在“**选择权限**”页签，勾选或取消勾选相关权限。

在“**SQL 预览**”页签，可以查看根据以上输入自动生成的 SQL 查询。

步骤 6 单击“**完成**”。

----结束

4.16.6 管理表

4.16.6.1 概述

本章节介绍如何有效地管理表。

说明

- 需要填写所有必选参数才能完成操作。必选参数用星号（*）标示。
- 对外表仅能进行刷新操作。

创建表后，可在表中执行操作。右键单击表，选择要执行的操作。

右键菜单

表的右键菜单中提供更多表格操作项。具体包括：

表4-19 表的右键菜单项

菜单项	说明
查看数据	打开表数据信息。有关详情，请参见 4.16.7.7 查看表数据。
编辑数据	打开表数据编辑窗口。有关详情，请参见 4.16.7.8 编辑表数据。
重建索引	重新创建表索引。有关详情，请参见 4.16.6.4 重建表索

菜单项	说明
	引。
分析	分析表。有关详情，请参见 4.16.6.5 分析表。
截断	截断表数据。有关详情，请参见 4.16.6.3 截断表。
清理	清空表数据。有关详情，请参见 4.16.6.6 清空表。
设置表描述	设置表描述。有关详情，请参见 4.16.6.7 设置表描述。
设置表模式	设置表的模式。有关详情，请参见 4.16.6.8 设置模式。
导出表数据	导出表数据。有关详情，请参见 4.16.7.4 导出表数据。
导入表数据	导入表数据。有关详情，请参见 4.16.7.6 导入表数据。
显示 DDL	显示表 DDL。有关详情，请参见 4.16.7.5 显示 DDL。
导出 DDL	导出表 DDL。有关详情，请参见 4.16.7.2 导出表 DDL。
导出 DDL 和数据	导出 DDL 和表的数据。有关详情，请参见 4.16.7.3 导出表 DDL 和数据。
重命名	重命名表。有关详情，请参见 4.16.6.2 重命名表。
删除	删除表。有关详情，请参见 4.16.6.9 删除表。
属性	显示表属性。有关详情，请参见 4.16.6.10 查看表属性。
授权/撤销权限	为对象授权/撤销权限。有关详情，请参见 4.16.6.11 授权/撤销权限。
刷新	刷新表。

4.16.6.2 重命名表

执行以下操作重命名表：

步骤 1 右键单击表，选择“**重命名**”重命名表。

弹出“**重命名表**”对话框，提示输入新名称输入名称。

步骤 2 输入表名，单击“**确定**”。“**对象浏览器**”中显示更新的表名称。Data Studio 将在状态栏中显示操作的状态。

说明

分区 ORC 表不支持此操作。

----结束

4.16.6.3 截断表

步骤 1 右键单击表，选择“**截断**”。该操作将删除当前表的所有数据。

Data Studio 提示“**截断**”对话框。

步骤 2 单击“**确定**”完成该操作。

弹出消息和状态栏显示已完成操作的状态。

----结束

4.16.6.4 重建表索引

使用索引可帮助用户更快地查找数据。以下场景需要重建索引：

- 索引已破坏，不再包含任何有效数据。虽然理论上不会发生，但事实上，索引可能由于软件或硬件故障而被破坏重建索引提供恢复方法。
- 索引包含很多空的或几乎为空的页面，这种情况会在一些非通用访问类型下 PostgreSQL 中的 B-tree 索引中出现重建索引提供一种通过写新版本的方式减少索引消耗的方法。新版本中无空页面。
- 已修改索引的存储参数（如“填充因子”），且希望保证修改完全生效。

执行以下步骤重建索引：

步骤 1 右键单击表，选择“**重建索引**”。

弹出消息和状态栏显示已完成操作的状态。

说明

分区 ORC 表不支持此操作。

----结束

4.16.6.5 分析表

“分析”统计表和表索引的数据，在数据库内部表中存储统计的信息。数据库中，查询优化器可以访问信息并根据该信息制定更好的查询规划策略。

执行以下步骤分析表：

步骤 1 右键单击表，选择“**分析**”。

“**分析表**”对话框和状态栏显示所有操作的状态信息。

----结束

4.16.6.6 清空表

“**清理**”用于收回空间，便于重复利用。

执行以下步骤清空表：

步骤 1 右键单击表，选择“**清理**”清理表。

“**清空表**”对话框和状态栏显示已完成的操作的状态信息。

----结束

4.16.6.7 设置表描述

执行以下步骤设置表描述：

步骤 1 右键单击表，选择“**设置表描述**”设置表描述。

“**更新描述**”对话框提示输入表描述。

步骤 2 输入描述，单击“**确定**”。

状态栏显示已完成操作的状态。

步骤 3 右键单击表，选择“**属性**”，查看表属性。

----结束

4.16.6.8 设置模式

执行以下步骤设置模式：

步骤 1 右键单击表，选择“**设置表模式**”，设置模式。

Data Studio 显示“**设置模式**”对话框，提示用户为所选表选择新模式。

步骤 2 从下拉列表中选择模式名称，单击“**确定**”已选表将被移动到新模式。

状态栏显示已完成操作的状态。

说明

- 分区 ORC 表不支持此操作。
- 如果指定模式中包含与当前表同名的表，则 Data Studio 不允许为该表设置模式。

----结束

4.16.6.9 删除表

可单独或批量删除表。要进行批量删除，详情请参见 4.21.2 批量删除对象。

该操作从数据库中移除整个表结构（包括表定义及索引信息等）。如需存储数据，需重新创建表。

执行如下步骤删除表：

步骤 1 右键单击表，选择“**删除**”删除表。

Data Studio 提示确认该操作。

步骤 2 单击“**确定**”完成该操作。

状态栏显示已完成操作的状态。

----结束

4.16.6.10 查看表属性

执行如下操作查看表属性：

步骤 1 右键单击表，选择“**属性**”查看表属性。

Data Studio 在不同页签显示所选表的属性（“一般”、“列”、“约束”和“索引”）。

下表列出了可在每个页签上执行的操作，以及数据的编辑和刷新。可双击单元格执行编辑操作。

页签	操作
一般	保存、取消和复制 说明 仅可修改“表描述”字段内容。
列	添加、删除、保存、取消和复制
约束	添加、删除、保存、取消和复制
索引	添加、删除、保存、取消和复制

关于编辑、保存、取消、复制和刷新操作的更多信息，请参阅 4.16.7.8 编辑表数据。

须知

查看表格数据时，Data Studio 会自动调整列宽以获得表视图。用户可以根据需要调整列的大小。如果单元格的文本内容超出了可用的显示区域，则调整单元格列的大小可能会导致 DS 无法响应。

说明

- 一个表显示一个属性窗口。
- 如果一个已打开的表格的属性被修改，刷新并重新打开该表格的属性查看更新后的表格属性信息。
- 如果列的内容包含空格，会在空格处自动断行以适应该列的显示区域。不包含空格的内容不会自动断行。
- 列的大小取决于内容最长的列的长度。
- 刷新（单击  ）“属性”页签后，将显示在“对象浏览器”对表属性所做的任何更改。
- “Data Type”列不允许粘贴操作。

----结束

4.16.6.11 授权/撤销权限

执行以下步骤授权/撤销权限：

步骤 1 右键单击普通表/分区表并选择“**授权/撤销权限**”。

弹出“**授权/撤销权限**”对话框。

步骤 2 参考 4.16.5 授权/撤销权限 - 普通表/分区表来进行授权/撤销权限。

----结束

4.16.7 管理表数据

4.16.7.1 概述

本节描述如何管理表数据。

4.16.7.2 导出表 DDL

执行以下步骤导出表 DDL：

步骤 1 在“**对象浏览器**”窗格中，右键单击所选表，选择“**导出 DDL**”。

Data Studio 显示“**Data Studio 安全免责声明**”对话框。

步骤 2 单击“**确定**”。

Data Studio 显示“**另存为**”对话框。

步骤 3 在“**另存为**”对话 DDL 的保存位置，单击“**保存**”。状态栏会显示操作进度。

说明

- 要终止导出操作，双击状态栏，打开“进度视图”页签，单击。有关详情，请参见取消导出表数据操作。
- 如果文件名包含 Windows 中文件名不支持的字符，则文件名的名称会与模式名称不同。
- 要执行该操作，需要 Microsoft Visual C Runtime 文件（msvcrt100.dll）。详情请参阅 4.25 故障处理。

“**导出完成**”对话框和状态栏显示已完成操作的状态。

数据库编码	文件编码	支持导出 DDL
UTF-8	UTF-8	是
	GBK	是
	LATIN1	是
GBK	GBK	是

数据库编码	文件编码	支持导出 DDL
	UTF-8	是
	LATIN1	否
LATIN1	LATIN1	是
	GBK	否
	UTF-8	是

说明

可选择并导出多个对象的 DDL。批量导出章节列举了不支持导出 DDL 的对象。

----结束

4.16.7.3 导出表 DDL 和数据

通过导出表 DDL 和数据，可导出如下内容：

- 表的 DDL
- 表的行和列

执行以下步骤导出表 DDL 和数据：

步骤 1 在“**对象浏览器**”窗格中，右键单击所选表，选择“**导出 DDL 和数据**”。

Data Studio 显示“**Data Studio 安全免责声明**”对话框。

步骤 2 单击“**确定**”。

Data Studio 显示“**另存为**”对话框。

步骤 3 在“**另存为**”对话 DDL 的保存位置，单击“**保存**”。状态栏会显示操作进度。

说明

- 要终止导出操作，双击状态栏，打开“进度视图”页签，单击。有关详情，请参见取消导出表数据操作。
- 如果文件名包含 Windows 中文件名不支持的字符，则文件名的名称会与模式名称不同。
- 要执行该操作，需要 Microsoft Visual C Runtime 文件（msvcrt100.dll）。详情请参阅 4.25 故障处理。

“**导出完成**”对话框和状态栏显示已完成操作的状态。

数据库编码	文件编码	支持导出 DDL
UTF-8	UTF-8	是
	GBK	是

数据库编码	文件编码	支持导出 DDL
	LATIN1	是
GBK	GBK	是
	UTF-8	是
	LATIN1	否
LATIN1	LATIN1	是
	GBK	否
	UTF-8	是

说明

可从普通表和分区表中选择多个对象，以导出 DDL 和数据。导出的内容包含列、行、索引、约束和分区。批量导出章节列举了不支持导出 DDL 和数据的对象。

----结束

4.16.7.4 导出表数据

执行以下步骤导出表数据：

步骤 1 右键单击表，选择“**导出表数据**”导出表数据。

Data Studio 显示“**导出表数据**”对话框，包含如下选项：

- “**格式**”：表数据可导出为 Excel（xlsx/xls）、CSV、文本或二进制格式。默认为 Excel（xlsx/xls）。
- “**包含标题**”：该选项对 CSV 和文本文件启用。若勾选该选项，则导出的数据中包含列标题。该项默认在导出 CSV 或文本文件时勾选，但不是必选。对 Excel（xlsx/xls）和二进制格式禁用。
- “**引号**”：该项用于定义引号。在此字段中仅能输入单字节字符。引号不得与分隔符相同。该项默认对 CSV 和文本格式启用，但不是必选。对 Excel（xlsx/xls）和二进制格式禁用。
  - 如果表数据中包含分隔符，则会使用该项中指定的符号。
  - 如果引号出现在值中，该值不会被转义。
  - 如果结果中包含多行值，则会用引号引用。
- “**转义符**”：该项定义转义值。该项仅支持单字节字符。“**转义符**”和“**引号**”的值不得相同。该项对 CSV 和文本格式启用，但不是必选。对 Excel（xlsx/xls）和二进制格式禁用。
- “**将 NULL 替换为**”：可指定字符串，用于替换表中的 null 值。该项不支持填入换行符或回车符，最多可填入 100 个字符。该字段的值不得与分隔符和引号值相同。该项默认对 CSV 和文本格式启用，但不是必选。对 Excel（xlsx/xls）和二进制格式禁用。

- “**编码**”字段会自动填充为“**首选项** > **会话设置**”页签选择的编码选项。该字段不是必选。
- “**分隔符**”：该项定义分隔符。可选择提供的分隔符，或在“**分隔符**”区域的“**其他**”字段中自定义分隔符。CSV 格式的默认分隔符为半角逗号“,”。“**其他**”字段中的内容最大可为 10 字节。该项对 CSV 和文本格式启用，但不是必选，对 Excel（xlsx/xls）和二进制格式禁用。如果“**其他**”字段被选中，则必须设置该字段。
- “**所有列**”：勾选该项可快速选中所有列。该项默认勾选。如果要手动选择列，则取消选中该项，并从“**可用列**”中选择要导出的项。
  - “**可用列**”：可通过该项选中要导出的字段。
  - “**选定的列**”：显示所选的待导出字段。字段顺序可调整。此处默认显示所有字段。

说明

有关 xlsx 和 xls 文件支持的行和列大小，请参见 4.26 FAQs 章节中的行和列大小。

- “**文件名**”：指定导出文件的名称。表名默认显示在此字段中。

说明

文件名遵循 Windows 文件命名规范。

- “**导出路径**”：选择保存导出文件的位置。所选路径自动填充“**导出路径**”字段。
- “**安全免责声明**”：显示安全免责声明。要继续导出操作，需阅读并同意该免责声明。
  - “**我同意**”：该项默认勾选。如果取消选择该项，则无法继续进行操作。
  - “**不再显示**”：如果勾选该项，则今后在当前登录的 Data Studio 实例中导出表数据时不再显示安全免责声明。

说明

- 字符串、Double、日期、日历和布尔数据类型按原样存储在 Excel 中。所有其他数据类型转换为字符串存储在 Excel 中。
- 对于 Excel 导出，如果单元格大小超过 32767 个字符，则导出到该单元格的数据会被截断。

步骤 2 填写所需字段，单击“**确定**”。

Data Studio 弹出“**另存为**”对话框。

步骤 3 单击“**保存**”，以所选格式保存导出的数据。状态栏会显示操作进度。

“**数据导出成功**”对话框和状态栏显示已完成操作的状态。

说明

- 在导出表过程中，若磁盘空间已满，Data Studio 显示 I/O 错误。执行以下步骤解决这一问题：
- 点击“确定”关闭数据库连接。
1. 清理磁盘。
2. 重建连接，导出表数据。

- 如果文件名包含 Windows 中文件名不支持的字符，则文件名的名称会与表名称不同。

----结束

取消导出表数据操作

执行以下步骤取消导出表数据操作：

步骤 1 双击状态栏，打开“**进度视图**”页签。

步骤 2 在“**进度视图**”页签中，单击。

步骤 3 在“**取消操作**”对话框中，单击“**是**”。

“**消息**”页签和状态栏显示已取消操作的状态。

----结束

4.16.7.5 显示 DDL

按照以下步骤显示表的 DDL 查询：

步骤 1 右键单击表，然后选择“**显示 DDL**”。

Data Studio 显示所选表的 DDL。

说明

- 每次执行显示 DDL 操作时，都会打开一个新的终端窗口。
- 要执行该操作，需要 Microsoft Visual C Runtime 文件（msvcrt100.dll）。详情请参阅 4.25 故障处理。

数据库编码	文件编码	支持显示 DDL
UTF-8	UTF-8	是
	GBK	是
	LATIN1	是
GBK	GBK	是
	UTF-8	是
	LATIN1	否
LATIN1	LATIN1	是
	GBK	否
	UTF-8	是

----结束

4.16.7.6 导入表数据

导入表数据的前提条件：

- 如果要导入的源文件与导入目标表定义不匹配，需在“**导入表数据**”对话框中修改目标表的属性。目标表中的多余列会插入默认值。
- 请务必了解被导入文件的导出属性，如分隔符、引号、转义字符等等。导出操作期间保存的导出属性在导入文件时无法修改。

执行以下步骤导入表数据：

步骤 1 右键单击表，选择“**导入表数据**”。

Data Studio 显示“**导入表数据**”对话框，包含如下选项：

- “**导入数据文件**”：该字段显示导入文件的路径。可单击“**浏览**”按钮选择其他文件。
- “**格式**”：可将表数据以 CSV、文本或二进制格式导入。默认使用 CSV 格式。
- “**包含标题**”：如果导入文件存在列标题，则需要勾选该字段。该项默认对 CSV 和文本格式勾选，但不是必选。对二进制格式禁用。
- “**引号**”：该字段仅能输入单字节字符。“引号”值中的字符不得与分隔符和参数空值相同。该项默认对 CSV 和文本格式选用，但不是必选。对二进制格式禁用。
- “**转义符**”：该字段仅能输入单字节字符。如果转义符与“引号”值中的字符相同，转义符会替换为'\0'。该字段默认对 CSV 和文本格式选用并使用半角双引号（"），但不是必选。对二进制格式禁用。
- “**用 Null 替换**”：可设置该字段，将表中的空值替换为字符串。导出时使用的空字符串需在导入时使用，且需明确指定。该字段默认对 CSV 和文本格式选用，但不是必选。对二进制格式禁用。
- “**编码**”字段会自动填充为“首选项 > 会话设置”页签选择的编码选项。该字段不是必选。
- “**分隔符**”：可选择系统提供的分隔符或在“**分隔符**”区域的“**其他**”字段自定义分隔符。CSV 和文本格式的默认分隔符为半角逗号（,）。该字段的值不得与“引号”和“用…替换 NULL”字段相同。该项默认对 CSV 和文本格式启用，但不是必选。对二进制格式禁用。如果“其他”字段被选中，则必须设置该字段。
- “**所有列**”：勾选该项可快速选中所有列。该项默认勾选。要手动选择列，取消选中该项，并从“**可用列**”中选择要导出的项。
  - “**可用列**”：可通过该项选中要导出的字段。
  - “**选定的列**”：显示所选的待导出字段。字段顺序可调整。此处默认显示所有字段。

步骤 2 单击“**导入数据文件**”字段旁的“**浏览**”按钮。

Data Studio 显示“**打开**”对话框。

步骤 3 在“**打开**”对话框中，选择要导入的文件，单击“**打开**”。

步骤 4 填写所需字段，单击“**确定**”。

“**进度视图**”页签会显示操作进度。导入的数据会添加至现有表数据中。

“**数据导入成功**”对话框和状态栏显示已完成操作的状态。

----结束

取消导入表数据操作

执行以下步骤取消导入表数据操作：

步骤 1 双击状态栏，打开“**进度视图**”页签。

步骤 2 在“**进度视图**”页签中，单击。

步骤 3 在“**取消操作**”对话框中，单击“**是**”。

“**消息**”页签和状态栏显示已取消操作的状态。

----结束

4.16.7.7 查看表数据

执行以下步骤查看表数据：

步骤 1 右键单击表，选择“**查看数据**”查看表数据。

Data Studio 显示“**查看数据**”页签，可在该页签查看表数据信息。

“**查看数据**”页签的工具栏菜单：

工具栏名称	工具栏图标	说明
复制		用于从“查看数据”页签将数据复制。快捷键为 Ctrl+C。
高级复制		用于将结果窗口中的内容复制。可以复制结果以包含行号和/或列标题。请参阅查询结果以设置此首选项。快捷键为 Ctrl+Shift+C。
显示/隐藏搜索栏		用于显示/隐藏搜索文本区域。点击该按钮可在两种状态间切换。
编码	-	有关如何选择编码，请参见执行 SQL 查询。

搜索区域中的图标：

图标名称	图标	描述
搜索		用于根据定义的标准，搜索显示的表数据。搜索内容不区分大小写。
清空搜索内容		用于清空在搜索字段中输入的搜索内容。

有关调整列位置或对列进行排序的具体信息，请参阅执行 SQL 查询。

- “**查询提交时间**”：提供查询的提交时间。
- 获取的行数和执行时间同时显示，且会显示默认行数。如果有其他记录待获取，此图标将显示为“**更多**”。用户可将光标滚至表底部读取并显示所有行。

须知

- 查看表数据时，Data Studio 会自动调整列宽。用户可以根据需要调整列的大小。如果单元格的文本内容超出了可用的显示区域，则调整单元格列的大小可能会导致 Data Studio 无法响应。
- 表的单元格最多可显示 1000 个字符，超出部分显示为“...”。
- 如果用户从表或“结果”页签的单元格复制数据到任意编辑器（如 SQL 终端/PLSQL 源编辑器、记事本或任意外部编辑器应用），将会粘贴全部数据。
- 如果用户从表或“结果”页签的单元格复制数据到一个可编辑的单元格（本单元格或其他单元格），该单元格仅显示 1000 个字符，并将超出部分显示为“...”。
- 导出表或“结果”页签数据时，导出的文件将包含全部数据。

说明

- 一个表显示一个表数据窗口。
- 如果已打开的表数据被修改了，刷新并重新打开该表数据并在同一打开的窗口查看更新后的表数据。
- 数据在加载过程中，表格下方会出现一条“读取中”的消息。
- 如果列的内容包含空格，会在空格处自动断行以适应该列的显示区域。不包含空格的内容不会自动断行。
- 要复制单元格中的部分内容，先选中所需部分，然后按下“Ctrl+C”或单击  。
- 列的大小取决于内容最长的列的长度。
- 用户可根据个人喜好保存首选项用于定义:
- 要获取的记录数
- 列宽
- 从结果集复制选项

  详情请参阅查询结果。

----结束

4.16.7.8 编辑表数据

执行如下步骤编辑表数据：

步骤 1 右键单击表，选择“**编辑数据**”。

页面显示“编辑表数据”页签。

有关复制、搜索工具栏和字符编码下拉列表选项的说明，请参阅 4.16.7.7 查看表数据。

----结束

Data Studio 仅识别单元格中的如下数据类型：

Bigint、bit、Boolean、char、date、decimal、double、float、integer、numeric、real、smallint、时间、包含时区的时间、时间戳、包含时区的时间戳、tinyint 和 varchar。不支持编辑数组数据类型。

数据库上报的与该操作相关的错误会显示在 Data Studio 中。包含时区的时间列和包含时区的时间戳列均不可编辑。

可在“编辑表数据”页签中执行以下操作：

- 插入
- 删除
- 更新单元格
- 复制
- 粘贴

插入

执行如下步骤插入行：

步骤 1 单击  插入行。

步骤 2 双击单元格，在插入的行中修改填写所需详细信息。

步骤 3 单击  保存更改。

“**编辑表数据**”页签状态栏显示“**查询提交时间**”获取的行数、操作的执行时间和执行状态。

须知

如果表未定义唯一键并且表中存在重复的行，则对其中重复的一行执行更新操作将更新与之相同的其他所有行。刷新“编辑表数据”页签查看更新的行。

说明

- 一行中未保存的单元格高亮为绿色。保存后，其颜色重置为默认颜色。
- 未保存成功的记录高亮为红色。所有成功以及失败的操作记录个数显示在 4.16.7.8 编辑表数据页签的状态栏。
- 单击“保存”可保存所有有效的更改操作。如果更改无效，则不保存任何内容。请参阅 4.16.7.8 编辑表数据执行保存操作相关设置。

步骤 4 使用  可撤销未保存的更改操作。

步骤 5 用户可在首选项中进行如下设置：

- 要获取的记录数
- 列宽
- 从结果集中复制

  详情请参阅查询结果。

----结束

您可在 Data Studio 中单独编辑新增行的分布键列。

删除

执行如下步骤删除行：

步骤 1 选择要删除行的行标题。

步骤 2 单击  删除行。

步骤 3 单击  保存更改。显示对话框，定义唯一键。

步骤 4 根据需要选择如下选项：

- **“使用所有列”**

  单击**“使用所有列”**将所有列定义为唯一键。
- “自定义唯一键”

  a. 单击**“自定义唯一键”**定义选中的列为唯一键。

  b. 显示**“定义唯一键”**对话框。

  c. 选择需要定义的列，点击**“确定”**。
- 取消

  单击**“取消”**以修改“编辑表数据”页签中的信息。

  “编辑表数据”页签状态栏显示查询提交时间、获取的行数、执行时间和操作的执行状态。

选择“**记住此选择**”可在继续编辑表数据操作的同时隐藏唯一定义窗口。在“编辑表数据”工具栏中单击  以清除先前选定的唯一键定义并再次显示唯一定义窗口。

说明

- 一行中未保存的单元格高亮为绿色。保存后，其颜色重置为默认颜色。
- 未保存成功的记录高亮为红色。所有成功以及失败的操作记录个数显示在 4.16.7.8 编辑表数据页签的状态栏。
- 单击“保存”可保存所有有效的更改操作。如果更改无效，则不保存任何内容。详情请参阅 4.16.7.8 编辑表数据。

步骤 5 使用  可撤销未保存的更改操作。

步骤 6 刷新表数据以查看删除的重复行。

----结束

更新单元格

执行如下步骤更新单元格内容：

步骤 1 双击单元格更新内容。

步骤 2 单击  保存更改。

显示对话框，定义唯一键。

步骤 3 根据用户需要单击如下选项：

- **“使用所有列”**

  单击“使用所有列”将所有列定义为唯一键。

- **“自定义唯一键”**

  a. 单击“**自定义唯一键**”定义选中的列为唯一键。

  b. 显示“**定义唯一键**”对话框。

  c. 选择需要定义的列，点击“**确定**”。

- **“取消”**

  单击“**取消**”修改“编辑表数据”页签中的信息。

  状态栏显示操作的执行时间和执行状态。

  选择“**记住此选择**”可在继续编辑表数据操作的同时隐藏唯一定义窗口。在“**编辑表数据**”工具栏中单击  以清除先前选定的唯一键定义并再次显示唯一定义窗口。

说明

- 一行中未保存的单元格高亮为绿色。保存后，其颜色重置为默认颜色。

- 未保存成功的记录高亮为红色。所有成功以及失败的操作记录个数显示在“编辑表数据”页签的状态栏。
- 单击“保存”可保存所有有效的更改操作。如果更改无效，则不保存任何内容。详情请参阅4.16.7.8 编辑表数据。

步骤 4 使用 撤销未保存的更改操作。

步骤 5 刷新表数据以查看更新后的重复行。

----结束

在 Data Studio 中执行编辑操作时，无法编辑分布键列，因为数据库通过该列在数据库集群中定位数据。

复制

用户可在“**编辑表数据**”页签复制数据。

执行如下操作复制数据：

步骤 1 选择单元格并单击 （复制）或 （高级复制）。

有关复制和高级复制区别的详情，请参阅执行 SQL 查询。

说明

- 用户可复制行号和/或列标题数据。请参阅查询结果设置此首选项。
- 要复制单元格中的部分内容，先选中所需部分，然后按下“Ctrl+C”或单击 。

----结束

粘贴

可从 CSV 文件中复制数据，粘贴到“**编辑表数据**”页签的单元格中。在已有数据的单元格上进行粘贴时，CSV 文件中的新数据会覆盖现有数据。执行如下步骤将内容粘贴到单元格：

步骤 1 从 CSV 文件中复制数据。

步骤 2 选中一个或多个单元格，单击 。

步骤 3 单击 保存更改。显示对话框，定义唯一键。

步骤 4 根据需要，单击如下选项：

- “**使用所有列**”

  单击“**使用所有列**”将所有列定义为唯一键。

- “**自定义唯一键**”

  a. 单击“**自定义唯一键**”，定义选中的列为唯一键。

b. 显示“**定义唯一键**”对话框。

c. 选择需要定义的列，点击“**确定**”。

- “**取消**”

  单击“**取消**”修改“**编辑表数据**”页签中的信息。

  状态栏显示操作的执行时间和执行状态。

  选择“**记住此选择**”可在继续编辑表数据操作的同时隐藏唯一定义窗口。在“**编辑表数据**”工具栏中单击  以清除先前选定的唯一键定义并再次显示唯一定义窗口。

说明

- 从 CSV 文件中复制的单元格数目应和“编辑表数据”页签中选择的单元格数目须匹配。
- 使用  撤销未保存的更改操作。
- 一行中未保存的单元格高亮为绿色。保存后，其颜色重置为默认颜色。
- 未保存成功的记录高亮为红色。所有成功以及失败的操作记录个数显示在“编辑表数据”页签的状态栏。
- 单击“保存”可保存所有有效的更改操作。如果更改无效，则不保存任何内容。详情请参阅 4.16.7.8 编辑表数据。

----结束

粘贴时，无法编辑分布键列，因为数据库通过该列在数据库集群中定位数据。

说明

空单元格显示为“[NULL]”。用户可使用 "Null 值"下拉框搜索“编辑表数据”页签的空单元格。

关于显示/隐藏搜索栏、排序、调整列的顺序和编码的具体信息，请参阅执行 SQL 查询。

4.16.8 编辑临时表

用户可在 Data Studio 中编辑临时表。如果用户在建表时创建了连接，则断开该连接时，临时表会被自动删除。

须知

在 SQL 终端编辑临时表时，请确保启用了连接重用功能。有关如何启用该功能，请参阅 4.20.12 管理 SQL 终端连接。

执行以下步骤编辑临时表：

步骤 1 在临时表上执行查询。

“**结果**”页签会显示 SQL 查询的结果以及执行的查询语句。

步骤 2 “**结果**”页签中编辑临时表。有关如何编辑结果集，请参阅执行 SQL 查询。

----结束

4.17 序列管理

4.17.1 创建序列

按照如下步骤创建序列：

步骤 1 在“**对象浏览器**”窗格，右键某个模式下的“**序列**”，然后选择“**创建序列**”。

Data Studio 弹出“**创建序列**”对话框。

步骤 2 设置相关参数以创建序列。

1. 在“**序列名称**”字段输入序列名称。

说明

勾选“区分大小写”，“序列名称”字段文本区分大小写。例如，输入的序列名称为“Employee”，则序列名称将创建为“Employee”。

2. 在“**最小值**”字段输入最小值。
3. 在“**增量**”字段输入要递增的值。
4. 在“**最大值**”字段输入最大值。

说明

最大和最小值的取值范围应在-9223372036854775808 到 9223372036854775807 之间。

5. 在“**初始值**”字段输入序列的起始值。
6. 在“**缓存数值**”字段输入缓存信息。缓存值表示存储在内存中用于快速访问的数字序列。
7. 勾选“**循环**”，可在序列数达到最大或最小值时进行循环。

说明

该模式名称自动填充到“模式”字段。

8. 在“**表**”中选择对应表。
9. 在“**列**”中选择对应列。

步骤 3 单击“**完成**”。

状态栏显示已完成操作的状态。

说明

可在“SQL 预览”页签自动查看输入数据的 SQL 查询。

----结束

4.17.2 授权/撤销权限

执行以下步骤授权/撤销权限：

步骤 1 右键单击序列组并选择“**授权/撤销权限**”。

弹出“**授权/撤销权限**”对话框。

步骤 2 打开“**选择对象**”页签，选择待授权/撤销权限的对象，并单击“**下一步**”。

步骤 3 打开“**选择权限**”页签，从“**角色**”中选择对应角色。

步骤 4 在“**选择权限**”页签，勾选“**授予**”或“**撤销**”。

步骤 5 在“**选择权限**”页签，勾选或取消勾选相关权限。

在“**SQL 预览**”页签，可以查看根据以上输入自动生成的 SQL 查询。

步骤 6 单击“**完成**”。

----结束

4.17.3 使用序列

本节描述如何使用序列，包含如下内容：

- 删除序列
- 级联删除序列
- 授权/撤销权限

删除序列

可单独或批量删除序列。要进行批量删除，详情请参见 4.21.2 批量删除对象。

按照如下步骤删除序列：

步骤 1 右键单击某序列，然后选择“**删除序列**”。

弹出“**删除序列**”对话框。

步骤 2 单击“**是**”删除该序列。

状态栏显示已完成操作的状态。

----结束

级联删除序列

按照如下步骤级联删除序列：

步骤 1 右键单击某序列，然后选择“**级联删除序列**”。

弹出“**删除序列**”对话框。

步骤 2 单击“**是**”删除该序列。

状态栏显示已完成操作的状态。

----结束

说明

仅 OLAP 支持该特性，OLTP 不支持。

授权/撤销权限

执行以下步骤授权/撤销权限：

步骤 1 右键单击序列并选择“**授权/撤销权限**”。

弹出“**授权/撤销权限**”对话框。

步骤 2 参考 4.17.2 授权/撤销权限来进行授权/撤销权限。

----结束

4.18 视图管理

4.18.1 创建视图

执行以下步骤创建视图：

步骤 1 右键单击视图，选择“**创建视图**”。

“SQL 终端”页签显示视图的 DDL 模板。

步骤 2 编辑该 DDL。

步骤 3 单击  执行 DDL。

步骤 4 按下 F5 刷新“**对象浏览器**”。

“**对象浏览器**”显示新视图。

说明

该操作完成后，状态栏将不显示完成情况。

----结束

4.18.2 授权/撤销权限

执行以下步骤授权/撤销权限：

步骤 1 右键单击视图组并选择“**授权/撤销权限**”。

弹出“**授权/撤销权限**”对话框。

步骤 2 打开“**对象选择**”页签，选择待授权/撤销权限的对象，并单击“**下一步**”。

步骤 3 打开“**选择权限**”页签，从“**角色**”中选择对应角色。

步骤 4 在“**选择权限**”页签，勾选“**授予**”或“**撤销**”。

步骤 5 在“**选择权限**”页签，勾选或取消勾选相关权限。

在“**SQL 预览**”页签，可以查看根据以上输入自动生成的 SQL 查询。

步骤 6 单击“**完成**”。

----结束

4.18.3 使用视图

用户可创建视图，控制对表格特定行或列的访问。视图可基于一个或多个表创建，具体取决于创建该视图的查询语句。

可在已有视图上进行如下操作：

- 导出视图 DDL
- 删除视图
- 级联删除视图
- 重命名视图
- 为视图设置模式
- 查看 DDL
- 为视图中的列设置默认值
- 查看视图属性
- 授权/撤销权限

导出视图 DDL

执行以下步骤导出视图 DDL：

步骤 1 右键单击所选视图，选择“**导出 DDL**”。

Data Studio 显示“**Data Studio 安全免责声明**”对话框。

步骤 2 单击“**确定**”。

Data Studio 显示“**另存为**”对话框。

步骤 3 在“**另存为**”对话框中，选择 DDL 的保存位置，单击“**保存**”。状态栏会显示操作进度。

说明

- 要终止导出操作，双击状态栏，打开“进度视图”页签，单击。
- 如果视图名称包含 Windows 不支持的字符，则导出的文件名称将与视图名称不同。
- 导出视图 DDL 时，可选择多个对象。批量导出章节列举了不支持导出视图 DDL 的对象。

“**导出完成**”对话框和状态栏显示已完成操作的状态。

数据库编码	文件编码	支持导出 DDL
UTF-8	UTF-8	是
	GBK	是
	LATIN1	是
GBK	GBK	是
	UTF-8	是
	LATIN1	否
LATIN1	LATIN1	是
	GBK	否
	UTF-8	是

----结束

删除视图

可单独或批量删除视图。要进行批量删除，详情请参见 4.21.2 批量删除对象。

执行如下步骤删除视图：

步骤 1 右键单击所选视图，选择“**删除**”。

显示“**删除视图**”对话框。

步骤 2 单击“**是**”删除视图。

状态栏显示已完成操作的状态。

----结束

级联删除视图

执行如下步骤删除视图及其下的数据库对象：

步骤 1 右键单击所选视图，选择“**级联删除**”。

显示“**删除视图**”对话框。

步骤 2 单击“**是**”删除视图及其下的数据库对象。

状态栏显示已完成操作的状态。

----结束

重命名视图

执行如下步骤重命名视图：

步骤 1 右键单击所选视图，选择“**重命名**”。

显示“**重命名视图**”对话框。

步骤 2 输入视图名，单击“**确定**”。“**对象浏览器**”显示重命名后的视图。

状态栏显示已完成操作的状态。

----结束

为视图设置模式

执行如下步骤为视图设置模式：

步骤 1 右键单击所选视图，选择“**设置模式**”。

显示“**设置模式**”对话框。

步骤 2 从下拉列表中选择模式，单击“**确定**”。

状态栏显示已完成操作的状态。

如果所选模式包含一个与当前视图名称相同的视图，则不能为当前视图设置该模式。

----结束

查看 DDL

执行如下步骤查看视图的 DDL：

步骤 1 右键单击所选视图，选择“**查看 DDL**”。

视图的 DDL 会显示在新“SQL 终端”页签中。要查看最新 DDL，必须刷新“**对象浏览器**”页签。

----结束

为视图中的列设置默认值

执行如下步骤为视图中的列设置默认值：

步骤 1 右键单击视图下的列名，选择“**设置默认值**”。

Data Studio 弹出对话框显示当前默认值（如果已设置），提示您提供默认值。

步骤 2 输入值并单击“**确定**”。

“**消息**”页签显示操作的状态。

----结束

查看视图属性

执行如下步骤查看视图属性：

步骤 1 右键单击视图并选择“**属性**”。

Data Studio 会在不同选项卡中显示所选视图的属性（“**一般**”和“**列**”）。

说明

如果修改了已打开的视图的属性，刷新并重新打开视图的属性，以在同一窗口中查看更新后的信息。

----结束

授权/撤销权限

执行以下步骤授权/撤销权限：

步骤 1 右键单击视图并选择“**授权/撤销权限**”。

弹出“**授权/撤销权限**”对话框。

步骤 2 参考 4.18.2 授权/撤销权限来进行授权/撤销权限。

----结束

4.19 用户/角色管理

4.19.1 创建用户/角色

多个用户可以使用同一数据库，为了方便管理，需要将用户进行分组。一个数据库角色代表一个或一组数据库用户。

在数据库中，用户和角色的概念类似。实际应用中，建议使用角色来进行权限管理而不是直接访问数据库。

用户 - 数据库用户的集合。这些用户与操作系统用户不同，可以为其他用户分配权限以访问数据库对象。

角色 - 根据用途的不同，可以将角色分为用户或用户组。角色是集群级别的定义，适用于集群中的所有数据库。

4.19.2 管理用户/角色

可以对当前用户/角色执行以下操作：

- 删除用户/角色
- 查看/编辑用户/角色属性
- 查看用户/角色 DDL

删除用户/角色

执行如下步骤删除用户/角色：

步骤 1 右键单击用户/角色，选择“**删除用户/角色**”。

弹出“**删除用户/角色**”对话框。

步骤 2 单击“**是**”。

状态栏显示已完成操作的状态。

----结束

查看/编辑用户/角色属性

执行如下步骤查看用户/角色属性：

步骤 1 右键单击用户/角色，选择“**属性**”。

Data Studio 在不同页签显示所选用户/角色的属性（“通用”，“权限”和“成员属性”）。

用户可对除了“OID”之外的属性进行编辑。

有关编辑、保存、取消、复制和刷新操作的具体信息，请参见 4.16.7.8 编辑表数据。

----结束

查看用户/角色 DDL

执行如下步骤查看用户/角色的 DDL：

步骤 1 右键单击用户/角色，选择“**显示 DDL**”。

用户/角色 DDL 会显示在新的“SQL 终端”页签。刷新“**对象浏览器**”后方可查看最新的 DDL。

----结束

4.20 SQL 终端管理

4.20.1 打开多个“SQL 终端”页签

可在 Data Studio 中打开多个“SQL 终端”页签。通过该功能，可在当前“SQL 终端”页签执行查询时使用多个 SQL 查询。执行如下步骤打开新“SQL 终端”页签：

用户还可在不同连接模板上打开多个“SQL 终端”页签。

步骤 1 在“**对象浏览器**”窗格中，右键单击所需数据库，选择“**打开新的终端**”，或在工具栏中单击  ，或使用快捷键“Ctrl+T”打开新的 SQL 终端。

显示“SQL 终端”页签。

----结束

说明

- Data Studio 中，连接中的每个数据库最多可打开 100 个“SQL 终端”和页签。根据执行的查询次数，每个“SQL 终端”页签都有多个“结果”和一个“消息”页签。如果数据库连接丢失，对应的“SQL 终端”页签仍可使用。
- 恢复操作会恢复所有最小化的 SQL 终端和页签，无法单独恢复某个页签。
- 所有终端关闭后，Data Studio 会重置 SQL 终端的计数器。
- 所有结果集页签关闭后，Data Studio 会重置结果集的计数器。
- Data Studio 支持对以下界面重置计数器：显示 DDL 表空间、显示 DDL 用户/角色、批量删除、结果集和执行计划界面。

Data Studio 在状态栏显示没有对应结果的错误和警告。“**结果**”页签显示执行成功的结果。

执行如下步骤在另一连接中打开新的“SQL 终端”页签：

步骤 1 从工具栏上的连接列表中选择所需连接。

步骤 2 在“**对象浏览器**”窗格中，右键单击连接中的所需数据库，选择“**打开新的终端**”，或在工具栏中单击。系统显示新“SQL 终端”页签。

新“SQL 终端”页签的名称格式如下：

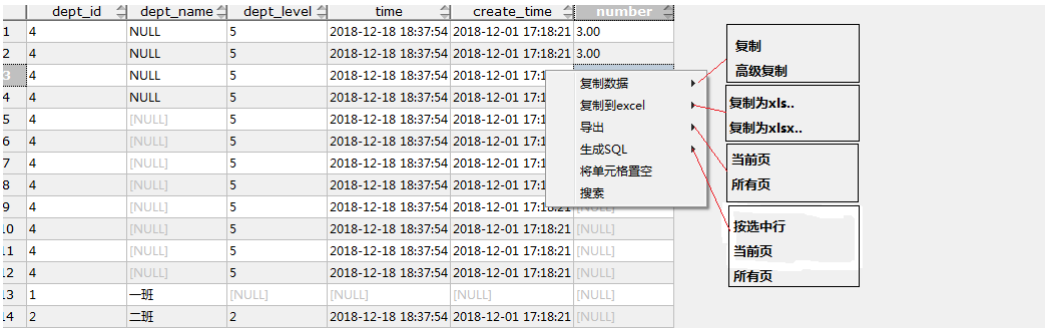
<数据库名>@<连接信息>(<页签编号>)例如：postgres@IDG_1(2)。同一连接信息的每个“SQL 终端”页签的页签编号不同。

----结束

结果窗口的右键选项

此功能允许用户复制/导出单元格数据到 Excel 文件并生成 SQL 查询文件。

在结果窗口中显示 SQL 查询结果之后，右键单击，显示如下菜单：



执行以下步骤为结果集中添加行号及列标题信息：

步骤 1 在 Data Studio 的菜单栏单击“**设置**”。

步骤 2 选择“**首选项**”

步骤 3　展开“**结果管理**”节点并选择“**查询结果**”。

步骤 4　在“**高级复制结果**”区域，勾选“**包含列标题**”和“**包含行号**”。

----结束

下表介绍了右键目录的具体选项。

表4-20 右键目录

目录选项	子项	描述
复制数据	复制	复制选中的单元格数据。
	高级复制	按照首选项设置复制选中单元格数据以及行号、列标题。
复制到 excel	复制为 xls	以 xls 格式导出选中单元格数据，最大支持 64000 行、256 列。
	复制为 xlsx	以 xlsx 格式导出选中单元格数据，最大支持 100 万行。
导出	当前页	导出当前页的表数据。
	所有页	导出全部表内容。
生成 SQL	按选中行	从逻辑插入语句的目标表中选中数据生成 SQL 文件。
	当前页	从逻辑插入语句的目标表中选中当前页数据生成 SQL 文件。
	所有页	从逻辑插入语句的目标表中选中全部表数据生成 SQL 文件。
将单元格置空	-	将单元格数据设为 null。
搜索	-	搜索选中的单元格数据并显示所有满足搜索条件的数据。

说明

以上 SQL 文件的生效范围不包括使用了 JOIN、表达式、视图、SET 运算符、聚合函数、GROUP BY 子句和列别名的查询生成的结果集。

“结果”页签文本视图

此功能允许用户在“**结果**”页签中以文本方式查看数据。

除了网格视图，文本视图下用户还可以进行复制和搜索。

单击  以文本方式获取结果。

说明

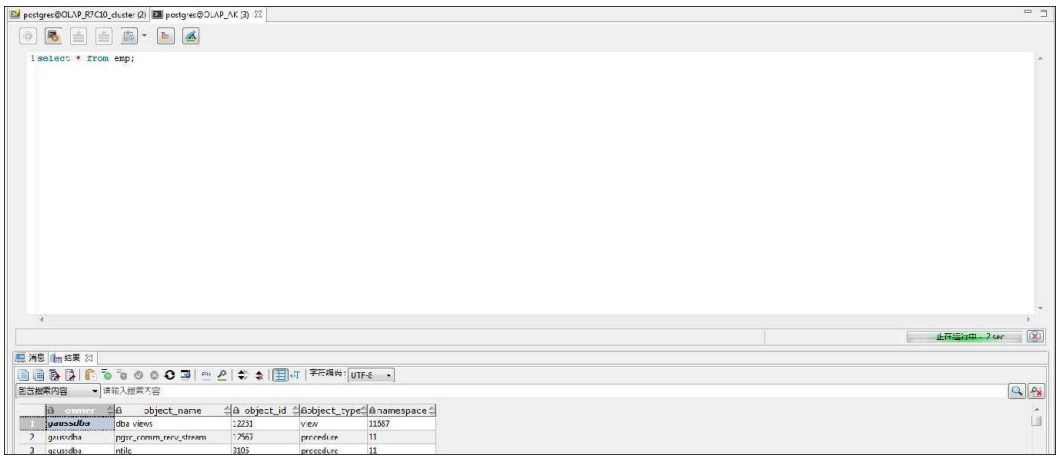
选择多个单元格数据进行搜索可能会导致系统显示一些不正确的文本结果，因为所有信息都需要以纯文本格式复制到搜索窗口。

显示执行进度条

查询在“SQL 终端”窗口执行时，窗口会呈现进度条动态显示运行时间。 查询执行完成后，时间条消失。 时间条旁显示完成查询所用总时间。

如果您想取消查询，可在时间条旁单击“**取消**”。

如下图所示：



说明

- “取消”按钮已从工具栏中删除。
- 在 PL/SQL 编辑器中编译/调试对象时也会显示执行进度条。
- 进度条中显示的时间格式如下：w 时 x 分 y 秒 z 毫秒
- 在 SQL 终端中执行批处理时，进度条会显示查询消耗的总时间。

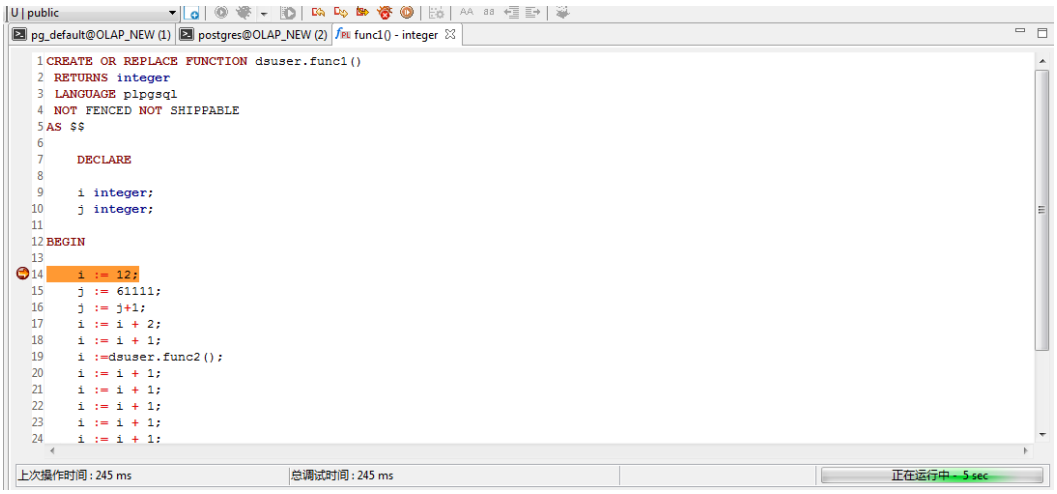
调试时间消耗

Data Studio 在调试期间显示状态栏，显示每个调试操作执行语句的“上次操作时间”和“总调试时间”。

调试期间，根据执行的每个操作，终端状态栏中的“上次操作时间”和“总调试时间”不断更新。 “总调试时间”是“上次操作时间”的总和。

这便于您查找调试对象中耗时长的语句。

如下图所示：



说明

- 调试适用于函数和过程。
- 调试仅适用于 OLAP。

4.20.2 管理 SQL 查询执行历史

可在 Data Studio 中查看和管理频繁执行的 SQL 查询。SQL 查询执行历史只保存在“SQL 终端”页签中。

执行如下步骤查看 SQL 历史。

步骤 1 在“SQL 终端”页签中单击  。

显示“**历史执行 SQL**”对话框。

----结束

说明

SQL 历史脚本未加密。

“**历史执行 SQL**”对话框显示的查询数取决于“**首选项**>**编辑器**>**SQL 历史记录**”中设置的值。有关如何设置 SQL 历史查询数，请参见 SQL 历史记录。查询数超过列表设置的值后，新执行的查询会覆盖较早的 SQL 历史查询。执行的查询会自动存储在列表中。

“**历史执行 SQL**”对话框包含如下列：

- “**锁定状态**”：显示查询的置顶状态。锁定的查询将始终显示在顶部，即使列表已满，也不会从历史记录中删除。
- “**SQL 语句**”：显示 SQL 查询。该列可显示的最大 SQL 查询长度显示的查询数取决于“首选项 > 编辑器 > SQL 历史记录”中设置的值。有关如何修改查询中的字符数，请参见 SQL 历史记录。
- “**获取记录数**”：显示 SQL 查询获取的记录条数。

- “**开始时间**”：显示查询的执行开始时间。
- “**执行时间**”：显示查询的执行用时。
- “**数据库**”：显示数据库名称。
- “**状态**”：显示查询的执行状态，可能为“成功”或“失败”。

删除连接信息时，查询历史会一并删除。如果“**历史执行 SQL**”对话框关闭，查询不会从列表中删除。

在“**历史执行 SQL**”对话框中，可执行如下操作：

- 在“SQL 终端”页签中加载单条 SQL 查询
- 在“SQL 终端”页签中加载多条 SQL 查询
- 删除单条 SQL 查询
- 删除所有 SQL 查询
- 锁定 SQL 查询
- 取消锁定 SQL 查询

在“SQL 终端”页签中加载单条 SQL 查询

执行如下步骤在“SQL 终端”页签中加载单条 SQL 查询：

步骤 1 选择所需查询，单击  。

该查询会添加到“SQL 终端”页签中光标所在位置。

----结束

在“SQL 终端”页签中加载多条 SQL 查询

单击“**加载至 SQL 终端并关闭**”按钮，可在“SQL 终端”页签中加载所选查询，并关闭“**历史执行 SQL**”对话框。

执行如下步骤在“SQL 终端”页签中加载多条所选 SQL 查询：

步骤 1 选择多条所需查询。

步骤 2 单击  。

这些查询会添加到“SQL 终端”页签中光标所在位置。

----结束

说明

如果作业出错时仍继续执行，则终端中的每条语句将作为定时任务依次运行。系统会在控制台上更新执行状态，在进度条中显示作业。当作业执行、进度条更新和控制台更新之间的时间差变得非常细微，则用户无法打开进度条去停止作业执行。此时，必须关闭 SQL 终端才能停止执行。

加载更多记录

如果要在结果页签加载更多数据，下滑鼠标直到页面底部（部分场景操作不便）。目前，Data Studio 提供专门的按钮，使加载更加便捷。

执行如下步骤加载更多记录：

步骤 1 选中目标查询，单击  。

显示所有记录。

----结束

删除单条 SQL 查询

执行如下步骤从“**历史执行 SQL**”列表中删除单条 SQL 查询：

步骤 1 选择所需查询，单击  。

显示确认对话框。

步骤 2 单击“**确定**”。

----结束

删除所有 SQL 查询

执行如下步骤从“**历史执行 SQL**”列表中删除所有 SQL 查询：

步骤 1 单击  。

显示确认对话框。

步骤 2 单击“**确定**”。

----结束

锁定 SQL 查询

如果不希望 Data Studio 从“**历史执行 SQL**”列表中自动删除某些查询，可锁定这些查询。最多可锁定 50 条查询。锁定的查询会显示在列表顶部。在“**SQL 历史记录数**”中设置的值不会影响锁定的查询。有关 SQL 历史记录数的更多信息，请参见 SQL 历史记录。

说明

“历史执行 SQL”窗口关闭后重新打开时，锁定的查询会出现在列表顶部。

执行如下步骤锁定 SQL 查询：

步骤 1 选择所需 SQL 查询，单击  。

“**锁定状态**”列显示查询的锁定状态。

----结束

取消锁定 SQL 查询

执行如下步骤取消锁定 SQL 查询：

步骤 1 选择所需 SQL 查询，单击  。

“**锁定状态**”列显示查询的锁定取消状态。

----结束

4.20.3 打开并保存 SQL 脚本

打开 SQL 脚本

执行以下步骤打开 SQL 脚本：

步骤 1 选择“**文件** > **打开**”。单击工具栏上的“**打开**”按钮，或在 SQL 终端页签单击右键，选择“**打开**”。

如果 SQL 终端页签已有 SQL 脚本，则系统允许用户选择是否进行覆盖或追加至当前 SQL 脚本后。

步骤 2 弹出“**打开**”对话框。

步骤 3 在“**打开**”对话框中，选择要导入的 SQL 文件，单击“**打开**”。

系统以文件终端页签的形式打开该文件。

文件终端页签图标不同于 SQL 终端页签。如果将鼠标悬停在源文件上，则系统会在文件终端上显示相应的数据库连接。

----结束

说明

- SQL 文件编码类型必须匹配首选项中指定的编码类型。
- 如果用户对某文件终端进行了编辑，该页签的图标将以脏标志*字符开头。保存该页签的更改后，该标志会消失。
- 文件终端不支持重命名。一个终端仅对应一个源脚本文件，但一个脚本文件可以在多个终端页签打开。
- 仅在当前终端类型为 SQL 终端时，才能执行打开 SQL 脚本功能。

Data Studio 允许用户在 SQL 终端中打开并保存 SQL 脚本。系统保存更改后，SQL 终端页签自动变更为文件终端页签。

保存 SQL 脚本

“**保存**”选项会自动将文件终端的内容保存到关联文件。

执行以下步骤保存 SQL 脚本：

步骤 1 执行以下任意操作：

- 从主菜单选择“**文件** > **保存**”。
- 按住 Ctrl+S。（该操作支持对文件终端内容进行保存。）
- 单击工具栏上的“**保存**”按钮，或在“SQL 终端”页签中单击右键，选择“**保存**”。

弹出“**Data Studio 安全免责声明** ”对话框。

步骤 2 单击“**确定** ”。

Data Studio 在状态栏显示保存状态。

说明

- Data Studio 将脚本保存为 SQL 文件，并为该文件设置读/写权限。为了保证文件安全，用户必须获取 SQL 文件所在文件夹的读写权限。
- 如果文件有修改或关联的文件不存在，将触发“另存为”操作。
- 在任何情况下，如果源文件保存失败，系统向用户提供“另存为”选项。如果用户选择不另存，则文件终端退回为 SQL 终端。
- 对文件终端的更改不会自动保存。

----结束

将 SQL 脚本另存为新文件

可使用“**另存为**”选项将终端内容保存为新文件。

执行以下步骤另存 SQL 脚本：

步骤 1 执行以下任意操作：

- 选择“**文件** > **另存为**”。
- 按住 Ctrl+Alt+S。（该操作支持对 SQL 终端和文件终端内容进行另存。）

弹出“**Data Studio 安全免责声明**”对话框。

步骤 2 单击“**确定**”。

弹出“**另存为**”对话框。

步骤 3 在“**另存为**”对话框中，选择保存脚本的路径，单击“**保存**”。

----结束

说明

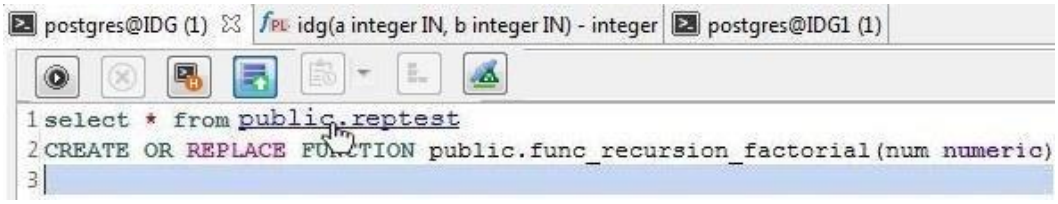
如果文件终端有修改未保存，则系统允许用户在 Data Studio 标准退出时选择保存或取消。

4.20.4 在“SQL 终端”页签中查看表属性和 PL/SQL 函数/过程

可在 Data Studio 中查看表属性和函数/过程。

执行如下步骤查看表属性：

步骤 1 按下 Ctrl 键，同时将光标移动到表名处。



步骤 2 单击突出显示的表名。Data Studio 中显示所选表的属性。

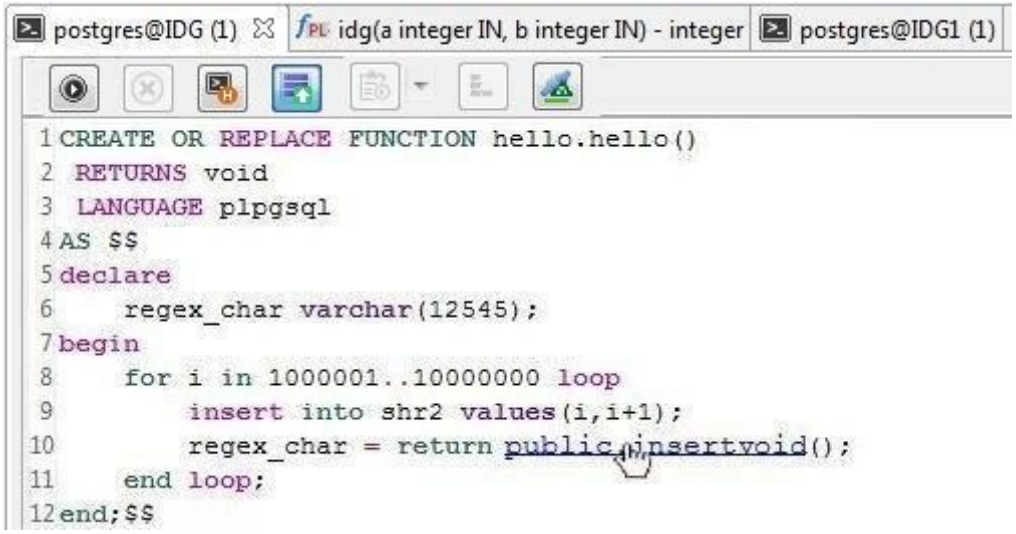
说明

表属性均为只读。

----结束

执行如下步骤查看函数/过程：

步骤 1 按下 Ctrl 键，同时将光标移动到函数/过程名上。



步骤 2 单击突出显示的函数/过程名。Data Studio 在“PL/SQL Viewer”页签中显示所选函数/过程。

----结束

执行以下步骤查看视图属性：

步骤 1 按下 Ctrl 键，同时将光标移至视图名称。

步骤 2 单击高亮的视图名称。Data Studio 会显示所选视图的属性。

----结束

关闭 Data Studio 前保存文件终端

退出前，Data Studio 会提醒用户保存在终端中所作的编辑。

执行以下步骤保存文件终端：

步骤 1 单击 Data Studio 的“**关闭**”按钮，弹出“**退出应用程序**”对话框。

步骤 2 单击“**标准退出**”。

弹出“**保存文件**”对话框，会显示未保存的文件终端。

步骤 3 选择需要保存的文件终端。

步骤 4 单击“**确定**”。

----结束

说明

“强制退出”模式下，不会弹出“保存文件”对话框。

4.20.5 终止正在执行的 SQL 查询

可在 Data Studio 的“SQL 终端”页签中终止正在执行的 SQL 查询。

执行以下步骤终止正在执行的 SQL 查询：

步骤 1 在“SQL 终端”页签中选择 SQL 查询并执行。

步骤 2 在“SQL 终端”页签中单击  或按 Shift+Esc。

也可在菜单栏中选择“执行 > 取消”，或在“SQL 终端”页签中单击右键，在右键菜单中选择“**取消**”，或在“**进度视图**”页签中单击“**取消**”。

----结束

终止正在执行的查询后，正在执行的 SQL 语句会停止执行。

如果某查询被终止，该查询对数据库所做的变更会被撤销。该查询后的查询不会执行。

以下情况下，查询无法终止，“**结果**”页签显示该查询的执行结果：

1. 服务器已执行完毕该查询，正在准备生成结果。
2. 服务器已执行完毕该查询，查询结果正在从服务器传送至 Data Studio 客户端。

查看某查询的执行计划时，该查询无法终止。有关详情，请参见 4.20.8 查看执行计划和开销。

“**消息**”页签显示查询终止消息。

说明

“取消”菜单项仅在查询执行期间可用。

4.20.6 SQL 查询格式化

Data studio 支持 SQL 和 PL/SQL 语句格式化在“SQL 终端”中一起高亮显示。

PL/SQL 格式化

执行如下步骤格式化 PL/SQL 语句：

步骤 1 选择需要格式化的 PL/SQL 语句。

步骤 2 在工具栏中点击  ，格式化查询。

按“Ctrl+Shift+F”或在主菜单中选择“编辑 > 格式化”。

PL/SQL 语句被格式化。

----结束

SQL 格式化

Data Studio 支持对语法正确的简单 SQL 语句（包括 SELECT，INSERT，UPDATE，和 DELETE）进行格式化。以下列举了该语句须满足的一些条件。

1. SELECT 语句必须包含以下子句：
   - Target list
   - From（包括 join）
   - Where
   - Group by
   - Having
   - Order by
   - Common table expression

   不包含 SET 操作，如 UNION、UNION ALL、MINUS、INTERSECT 等。

   不包含子查询。
2. 仅包含以下子句的 INSERT 语句：
   - Insert Into Table name
   - Values
   - Values Column List
   - RETURNING
3. 仅包含以下子句的 UPDATE 语句：
   - Update Table name
   - SET
   - From (包括 join)
   - Where
   - RETURNING
4. 仅包含如下子句的 DELETE 语句：
   - Delete From Table name

- Using (包括 join)
- Where
- RETURNING

执行以下步骤对 SQL 查询进行格式化：

步骤 1 选择需要格式化的 SQL 查询。

步骤 2 在工具栏中点击  ，格式化查询。

按“Ctrl+Shift+F”或在主菜单中选择“编辑 > 格式化”。

查询被格式化。

下表描述了查询格式化的具体规则 。

表4-21 查询格式化规则

语句	子句	格式化规则
SELECT	SELECT list	第一列之前插入换行符
		缩进列表中的列
	FROM	FROM 之前插入换行符
		FROM 之后插入换行符
		缩进 FROM list
		堆叠 FROM list
	JOIN（FROM 子句）	JOIN 之前插入换行符
		JOIN 之后插入换行符
		ON 之前插入换行符
		ON 之后插入换行符
		JOIN 之后缩进表
		缩进 ON 条件
	WHERE	WHERE 之前插入换行符
		WHERE 之后插入换行符
		缩进 WHERE 条件
		将 WHERE 条件放在同一行
	GROUP BY	GROUP 之前插入换行符
		GROUP BY expression 之前插入换行符
		缩进列表中的列

语句	子句	格式化规则
		堆叠列表中的列
	HAVING	HAVING 之前插入换行符
		HAVING 之后插入换行符
		缩进 HAVING 条件
	ORDER BY	ORDER 之前插入换行符
		BY 之后插入换行符
		缩进列表中的列
		堆叠列表中的列
	CTE	缩进子查询括号
		每个 CTE 占一行
INSERT	INSERT INFO	左大括号前插入换行符
		左大括号后插入换行符
		右大括号前插入换行符
		缩进列表列的大括号
		缩进列表中的列
		VALUES 之前插入换行符
		堆叠列表中的列
		VALUES 之前插入换行符
		左大括号前插入换行符
		左大括号后插入换行符
		右大括号前插入换行符
		缩进 VALUES 表达式列表的大括号
		缩进 VALUES 表达式列表
		堆叠 VALUES 表达式列表
	DEFAULT	DEFAULT 前插入换行符
		缩进 DEFAULT 关键字
	CTE	每个 CTE 占一行
	RETURNING	RETURNING 前插入换行符
		RETURNING 后插入换行符

语句	子句	格式化规则
		缩进 RETURNING 列表中的列
		将 RETURNING 列表中的列放在单行上
UPDATE	UPDATE Table	表前插入换行符
		缩进表
	SET Clause	SET 前插入换行符
		缩进列分配列表中的列
		缩进列分配列表中的列
	FROM CLAUSE	FROM 前插入换行符
		FROM 后插入换行符
		缩进 FROM 列表
		堆叠 FROM 列表
	JOIN CLAUSE(FROM CLAUSE)	JOIN 前插入换行符
		JOIN 后插入换行符
		ON 前插入换行符
		ON 后插入换行符
		JOIN 后缩进表
		缩进 ON 条件
	WHERE CLAUSE	WHERE 前插入换行符
		WHERE 后插入换行符
		缩进 WHERE 条件
		缩进 WHERE 条件
	CTE	每个 CTE 占一行
	RETURNING	RETURNING 前插入换行符
		RETURNING 后插入换行符
DELETE	USING CLAUSE	缩进 RETURNING 列表中的列
		FROM 前插入换行符
		FROM 后插入换行符
		缩进 USING 列表
		堆叠 FROM 列表

语句	子句	格式化规则
	JOIN CLAUSE	JOIN 前插入换行符
		JOIN 后插入换行符
		ON 前插入换行符
		ON 后插入换行符
		JOIN 后缩进表
		缩进 ON 条件列表
	WHERE CLAUSE	WHERE 前插入换行符
		WHERE 后插入换行符
		缩进 WHERE 条件
		堆叠 WHERE 条件列表
	CTE	每个 CTE 占一行
	RETURNING	RETURNING 前插入换行符
		RETURNING 后插入换行符
		缩进 RETURNING 列表中的列

----结束

将光标放在某个标点符号前后或选中该标点符号，Data Studio 会自动高亮显示成对的标点符号，如下所示：

- 小括号：( )
- 中括号：[ ]
- 大括号：{ }
- 单引号（字符串文字）：' '
- 双引号（字符串文字）：" "

使用如下方法可在“SQL 终端”页签中修改 SQL 查询和 PL/SQL 语句的大小写：

方法 1：

步骤 1 选中要修改的文本，选择“编辑 > 大写/小写”。

文本转变为所选大小写。

----结束

方法 2：

步骤 1 选中要修改的文本，在工具栏中单击 AA 或 aa 。

文本转变为所选大小写。

----结束

方法 3：

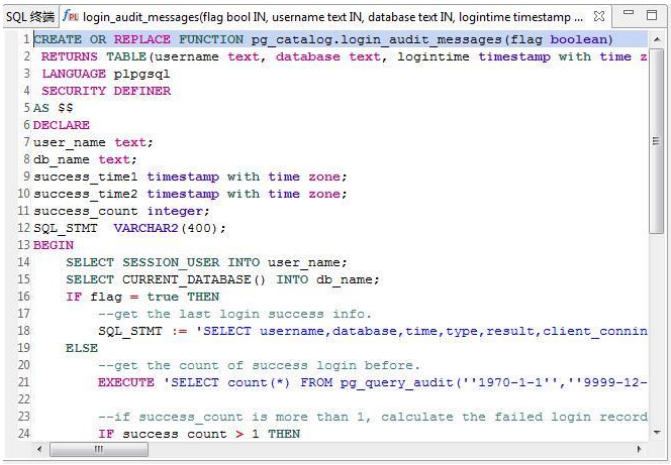
步骤 1 选中要修改的文本，按下“Ctrl+Shift+U”将其转变为大写，或按下“Ctrl+Shift+L”将其转变为小写。

文本转变为所选大小写。

----结束

SQL 高亮显示

输入时关键字自动高亮显示如下（按照默认高亮方案显示）：



语法类型颜色，如下图所示：

语法类别	颜色代码	HEX值	颜色
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	

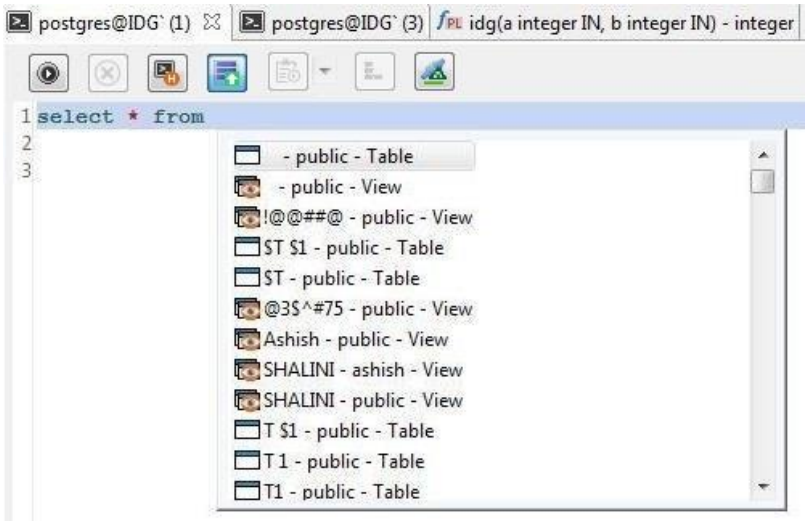
还可为特定类型的语法自定义 SQL 高亮方案。有关详情，请参见语法高亮。

4.20.7 在“SQL 终端”页签中选择数据库对象

Data Studio 在“SQL 终端”页签中显示建议列表，提供建议的模式、表、列和视图名。

执行以下步骤选择数据库对象：

步骤 1 按下“Ctrl+空格”，输入所需数据库父对象名。列表内容会随输入的数据库对象名进行调整。数据库对象列表显示连接到“SQL 终端”页签的数据库的所有对象。



步骤 2 按下键盘上、下键移动至要选择的数据库父对象，然后按下回车键或双击选择数据库父对象。

步骤 3 按下“.”键列出所有数据库子对象。

步骤 4 按下键盘上、下键移动至要选择的数据库子对象，然后按下回车键或双击选择数据库子对象。

选择后，数据库子对象会添加至数据库父对象后（带英文句号“.”）。

说明

- 自动建议也适用于所有用户有权访问的模式对象的关键字、数据类型、模式名、表名、视图和表别名，方式同上。

  以下查询示例中指定了别名对象：

```
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>
```

- 如下场景下，自动建议可能会在终端显示“正在加载”：
- 当对象数量超过“加载上限”字段中出现的值时，不会加载这些对象。详情请参见 4.8.2 添加连接。
- 当要加载的对象已经添加在“不包含”列表项中时，不会加载这些对象。详情请参见 4.8.2 添加连接。
- 从服务器上获取到对象时存在时延。
- 如果不同场景下存在名称相同的对象，则自动建议时会同时显示这些对象的子对象。

  例如：

  如果有两个模式分别名为 public 和 PUBLIC，则会显示这两个模式的所有子对象。

----结束

4.20.8 查看执行计划和开销

执行计划显示如何对指代 SQL 语句的表格进行扫描，分为次序扫描和索引扫描。

SQL 语句执行成本为预估的查询时间（查询的语句成本单位是随机的，通常情况下检查对象为磁盘页）。

查看 SQL 查询的计划和成本，可通过以下方式：

步骤 1 在“SQL 终端”中输入查询或使用已有查询，单击工具栏的  来查看解释计划。

若要查看使用了 Analyze 的解释计划，单击查询按钮，选择“**包含 ANALYZE 结果**”。然后再次单击查询。

“执行计划”默认在底部的新页签以树形图样式显示。显示支持树形样式和文本样式。

说明

树形执行计划和 Visual Explain 中显示的数据可能会有所不同，因为二者执行的参数不同。

下表展示了使用和未使用 Analyze 解释计划时所选择的参数和显示的列：

表4-22 解释计划选项

解释计划类型	参数	列
未选择“包含 ANALYZE 结果”（默认值）	Verbose、Costs	Node type、startup cost、total cost、rows、width、additional Info
选择“包含 ANALYZE 结果”	Analyze、Verbose、Costs、Buffers、Timing	Node type、startup cost、total cost、rows、width、Actual startup time、Actual total time、Actual Rows、Actual

解释计划类型	参数	列
		loops、Additional Info

Additional Info 列包括谓词信息（过滤谓词和 hash 条件）、分布键、输出信息以及节点类型信息。

树形样式将节点划分为 16 个类型。在树形样式中，每个节点都将以相应类型的图标开头。下表列举了节点类别及相应图标。

表4-23 节点类别和图标

节点类别	图标
Aggregate	
Group Aggregate	
Function	
Hash	
Hash Join	
Nested Loop	
Nested Loop Join	
Modify Table	
Partition Iterator	
Row Adapter	
Seq Scan on	
Set Operator	
Sort	
Stream	
Union	
Unknown	

将鼠标悬停在突出显示的单元格上，可以识别负载最重、开销最大、速度最慢的节点。只有树形样式支持单元格突出显示。

若选择了多个查询，则仅针对最后一个查询显示使用/未使用 Analyze 的解释计划。

每次执行一个执行计划，该计划都会在新页签中打开。

如果丢失连接但对象浏览器中依然保持数据库的连接，会弹出“**连接错误**”对话框，显示以下内容：

- “**是**”：重建连接，获取查询计划及开销。
- “**否**”：断开对象浏览器中的数据库连接。

“**执行计划**”窗口中的工具栏菜单选项如下：

工具栏菜单	图标	描述
树形样式		此图标用于以树形样式查看解释计划。
文本样式		此图标用于以文本样式查看解释计划。
复制		此图标用于将所选内容从结果窗口复制到剪贴板，支持快捷键 Ctrl+C。
保存		此图标用于以文本样式保存解释计划。

有关刷新、SQL 预览和搜索栏的具体信息，请参见执行 SQL 查询。

刷新之后会重新执行 Explain/Analyze 查询并刷新当前页签显示的计划内容。

结果显示在“**消息**”页签中。

----结束

4.20.9 图形化查看执行计划和开销

Visual Explain 计划从扩展 JSON 格式中获取信息，以图形化方式显示 SQL 查询。这有助于优化查询以增强查询和服务器性能，有助于分析数据库所用的查询路径，并找出最拥挤，开销最高和运行最慢的节点。

图形化执行计划展示了 SQL 语句所引用的表是如何被扫描的（普通顺序扫描和索引扫描）。

SQL 语句的执行开销取决于其运行时长（可使用任意开销单位度量，但是通常以磁盘中每页的抓取数计算。）

Costliest：“Self Cost”最高的计划节点。

Heaviest： 输出行数最大的计划节点被认为是最拥挤的计划节点。

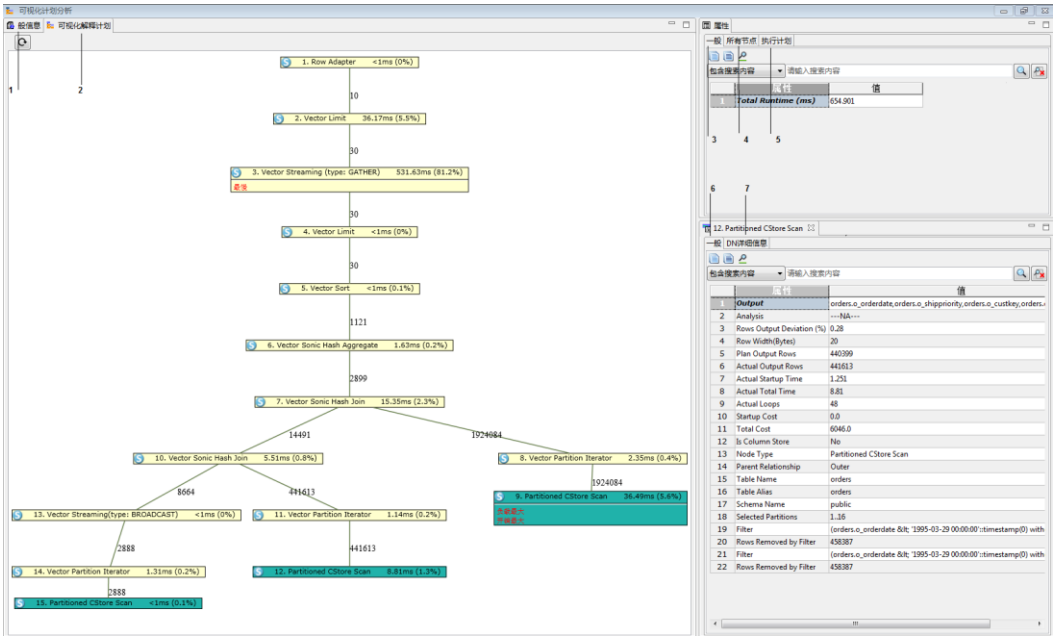
Slowest：所需执行时间最长的计划节点。

按照以下步骤以图表形式查看所需 SQL 查询的计划和成本：

步骤 1 在“SQL 终端”页签输入查询或使用现有查询，然后单击工具栏中的  。

显示“**可视化计划分析**”窗口。

如果在获取执行计划和开销的过程中丢失连接，请参见 4.20.8 查看执行计划和开销获取关于重新连接选项的具体信息。



- 1 - “**一般信息**”：显示查询详情。
- 2 - “**可视化解释计划**”：以图形化方式显示所有节点，如开销最高、最拥挤的和最慢的计划节点。单击每个计划节点可查看节点详情。
- 3 - “**属性 - 一般**”：以毫秒为单位提供查询的执行时间。
- 4 - “**属性 - 所有节点**”：提供所有节点信息。

字段名	说明
Node Name	显示节点名称。
Analysis	显示各节点的分析信息。
RowsOutput	显示计划节点输出的行数。
RowsOutput Deviation (%)	显示预估的输出行数和计划节点实际的输出行数之间的偏差百分比。
Execution Time (ms)	显示查询在计划节点上的执行时间。
Contribution (%)	显示查询在计划节点上的执行时间占整个查询执行时间的百分比。
Self Cost	显示在计划节点上执行查询的“Total Cost”，即为所有子节点的总开销。
Total Cost	显示在计划节点上执行查询所消耗的总成本。

- 5 - “属性 - 执行计划”：显示所有节点上的执行信息。

列名	说明
节点名称	节点名称
实体名称	对象名称
开销	计划节点执行时间
行	计划节点输出行数
循环	每个节点执行的循环数
宽度	计划节点输出行的平均宽度估算值（以字节为单位）
实际行数	计划节点输出行数的估算值
实际时间	计划节点的实际执行时间

- 6 - “计划节点 - 一般”：显示各节点信息。

行名	描述
Output	显示计划节点返回的字段信息。
Analysis	显示开销最大、最慢和最拥挤的计划节点分析信息。
RowsOutput Deviation (%)	显示预估的输出行数和计划节点的实际输出行之间的偏差百分比。
Row Width (bytes)	显示预估的计划节点输出行的平均宽度（单位：字节）。
Plan Output Rows	显示计划节点输出的行数。
Actual Output Rows	显示预估的计划节点输出行数。
Actual Startup Time	计划节点生成第一条记录所耗费的执行时间。
Actual Total Time	显示计划节点的实际执行时间。
Actual Loops	显示该节点执行的迭代数。
Startup Cost	显示计划节点输出第一条记录所耗费的时间。
Total Cost	显示查询在计划节点上的执行时间。
Is Column Store	表示表的存储方式（列或行存储）。

行名	描述
Shared Hit Blocks	显示缓存命中的共享块数量。
Shared Read Blocks	显示从缓存读取的共享块数量。
Shared Dirtied Blocks	显示缓存中弄脏的共享块数量。
Shared Written Blocks	显示缓存写入的共享块数量。
Local Hit Blocks	显示缓存命中的局部块数量。
Local Read Blocks	显示从缓存读取的局部块数量。
Local Dirtied Blocks	显示缓存中弄脏的局部块数量。
Local Written Blocks	显示缓存写入的局部块数量。
Temp Read Blocks	显示从缓存读取的临时块数量。
Temp Written Blocks	显示缓存写入的临时块数量。
I/O Read Time (ms)	显示该节点执行任意 I/O 读操作的耗时。
I/O Write Time (ms)	显示该节点执行任意 I/O 写操作的耗时。
Node Type	显示计划节点的类型。
Parent Relationship	显示与父节点的关系。
Inner Node Name	子节点名称
Node/s	无需输入，将从属性中移除。

根据计划节点类型可以显示其他信息。举例如下:

计划节点	其他
Partitioned CStore Scan	表名、表别名和模式名
Vector Sort	排序键
Vector Hash Aggregate	分组键
Vector Has Join	Join 类型和 Hash 条件
Vector Streaming	分布键和 Spawn On

- 7 - **“计划节点 - DN 详情”**：为每个节点提供详细的 DN 信息。只有在从 DN 收集数据时，**“DN 详情”**才可用。

  有关复制和搜索工具栏选项的说明，请参阅 4.16.7.7 查看表数据。

----结束

4.20.10 使用 SQL 终端

在“SQL 终端”页签中，可进行如下操作：

- 自动提交
- 执行 SQL 查询
- 多列排序
- 备份未保存的查询/函数/过程
- 错误定位
- 在“PL/SQL Viewer”或“SQL 终端”页签中进行搜索
- 在“PL/SQL Viewer”或“SQL 终端”中定位到某行
- 注释/取消注释
- 缩进/取消缩进行
- 插入空格
- 执行多条函数/过程或查询
- 重命名 SQL 终端
- SQL 助手
- 使用模板

自动提交

您可在“首选项”窗口启用或禁用“自动提交”功能，具体参见事务。

- 若启用“自动提交”功能，“提交”和“回滚”会被禁用，事务将会被自动提交。
- 若禁用“自动提交”功能，“提交”和“回滚”会被启用，可手动提交或回滚事务。

说明

- 对于 OLAP，服务器会为所有 SQL 语句，如 select，explain select，及 set parameter，打开事务。
- 对于 OLTP，服务器仅为 DML 语句，如 INSERT, UPDATE, 及 DELETE，打开事务。

重用连接

“重用连接”允许用户为结果集选择相同的 SQL 终端连接或新连接。根据数据库服务器中定义的隔离级别，该选择会影响记录的可见性。

- 如果“重用连接”打开，可以通过终端连接进行数据操作、刷新结果窗口。

对于某些数据库，可以在结果窗口中编辑由终端创建或使用的临时表。

- 如果“重用连接”关闭，将使用新连接进行数据操作、刷新结果窗口。

对于某些数据库，可以在结果窗口中编辑临时表。

 ：“重用连接”打开时显示该图标。

 ：“重用连接”关闭时显示该图标。

 ：“重用连接”被禁用时显示该图标。

执行以下步骤关闭“重用连接”：

步骤 1 在“SQL 终端”工具栏，单击  。

该终端的重用连接功能关闭，显示  。

说明

- “重用连接”功能默认启用，用户可自行关闭。如果用户启用“自动提交”，系统会自动启用“重用连接”功能。
- 如果用户禁用“自动提交”，系统会自动禁用“重用连接”功能，但该功能在界面上仍显示为启用状态，且不允许用户修改。

----结束

有关“自动提交”和“重用连接”的更多信息，请参见表 4-28。

执行 SQL 查询

执行如下步骤，执行函数/过程或 SQL 查询：

在“SQL 终端”页签输入函数/过程或 SQL 语句，单击“SQL 终端”页签的；或按“Ctrl+Enter”；或从菜单栏选择“运行 > 编译/执行声明”。

另一种执行 SQL 语句的方法为，在“SQL 终端”页签，右击选择“执行语句”。

说明

可从状态栏查看正被执行的查询的状态。

执行函数/过程或 SQL 查询和查询语句后，生成结果，并显示在“结果”页签中。

如果在查询执行期间丢失连接但对象浏览器中依然保持数据库的连接，会弹出“连接错误”对话框，显示以下内容：

- “重新连接”：重新建立连接。
- “重新连接并执行”：如果打开了自动提交功能，执行将从失败语句开始继续。如果关闭了自动提交功能，执行将从光标所在位置开始继续。
- “取消”：在对象浏览器中断开数据库连接。

如果三次尝试后依然无法重新连接，将断开对象浏览器中数据库的连接。之后，在对象浏览器中建立数据库的连接，并重试查询。

说明

- 对于运行时间长的查询，只有在获取完整结果后才能编辑结果集。
- 以下场景的查询结果可编辑：

- 选择的对象来自同一个表
- 选择了所有列或部分列（不存在别名、聚合函数或有关列的表达式）
- 查询中含有 WHERE 条件
- 查询中含有 ORDER BY 子句
- 普通、分区和临时表
- 如果提交空行，将为其所有列分配 Null 值。
- 对象浏览器上可用表的查询结果集是可编辑的。
- 在 SQL 终端上执行查询的结果是可编辑的。

可通过“设置 > 首选项”设置列宽。详情请参见“查询结果”小节。

调整列位置

可单击选中的列标题，并将其拖动到需要的位置。

多列排序

此功能支持用户对某些界面的表数据按照多列进行排序，还可以设置多个排序列的排序优先级。

多列排序适用的界面包括：

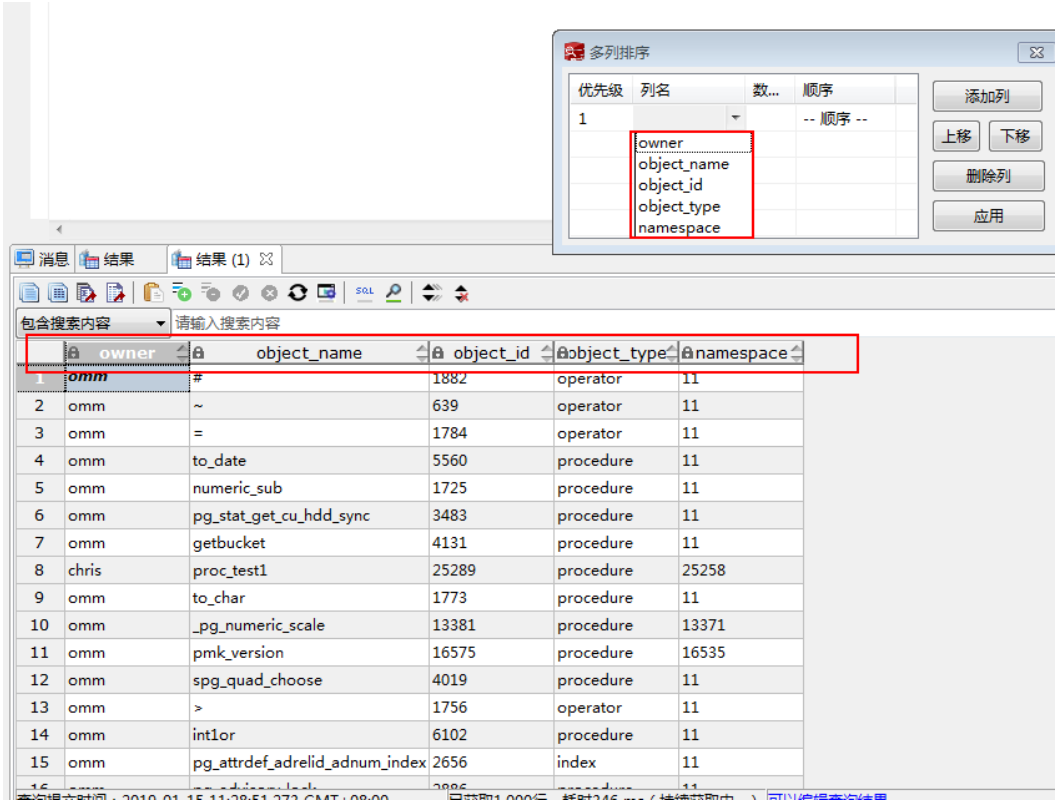
- 结果集页签
- 编辑表数据窗口
- 查看表数据窗口
- 批量表删除窗口

执行以下步骤打开多列排序：

步骤 1 单击工具栏的  。

弹出“**多列排序**”对话框。

步骤 2 单击“**添加列**”，从下拉列表中选择目标排序列。



步骤 3　按照需要选择顺序。

步骤 4　单击“**应用**”。

----结束

多列排序对话框包含以下元素：

表4-24 多列排序对话框界面元素

名称	界面元素类型	描述/操作
优先级	只读文本字段	显示相应列在多列排序时的优先级。
列名	组合字段，可选值包括表的所有列名称	显示为排序添加的列的名称。
数据类型	只读文本字段	显示相应列的数据类型。
顺序	组合字段，可选值包括升序和降序	显示相应列的排序顺序。
添加列	按钮	为多列排序表添加新列。
删除列	按钮	从多列排序表删除选中列。
上移	按钮	将选中列上移一步，调整

名称	界面元素类型	描述/操作
		排序优先级。
下移	按钮	将选中列下移一步，调整排序优先级。
应用	按钮	应用排序设置。

说明

除以下数据类型外，所有其他数据类型都按照字母顺序进行排序：

TINYINT、SMALLINT、INTEGER、BIGINT、FLOAT、REAL、DOUBLE、NUMERIC、BIT、BOOLEAN、DATE、TIME、TIME_WITH_TIMEZONE、TIMESTAMP、TIMESTAMP_WITH_TIMEZONE

多列排序还提供以下图标：

表4-25 多列排序图标

图标	描述	操作
	不排序	如果列标题显示此图标，表示该列未排序。单击此图标，会对该列数据按升序排序。 此外，还可以使用快捷键ALT+单击（列名称）。
	按升序排序	如果列标题显示此图标，表示该列已按照升序排序。单击此图标，会对该列数据按降序重新排序。 此外，还可以使用快捷键ALT+单击（列名称）。
	按降序排序	如果列标题显示此图标，表示该列已按照降序排序。单击此图标，会取消对该列的排序。 此外，还可以使用快捷键ALT+单击（列名称）。

以下图标显示排序优先级：

 ：该图标包含 3 个黑点，优先级最高。

 ：该图标包含 2 个黑点，优先级次之。

 ：该图标包含 1 个黑点，优先级第三或更低。

表4-26 工具栏按钮

名称	图标	描述
复制		用于将数据从结果窗口复制到剪切板。快捷键为 Ctrl+C。
高级复制		用于将数据从结果窗口复制到剪切板，高级复制会复制列标题。请参阅查询结果设置此首选项。快捷键为 Ctrl+Shift+C。
导出所有数据		用于将所有数据导出到 xcel（xlsx/xls）、CSV、文本或二进制文件中。有关详情，请参见 4.16.7.4 导出表数据。 说明 • 查询中提到的列将自动填充在“选定的列”区域中。“可用列”区域为空。 • 要导出查询结果，会使用新连接重新执行查询。导出的结果可能与结果选项卡中的数据不同。 • 该功能对于解释/分析查询不可用。要导出解释/分析查询，请使用“导出当前页数据”选项。
导出当前页数据		用于将当前页数据导出到 Execl（xlsx/xls）或 CSV 文件中。
粘贴到表格		用于粘贴复制的信息。详情请参阅粘贴。
添加		用于向结果集中添加行。详情请参阅插入。
删除		用于从结果集中删除行。详情请参阅删除。
保存更改		用于保存结果集中所做的更改。详情请参阅 4.16.7.8 编辑表数据。
取消更改		用于回滚对结果集所做的更改。详情请参阅 4.16.7.8 编辑表数据。
刷新		用于刷新结果集信息。如果针对同一个

名称	图标	描述
		表打开多个结果集，刷新后，对一个结果集所做的更改也会体现在其他结果集中。同样，如果对表格进行了编辑，则结果集将在刷新后再次更新。
清除已保存的唯一键选项		用于清除先前选择的唯一键。详情请参阅 4.16.7.8 编辑表数据。
显示/隐藏查询面板		用于显示/隐藏为特定结果集执行的查询。此为状态开关按钮。
显示/隐藏搜索面板		用于显示/隐藏搜索文本字段。此为状态开关按钮。
编码		用户能否设置该字段，取决于“首选项 > 结果管理 > 查询结果 > 结果数据编码”中的设置。在该下拉列表中，用户可选择适当的编码以准确查看数据。默认编码为 UTF-8。请参阅结果数据编码设置编码首选项。 **说明** **修改默认编码后，除数据插入之外的其他数据编辑操作将受到限制。**
多列排序		用于弹出多列排序对话框。
清除排序		用于重置所有已排序的列。

搜索字段中的图标：

图标名称	图标	描述
搜索		用于根据定义的条件，搜索结果集。搜索内容不区分大小写。
清空搜索内容		用于清空搜索字段中输入的搜索内容。

“**结果**”页签的右键菜单选项如下：

选项	描述
关闭	仅关闭当前结果窗口。
关闭其它	关闭除当前结果窗口外的其他所有结果窗口。
关闭右侧窗口	关闭当前结果窗口右侧的所有结果窗口。

选项	描述
关闭所有	关闭所有结果窗口，包括当前结果窗口。
分离窗口	单独打开当前结果窗口。

以下为“**结果**”窗口中显示的状态信息：

- “**查询提交时间**”：提交时间。
- 获取的行数和执行时间同时显示，且会显示默认行数。如果有其他记录待获取，此图标将显示为“更多”。用户可将光标滚至表底部读取并显示所有行。

须知

用户在查看表数据时，Data Studio 会自动调整列宽度以获得良好的视觉效果。用户可根据需要重新设置列的宽度。如果单元格的文本内容超过整个可显示区域，重新调整单元格宽度可能会导致 Data Studio 无响应。

说明

- 执行一次查询打开一个新的结果窗口。要在新窗口中查看结果，需选中新打开的窗口。
- 将 focusOnFirstResult 参数设为 false，可自动定位到最新打开的“结果”页签。有关详情，请参见 4.2 安装配置 Data Studio。
- 可从结果集中复制行、列、所选单元格。
- 删除连接后，所有数据仍会成功导出。
- 如果列的内容包含空格，会在空格处自动断行以适应该列的显示区域。不包含空格的内容不会自动断行。
- 要复制单元格中的部分内容，先选中所需部分，然后按下“Ctrl+C”或单击  。
- 列的大小取决于内容最长的列的长度。
- 用户可根据喜好保存首选项以定义：
- 要获取的记录数
- 列宽
- 从结果集中复制选项

  详情请参见查询结果。
- 如果 resultset 页签中的任何一列包含锁图标，参数值将无法编辑。

备份未保存的查询/函数/过程

Data Studio 根据“首选项”页签中定义的时间间隔对“SQL 终端”和“PL/SQL Viewer”中未保存的数据进行定期备份。Data Studio 会根据“首选项”设置对数据进

行加密和保存。请参见备份查询/函数/过程来打开/关闭备份功能，定义数据保存间隔，并加密保存的数据。

“SQL 终端”和“PL/SQL Viewer”中未保存的更改作为备份保存在 DataStudio\UserData\<username>\Autosave 文件夹中。如果在 Data Studio 意外关闭之前已经保存了这些备份文件，在下次登录时这些文件全部可用。

在标准退出时，如果“SQL 终端”和“PL/SQL Viewer”中存在未保存的数据，Data Studio 将等待备份完成之后才关闭。

错误定位

查询/函数/过程的执行过程中，如果出错，会显示错误定位消息：

“**是**”： 单击“是”继续执行。

“**否**”：单击“否”停止执行。

可选择“不显示此次执行中发生的其他错误。”选项隐藏报错消息并继续执行当前 SQL 查询。

错误消息的行号和位置显示在“**消息**”页签。在“终端”或“PL/SQL Viewer”页签中，对应的行编号在出错位置用  和红色下划线标记出来。要查看错误消息，可将光标悬停在  上。如有行号和错误详情不匹配，请参见 4.26 FAQs。

说明

如果查询/函数/过程在执行过程中被修改，错误定位器可能无法显示正确的行和位置编号。

在“PL/SQL Viewer”或“SQL 终端”页签中进行搜索

执行如下步骤，在“PL/SQL Viewer”或“SQL 终端”页签中进行搜索：

“**F3**”用于搜索下一处，“Shift+F3”用于搜索上一处。这些快捷键在按下“Ctrl+F”进行文本搜索后，搜索关键词期间可用，直到搜索完毕。“Ctrl+F”、“F3”、“Shift+F3”仅限于在当前实例内搜索关键词。

步骤 1 在主菜单中选择“编辑 > 查找和替换”。

也可按下“Ctrl+F”。

系统显示“查找和替换”对话框。

步骤 2 在“查找内容”字段中输入要搜索的文本，单击“查找下一处”按钮。

搜索的文本高亮显示。

可使用“F3”或“Shift+F3”向前或向后搜索。

说明

搜索到 SQL 查询或 PL/SQL 语句的最后一行后，选择“从当前位置搜索”选项继续搜索。

----结束

在“PL/SQL Viewer”或“SQL 终端”中定位到某行

执行如下步骤在“PL/SQL Viewer”或“SQL 终端”中定位到某行：

“**转到行**”选项用于直接跳转到终端内某行。

步骤 1 选择“编辑 > 转到行”或按下“Ctrl+G”。

系统显示“转到行”对话框。

步骤 2 在“请输入行号”字段中输入所需行号，单击“确定”。

光标会移动到指定行的行首。

说明

如下输入对该字段无效：

- 非数字
- 特殊符号
- 编辑器中不存在的行号
- 超过 10 位的数字

----结束

注释/取消注释

注释/取消注释选项用于注释或取消注释行或整段。

按照以下步骤在“PL/SQL Viewer”或“SQL 终端”中注释/取消注释行：

步骤 1 选择目标行。

步骤 2 在主菜单选择“编辑>行注释/取消行注释”，单独注释/取消注释每个选中的行。

也可以使用快捷键“Ctrl+/”，或右键单击行并选择“行注释/取消行注释”完成此操作。

----结束

按照以下步骤在“PL/SQL Viewer”或“SQL 终端”中注释/取消注释整段：

步骤 1 选择目标行或整段内容。

步骤 2 在主菜单选择“编辑 > 块注释/取消块注释”，注释/取消注释选中的行或整段内容。

也可以使用快捷键“Ctrl+Shift+/”，或右键单击行或整段内容并选择“块注释/取消块注释”完成此操作。

----结束

缩进/取消缩进行

缩进/取消缩进选项用于根据“首选项”页签中定义的缩进大小来移动行。

按照以下步骤在“PL/SQL Viewer”或“SQL 终端”中缩进行：

步骤 1 选中目标行。

步骤 2 按下 Tab 键或单击  。

根据“首选项”页签中定义的缩进大小来移动选中的行。可参见格式编辑器来修改缩进大小。

----结束

按照以下步骤在“PL/SQL Viewer”或“SQL 终端”中取消缩进行：

步骤 1 选中目标行。

步骤 2 使用快捷键“Shift+Tab”或单击  。

根据“首选项”页签中定义的缩进大小来移动选中的行。可参见格式编辑器来修改缩进大小。

说明

如果选中了多行，则只对存在缩进的行取消缩进。例如，假设用户选择了多行，其中有一行从位置 1 开始。在这种场景下，使用“Shift+Tab”键将取消除该行以外所有行的缩进。

----结束

插入空格

“**插入空格**”选项用于根据“首选项”页签中定义的缩进大小，将制表符替换为空格。

按照以下步骤在“PL/SQL Viewer”或“SQL 终端”中将制表符替换为空格：

步骤 1 选中目标行。

步骤 2 按下 Tab 键或“Shift+Tab”键。

Data Studio 会根据“首选项”页签中定义的缩进大小，将制表符替换为空格。可参见格式编辑器来修改缩进大小

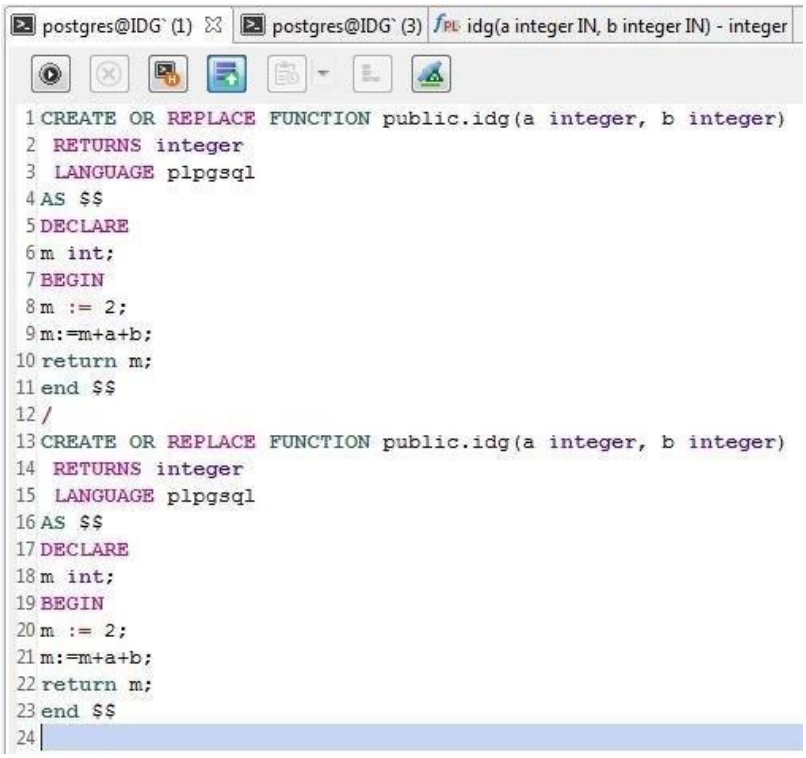
----结束

执行多条函数/过程或查询

执行以下步骤，执行多条函数/过程：

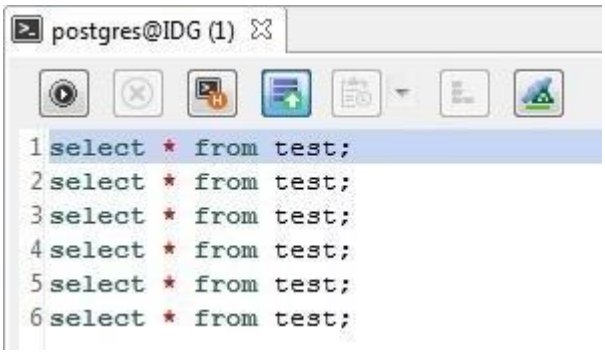
在“SQL 终端”页签中，在函数/过程定义之后插入“/”。

在下一行添加新的函数/过程语句。



执行如下步骤执行多条 SQL 查询：

步骤 1　在“SQL 终端”页签中输入多个 SQL 查询，如下所示：



步骤 2　在“SQL 终端”页签中单击，或按下“Ctrl+Enter”，或从主菜单中选择“**运行** > **编译/执行**”。

说明

- 如果没有选中任何查询，那么只有光标所在行的查询才会被执行。
- 如果光标处于一个空行，则将执行下一个可用的查询语句。
- 如果光标处于最后一个空白行，则不会执行任何查询。
- 如果单条查询以多行形式写入，且光标处于该条查询的任意一行，则执行该查询。多条查询使用英文分号（;）隔开。

----结束

执行以下步骤，在执行函数/过程后执行 SQL 查询：

在“SQL 终端”页签中，在函数/过程定义之后插入‘/’，然后添加新的查询语句或函数/过程语句。

执行以下步骤，在不同连接上执行 PL/SQL 语句和 SQL 查询：

在“连接”下拉菜单中选择所需连接，在“SQL 终端”中单击图标  。

重命名 SQL 终端

执行以下步骤重命名 SQL 终端：

步骤 1 在“SQL 终端”页签单击右键，选择“重命名终端”。

在显示的“重命名终端”对话框中，输入新的终端名称。

步骤 2 输入新的名称，单击“确定”。

说明

- 终端命名需遵从 Windows 系统的文件命名规则。
- “重命名终端”对话框最多允许输入 150 个字符。
- 恢复选项不能用于恢复到前一个名称。用户须手动将终端重命名为先前的名称。
- 重命名后的终端中，工具提示会显示原终端名称。

----结束

SQL 助手

SQL 助手为用户在 SQL 终端和“PL/SQL Viewer”页签中输入信息提供建议或参考。请按照以下步骤打开 SQL 助手：

启动 Data Studio 时，SQL 助手面板会显示相关语法。在 SQL 终端输入查询后，将显示对应语法详情，包括注意事项、示例、语法、函数和参数说明。选取文本后，可以通过右键选项复制所选信息，或复制粘贴至 SQL 终端。

说明

- 用户可选择“设置 > 首选项”，在“环境 > 会话设置”的“SQL 助手”区域设置永久启用或禁用 SQL 助手工具，默认情况下 SQL 助手功能永久启用。
- 当启用 SQL 助手功能后，可以单击工具栏中的 SQL 助手图标（  ）打开“SQL 助手”窗口。如果 SQL 助手已打开，SQL 助手图标是灰色，表示无效。

使用模板

在 Data Studio 中，用户可使用“**模板**”选项在 SQL 终端或 PL/SQL 查看器中插入常用 SQL 语句。Data Studio 中已保存了一些常用 SQL 语句以便用户使用。您可以修改这些现有模板，或删除或创建模板。有关如何添加、删除和创建模板，请参见添加/修改模板。

可用的默认模板如下：

名称	说明
df	delete from
is	insert into
o	order by
s*	select from
sc	select row count
sf	select from
sl	select

执行以下步骤使用“**模板**”选项：

步骤 1 在 SQL 终端或 PL/SQL 查看器中输入模板名称。

步骤 2 按下“Ctrl+Alt+空格”。

页面显示已有的模板信息列表。列表可按照如下模式显示。

精确匹配	列表显示
是	显示与输入文本大小写相匹配的所有条目。 示例：在 SQL 终端或 PL/SQL 查看器中输入“SF”，会显示所有以“SF”开头的条目。
否	显示与输入文本相匹配的所有条目，不考虑文本大小写。 示例：在 SQL 终端或 PL/SQL 查看器中输入“SF”，会显示所有以“SF”、“Sf”、“sF”或“sf”开头的条目。

文本选择/光标定位	列表显示
选择文本并使用快捷键	显示与所选内容左侧字符到其左侧首个空格或换行符为止的内容匹配的条目。
未选择文本但使用快捷键	显示与光标位置到其左侧首个空格或换行符为止的内容匹配的条目。

说明

- 在 SQL 终端或 PL/SQL 查看器中，如果不输入文本，直接按下快捷键“Ctrl+Alt+空格”，会显示“模板”中的所有条目。

- 在 SQL 终端或 PL/SQL 查看器中，如果输入的文本只有一个匹配项，则该项会直接替换输入的内容，而不显示模板列表。

----结束

4.20.11 导出查询结果

将 SQL 查询结果导出到 CSV、文本或二进制文件中。

本节包含如下内容：

导出所有查询结果

在执行导出操作过程中，以下操作无法启动：

- 在“SQL 终端”页签，执行 SQL 查询。
- 执行 PL/SQL 语句。
- 调试 PL/SQL 语句。

导出所有查询结果。

步骤 1 选择“**结果**”页签。

步骤 2 单击  。

显示“**导出结果集数据**”窗口。

请参见 4.16.7.4 导出表数据完成导出操作。

说明

可在状态栏查看结果的导出状态。

Data Studio 显示“**数据导出成功**”对话框。

步骤 3 单击“**确定**”。“**消息**”页签显示已完成操作的状态。

说明

如果在导出结果时磁盘已满，则 Data Studio 会在“消息”选项卡中报错。请清理磁盘，重新建立连接并导出结果数据。

----结束

“**消息**”页签显示“**执行时间**”、“**导出的总行数**”和 CSV 文件的保存路径。

导出当前页的查询结果

如要导出查询结果，建议导出所有查询结果，而非当前页的查询结果。“导出当前页到 CSV”功能已弃用。

执行以下步骤导出当前页查询结果。

步骤 1 选择“**结果**”页签。

步骤 2 单击  图标，导出当前页的查询结果。

显示“**Data Studio 安全免责声明**”对话框。

步骤 3 选择“**确定**”。

步骤 4 选择保存当前页的路径。

说明

可在状态栏查询导出页的状态。

步骤 5 点击“**保存**”。显示“**数据导出成功**”对话框。

步骤 6 单击“**确定**”。Data Studio 在“**消息**”选项卡显示操作状态。

说明

如果在导出结果时磁盘已满，则 Data Studio 会在“消息”选项卡中报错。清理磁盘，重新建立连接并导出结果数据。

----结束

4.20.12 管理 SQL 终端连接

在 Data Studio 中，用户查看执行计划和成本、可视化计划解释以及在结果集中进行操作时，可重用 SQL 终端中的现有连接或新建连接。默认情况下，SQL 终端会重用现有连接。当有多个查询在现有连接中排队等待执行时，请使用新连接，因为查询会按顺序执行且可能存在延迟。在处理临时表时请重用现有连接。有关编辑临时表的详情，请参阅 4.16.8 编辑临时表。

要更改该设置，请执行以下步骤：

步骤 1 单击  启用或禁用 SQL 终端中的连接重用功能。

有关重用连接和新建连接时的查询执行行为，请参阅 FAQs。

说明

用户仅能在已有连接中编辑临时表。

----结束

4.21 批量操作管理

4.21.1 概述

用户可以树形图方式查看“**对象浏览器**”中有权访问的数据库对象。例如，在树形图中，模式显示在所选数据库中，数据库表则显示在对应模式中。

“**对象浏览器**”仅显示满足当前用户的以下最低权限类型要求的对象。

对象类型	“对象浏览器”中显示的权限
数据库	连接
模式	使用
表	选择
列	选择
序列	使用
函数/过程	执行

用户有权访问的父对象的子对象没有必要在“**对象浏览器**”中显示。例如，如果用户有权访问表，但无权访问该表中的某一列，则“对象浏览器”仅显示该表中用户有权访问的列，而用户无权访问的列将不会显示。如果用户对某对象的访问权限在该用户执行该对象相关操作时取消，则会显示错误消息，提示用户无权执行该操作。刷新“**对象浏览器**”后，该对象将被删除。

可以树形图方式显示的对象如下所示：

- 模式
- 函数/过程
- 表
- 序列
- 视图
- 列、约束和索引

所有默认创建的模式（除公共模式外）均分组在“**系统模式**”下，用户模式分组在相应数据库的“**用户模式**”下。

说明

“**对象浏览器**”的过滤选项会打开一个新的页签，支持用户输入搜索范围。输入完成后，按回车键即可开始搜索。为了提高可用性，“**对象浏览器**”还提供了搜索栏，允许用户输入目标对象名称。如果节点树已展开，则工具仅显示符合过滤条件的对象。

如果节点未展开，则在节点展开时应用过滤规则。

4.21.2 批量删除对象

用户可使用批量删除操作选择多个对象进行删除。对于搜索到的对象，也可执行此操作。

说明

- 仅可在数据库中执行批量删除操作。
- 批量删除系统对象会报错，因为系统对象禁止被删除。

执行如下步骤批量删除对象：

步骤 1 按下 Ctrl+单击左键（逐个选择对象）或 Shift+单击左键（选择多个对象）选择要删除的对象。

步骤 2 单击右键并选择“**删除对象**”。

“**删除对象**”页签显示要删除的对象列表。

列名	说明	示例
类型	显示有关对象类型的信息。	表、视图
名称	显示对象名称。	public.bs_operation_201804
查询	显示将被执行以删除对象的查询。	DROP TABLE IF EXISTS public.a123
状态	显示删除操作的状态。 • - 未开始：删除操作尚未启动。 • - 进行中：该对象正在被删除。 • - 已完成：删除操作已完成。 • - 错误：发生错误，未删除该对象。	• 未开始 • 进行中 • 已完成 • 错误
错误消息	显示删除操作的失败原因。	表“*abc*”不存在，跳过。

步骤 3 选择所需参数。

选项	说明
级联	级联删除操作将删除其依赖对象和属性。只有在执行刷新操作后，被删除的依赖对象才会从对象浏览器中删除。

选项	说明
原子	原子删除操作执行成功会删除所有对象，执行失败则不会删除任何对象。
未选择	表示未选择自动或级联删除操作，不会删除依赖项其他对象的对象。

步骤 4 单击“**开始**”。

“**运行**”：显示从对象列表中删除的对象数量。

“**错误**”：显示由于出错而未删除的对象数量。

步骤 5 单击“**停止**”或关闭“**删除对象**”对话框，以停止删除操作。

关于复制到剪贴板、高级复制到剪贴板、显示/隐藏搜索栏和排序选项的具体信息，请参阅执行 SQL 查询。

说明

- 要复制单元格中的部分内容，先选中目标，然后按下“Ctrl+C”或单击  。
- 如果用户在“对象浏览器”中选择了多个对象进行删除，则系统将打开批量删除窗口，并在菜单栏中启用其相应的图标。如果用户此时断开数据库连接，则图标变为禁用状态，即使重新连接也不会启用。用户需要重新选择待删除的对象，之后所选对象将显示在新的批量删除窗中。

----结束

4.21.3 授权/撤销权限

批量授权/撤销操作允许用户同时选择多个对象，还支持搜索对象进行选择。

仅 OLAP 支持该特性，OLTP 不支持。

说明

只有模式和类型相同的对象才支持批量授权/撤销。

执行以下步骤批量授权/撤销权限：

步骤 1 按下 Ctrl+左键逐个选择对象，或 Shift+左键批量选中对象。

步骤 2 单击右键并选择“**授权/撤销权限**”。

弹出“**授权/撤销权限**”对话框。

步骤 3 参考 4.10.7 授权/撤销权限来授权/撤销权限。

----结束

4.22 自定义 Data Studio

4.22.1 通用

本节介绍如何自定义快捷键。

设置快捷键

可根据需要设置 Data Studio 快捷键。

执行如下步骤设置或修改快捷键：

步骤 1 从主菜单中选择“**设置** > **首选项**”。

显示“**首选项**”对话框。

步骤 2 选择“**快捷键**”。显示“**快捷键**”页签。

步骤 3 选择要修改的快捷键，单击“**修改**”。

步骤 4 在“**快捷键**”文本框中输入所需快捷键。

例如，要将“单步进入”的快捷键由“F7”改为“F6”，将光标移至文本框中，按下“F6”。

步骤 5 单击“**确认**”。显示“**Data Studio 即将重启**”对话框。

说明

可修改多个快捷键，然后再启动 Data Studio。

步骤 6 单击“**是**”重启 Data Studio。如果正在进行导入、导出或执行操作，会显示“**重启确认**”对话框。

步骤 7 单击“**确定**”以关闭运行中的作业并重新启动应用程序，或单击“**取消**”以取消重启操作。

----结束

执行如下步骤删除快捷键：

步骤 1 从主菜单中选择“**设置** > **首选项**”。

显示“**首选项**”对话框。

步骤 2 选择“**快捷键**”。显示“**快捷键**”页签。

步骤 3 选择要修改的快捷键，单击“**取消快捷键**”。

步骤 4 单击“**确认**”。显示“**Data Studio 即将重启**”对话框。

说明

可删除多个快捷键，然后再启动 Data Studio。

步骤 5 单击“**是**”重启 Data Studio。如果正在进行导入、导出或执行操作，会显示“**进程正在运行**”对话框。

步骤 6 单击“**确定**”等待操作完成，或单击“**强制重启**”放弃现有操作。

----结束

执行如下步骤还原默认快捷键设置：

步骤 1 从主菜单中选择“**设置** > **首选项**”。

显示“**首选项**”对话框。

步骤 2 选择“**快捷键**”。显示“**快捷键**”页签。

步骤 3 单击“**恢复默认**”。默认快捷键有关详情，请参见 4.7 Data Studio 右键菜单。

步骤 4 单击“**确认**”。显示“**Data Studio 即将重启**”对话框。

步骤 5 单击“**是**”重启 Data Studio。如果正在进行导入、导出或执行操作，会显示“**进程正在运行**”对话框。

步骤 6 单击“**确定**”等待操作完成，或单击“**强制重启**”放弃现有操作。

----结束

快捷键

与其他基于 Windows 的应用一样，Data Studio 支持快捷键。下表列出了一些快捷键以高效使用 Data Studio 提供的功能。要自定义快捷键，参见设置快捷键。

表4-27 Data Studio 默认键盘快捷键

功能	快捷键
按升序、降序或服务器接收顺序对视图表、编辑表和查询的结果集进行排序	Alt+单击
“帮助”菜单	Alt+H
保存 SQL 脚本	Ctrl+S
“编辑”菜单	Alt+E
编译/执行 SQL 终端语句	Ctrl+Enter
查找和替换	Ctrl+F
查找上一处	Shift+F3
查找下一处	F3
重做	Ctrl+Y
从“编辑表数据”页签中复制“执行时间”和“状态”	Ctrl+Shift+K

功能	快捷键
从自动建议列表中复制数据库对象	Alt+U
打开“调用堆栈”、“断点”窗格、和“变量”窗格	Alt+V
打开 SQL 脚本	Ctrl+O
单步跳过	F8
单步进入	F7
单步退出	Shift+F7
单独注释/取消注释行	Ctrl+/
定位到“对象浏览器”中第一个元素	Alt+Page Up 或 Alt+Home
定位到“对象浏览器”中最后一个元素	Alt+Page Down 或 Alt+End
定位到行	Ctrl+G
断开连接	Ctrl+Shift+D
格式化（SQL、PL/SQL）	Ctrl+Shift+F
更改为大写	Ctrl+Shift+U
更改为小写	Ctrl+Shift+L
更新“编辑表数据”、“属性”和“结果”窗口中的单元格或列。单击单元格或列标题以启用此选项。	F2
关闭当前“PL/SQL Viewer”页签、“表数据查看页签”、“执行查询”页签、或“属性”页签	Shift+F4
继续 PL/SQL 调试操作	F9
剪切	Ctrl+X
复制“对象浏览器”或终端中修饰的对象名。从“终端”,“结果”,“查看表数据”或“编辑表数据”页签复制所选数据。	Ctrl+C
复制“结果”,“查看表数据”或“编辑表数据”页签中的数据，包含/不包含列标题和行数字。	Ctrl+Shift+C
复制“编辑表数据”页签中的查询	Ctrl+Alt+C
复制“变量”页签内容	Alt+K

功能	快捷键
复制“调用堆栈”页签内容	Alt+J
复制“断点”页签内容	Alt+Y
可视化解释计划	Alt+Ctrl+X
联机帮助（显示用户手册）	F1
模板	Alt+Ctrl+Space
切换到第一个“SQL 终端”页签	Alt+S
全选	Ctrl+A
“设置”菜单	Alt+G
刷新（在“对象浏览器”区域框中）	F5
搜索对象	Ctrl+Shift+S
“调试”菜单	Alt+D
调试模板	F10
调试数据库对象	Ctrl+D
突出显示对象浏览器	Alt+X
“文件”菜单	Alt+F
新建连接	Ctrl+N
“运行”菜单	Alt+R
在“SQL 终端”页签间切换	Ctrl+Page Up 或 Ctrl+Page Down
展开/折叠所有对象	Ctrl+M
粘贴	Ctrl+V
折叠对象浏览导航树	Alt+Q
执行	Ctrl+E
执行计划和开销	Ctrl+Shift+X
终止运行中的查询	Shift+Esc
注释/取消注释行或整段	Ctrl+Shift+/
自动建议数据库对象列表	Ctrl+Space

4.22.2 编辑器

本节介绍如何自定义语法高亮、SQL 历史查询信息、模板和格式编辑器。

语法高亮

执行以下步骤自定义 SQL 高亮颜色：

步骤 1 从主菜单中选择“设置 > 首选项”。

显示“**首选项**”对话框。

步骤 2 选择“编辑器 > 语法高亮”，会显示“**语法高亮**”页签。

步骤 3 单击颜色按钮为语法类型自定义颜色。

例如，单击  自定义“字符串”的颜色。显示颜色选择对话框。

在对话框中是为特定语法类型选择所需颜色。可选择基础颜色或自定义颜色。

说明

可在“语法高亮”页签中单击“恢复默认”恢复到默认颜色方案。

步骤 4 单击“**确认**”。显示“**Data Studio 即将重启**”对话框。

步骤 5 单击“**是**”重启 Data Studio。如果正在进行导入、导出或执行操作，Data Studio 会显示“**进程正在运行**”对话框。

步骤 6 可单击“**强制重启**”放弃现有操作并重启 Data Studio，或单击“**确定**”继续进行操作。

说明

Preferences.prefs 文件包含自定义颜色设置。如果该文件损坏，Data Studio 会显示默认设置。

重启 Data Studio 后自定义颜色会生效。

----结束

SQL 历史记录

用户可在 Data Studio 中设置可用的 SQL 历史记录数，以及 SQL 历史记录中所保存查询的字符数。

执行以下步骤自定义 SQL 历史记录中要保存的已执行查询数和查询中的字符数：

步骤 1 从主菜单中选择“设置 > 首选项”。

显示“**首选项**”对话框。

步骤 2 选择“编辑器 > SQL 历史记录”。显示“**SQL 历史记录**”窗格。

步骤 3 在“**SQL 历史记录数**”字段设置要保存的查询数。

说明

最小值为 1，最大值为 1000。页面会显示该项当前的值。

步骤 4 在“**SQL 查询字符数**”字段中，设置保存在“SQL 历史记录”中的每个查询可包含的字符数。

说明

最小值为 1，最大值为 1000。输入“0”表示不设置字符限制。页面会显示该项当前的值。

步骤 5 单击“**应用**”。

步骤 6 单击“**确认**”。

说明

- 如要恢复默认值，在“SQL 历史记录”窗格单击“恢复默认”。
- “SQL 历史记录数”的默认值是 50，“SQL 查询字符”的默认值是 1000。
- 如果输入的值小于原有值，可能导致数据丢失。此时会显示提示消息，告知用户数据丢失的风险，并询问是否要继续操作。
- 如果在保存更改前离开该窗格，会显示提示消息，提醒用户存在未保存的更改。
- 更改“SQL 历史记录数”字段不会影响锁定查询的数量。例如，如果锁定查询的数量为 50，且“SQL 历史记录数”设置为 25，则 SQL 历史记录中将显示 50 个锁定查询。
- 更改“SQL 查询字符”仅对在此次更改后添加的查询生效。

----结束

添加模板

用户可自定义 Data Studio，创建、编辑和删除模板。有关模板的详细信息，请参见使用模板。

说明

如果恢复默认设置，列表中所有用户自定义模板将被删除。

执行以下步骤创建模板：

步骤 1 从主菜单中选择“设置 > 首选项”。

显示“**首选项**”对话框。

步骤 2 选择“编辑器 > 模板”。显示“**模板**”窗格。

步骤 3 单击“**新建**”。

步骤 4 在“**名称**”字段中输入模板的名称。

步骤 5 在“**说明**”字段中输入说明。

步骤 6 在“**模式**”字段中输入 SQL 语句模式。

说明

在“模式”字段中输入的文本会进行语法高亮。

步骤 7 单击“**确定**”。

----结束

修改模板

执行以下步骤编辑模板：

步骤 1 从主菜单中选择“设置 > 首选项”。

显示“**首选项**”对话框。

步骤 2 选择“编辑器 > 模板”。显示“**模板**”窗格。

步骤 3 单击“**编辑**”。

步骤 4 根据需要修改“**名称**”字段。

步骤 5 根据需要修改“**说明**”字段。

步骤 6 根据需要修改“**模式**”字段。

说明

在“模式”字段中输入的文本会进行语法高亮。

步骤 7 单击“**确定**”。

----结束

删除模板

执行以下步骤删除模板：

步骤 1 从主菜单中选择“设置 > 首选项”。

显示“**首选项**”对话框。

步骤 2 选择“编辑器 > 模板”。显示“**模板**”窗格。

步骤 3 选中要删除的模板，单击“**删除**”。

所选模板从“**模板**”列表中删除。

说明

删除的默认模板可以通过“还原删除项”选项重新加回。它会将模板还原为最近更新的版本。“还原删除项”选项不适用于用户定义的模板。

----结束

恢复默认模板

执行以下步骤恢复默认模板：

步骤 1 从主菜单中选择“设置 > 首选项”。

显示“**首选项**”对话框。

步骤 2 选择“编辑器 > 模板”。显示“**模板**”窗格。

步骤 3 至少选择一个修改后的模板，将其恢复到默认设置。

步骤 4 单击“**恢复默认**”。

----结束

格式编辑器

在 Data Studio 中，用户可设置缩进和取消缩进时的制表符宽度，以及是否将制表符转换为空格。详情请参见缩进/取消缩进行。

按照以下步骤自定义缩进大小并将制表符转换为空格：

步骤 1 从主菜单中选择“设置 > 首选项”。

显示“**首选项**”对话框。

步骤 2 选择“编辑器 > 格式化”。显示“**格式化**”窗格。

步骤 3 选择“插入空格”选项将制表符转换为空格，或选择“**插入 Tab 制表符**”在行缩进/取消缩进操作时添加/删除制表符。

步骤 4 在“**缩进**”字段中输入缩进大小，定义缩进/取消缩进/空格长度。

----结束

事务

按如下步骤修改“**事务**”相关设置：

步骤 1 从主菜单中选择“设置 > 首选项”。

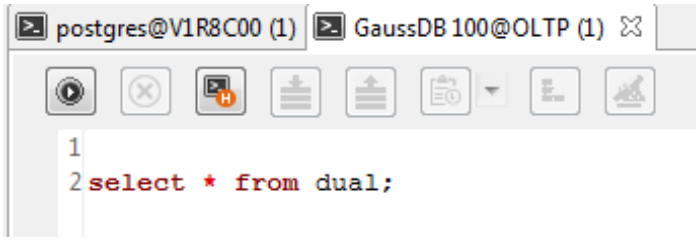
显示“**首选项**”对话框。

步骤 2 选择“编辑器 > 事务”。

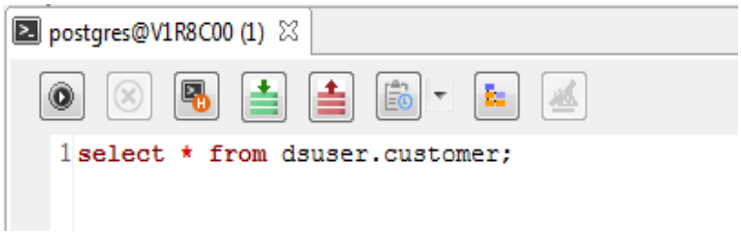
显示“**事务**”窗格。

步骤 3 在“**自动提交**”窗口可进行如下操作：

- 选择“**启用**”开启自动提交功能。此时，手动提交和回滚事务将被禁用，事务会被自动提交。



- 选择“**禁用**”关闭自动提交功能。此时，可手动提交或回滚事务。



说明

“自动提交”功能默认开启。

----结束

折叠

执行以下步骤进行折叠：

步骤 1 从主菜单中选择“设置 > 首选项”。

显示“**首选项**”对话框。

步骤 2 选择“编辑器 > 折叠”。

显示“**折叠**”窗格。

步骤 3 选择“**启用**”（默认）或“**禁用**”。

- **启用**：启用 SQL 折叠功能。支持对相应 SQL 语句进行折叠/展开。
- **禁用**：禁用 SQL 折叠功能。

说明

该功能设置变更仅在新的编辑器中生效；当前已打开的编辑器不生效，重启后生效。

----结束

字体

执行以下步骤设置字体大小：

步骤 1 从主菜单中选择“设置 > 首选项”。

显示“**首选项**”对话框。

步骤 2 选择“编辑器 > 字体”。

显示“**字体**”窗格。

步骤 3 设置字体大小（取值范围：1-50；默认值：10）。

----结束

自动建议

执行以下步骤实现自动建议：

步骤 1 从主菜单中选择“设置 > 首选项”。

显示“**首选项**”对话框。

步骤 2 选择“编辑器 > 自动建议”。

显示“**自动建议**”窗格。

步骤 3 在“**自动建议**”窗格，设置“**自动建议最小字符数**”。（默认值：2；取值范围：2-10）

自动建议要求对下列组进行排序：

1. 关键字
2. 数据类型
3. 已加载数据库对象

说明

- 每个组都必须经过排序。
- 关键字/数据类型区分数据库类型。
- 如果数据库断连，则必须显示默认关键字（即数据库类型）。
- 如果输入点（.），仅显示相关数据库对象，不显示关键字/数据类型。
- 支持通过快捷键启动自动建议功能。

----结束

4.22.3 环境

会话设置

执行以下步骤设置 Data Studio 和文件编码：

步骤 1 在主菜单中选择“设置 > 首选项”。

显示“**首选项**”对话框。

步骤 2 选择“环境 > 会话设置”，会显示“**会话设置**”页签。

步骤 3 从“**Data Studio 编码**”中选择编码。

步骤 4 从“**文件编码**”中选择文件编码。

说明

Data Studio 仅支持 UTF-8 和 GBK 文件编码类型。

步骤 5 单击“**确定**”，会显示“**Data Studio 即将重启**”对话框。

步骤 6 单击“**是**”重启 Data Studio。如果正在进行导入、导出、执行操作，Data Studio 会显示“**进程正在运行**”对话框。

步骤 7 单击“**强制重启**”放弃正在进行的操作，重新启动 Data Studio；或单击“**确定**”继续正在进行的操作。

说明

若要恢复默认值，可在“会话设置”窗格单击“恢复默认”。Data Studio 编码和文件编码的默认值为 UTF-8。

----结束

SQL 助手

执行以下步骤启用/禁用 SQL 助手：

步骤 1 在主菜单中选择“设置 > 首选项”。

显示“**首选项**”对话框。

步骤 2 选择“环境 > 会话设置”。显示“**会话设置**”窗格。

步骤 3 在“**SQL 助手**”页签选择“**启动/禁用**”选项。

步骤 4 单击“**确定**”。

说明

在“会话设置”窗格中单击“恢复默认值”以重置为默认值。“SQL 助手”的默认值为“启用”。

----结束

备份查询/函数/过程

有关 Data Studio 提供的备份功能的具体信息，请参阅备份未保存的查询/函数/过程。

按照以下步骤打开/关闭针对“SQL 终端”、“PL/SQL Viewer”中未保存数据的备份功能：

步骤 1 在主菜单中选择“设置 > 首选项”。

显示“**首选项**”对话框。

步骤 2 选择“环境 > 会话设置”。显示“**会话设置**”窗格。

步骤 3 在“**自动保存**”区域勾选/取消勾选“**自动保存**”。

步骤 4 在“**时间间隔**”字段输入数据备份时间间隔。

步骤 5 单击“**确认**”。

说明

在“会话设置”窗格中单击“恢复默认”可以重置为默认值。数据备份默认打开，备份时间间隔为 5 分钟。

----结束

按照以下步骤打开/关闭已存数据加密功能：

步骤 1 在主菜单选择“设置 > 首选项”。

弹出“**首选项**”对话框。

步骤 2 选择“环境 > 会话设置**”**。显示“**会话设置**”窗格。

步骤 3 在“**自动保存**”区域勾选/取消勾选“**加密**”。

步骤 4 单击“**确认**”。

说明

在“会话设置”窗格中单击“恢复默认”可以重置为默认值。加密功能默认打开。

----结束

按照以下步骤设置“**导入表数据限制**”和“**导入文件数据限制**”。

步骤 1 从主菜单选择“设置 > 首选项”。

弹出“**首选项**”对话框。

步骤 2 选择“环境 > 会话设置”。

弹出“**会话设置**“窗格。

在“**文件限制**”区域，设置“**导入表数据限制**”和“**导入文件数据限制**”参数。

导入表数据限制：设定最大可导入的表数据大小。

导入文件数据限制： 设定最大可导入的文件大小。

步骤 3 点击“**确认**”。

说明

上述截图中的值为默认值。

----结束

按照以下步骤进行渲染：

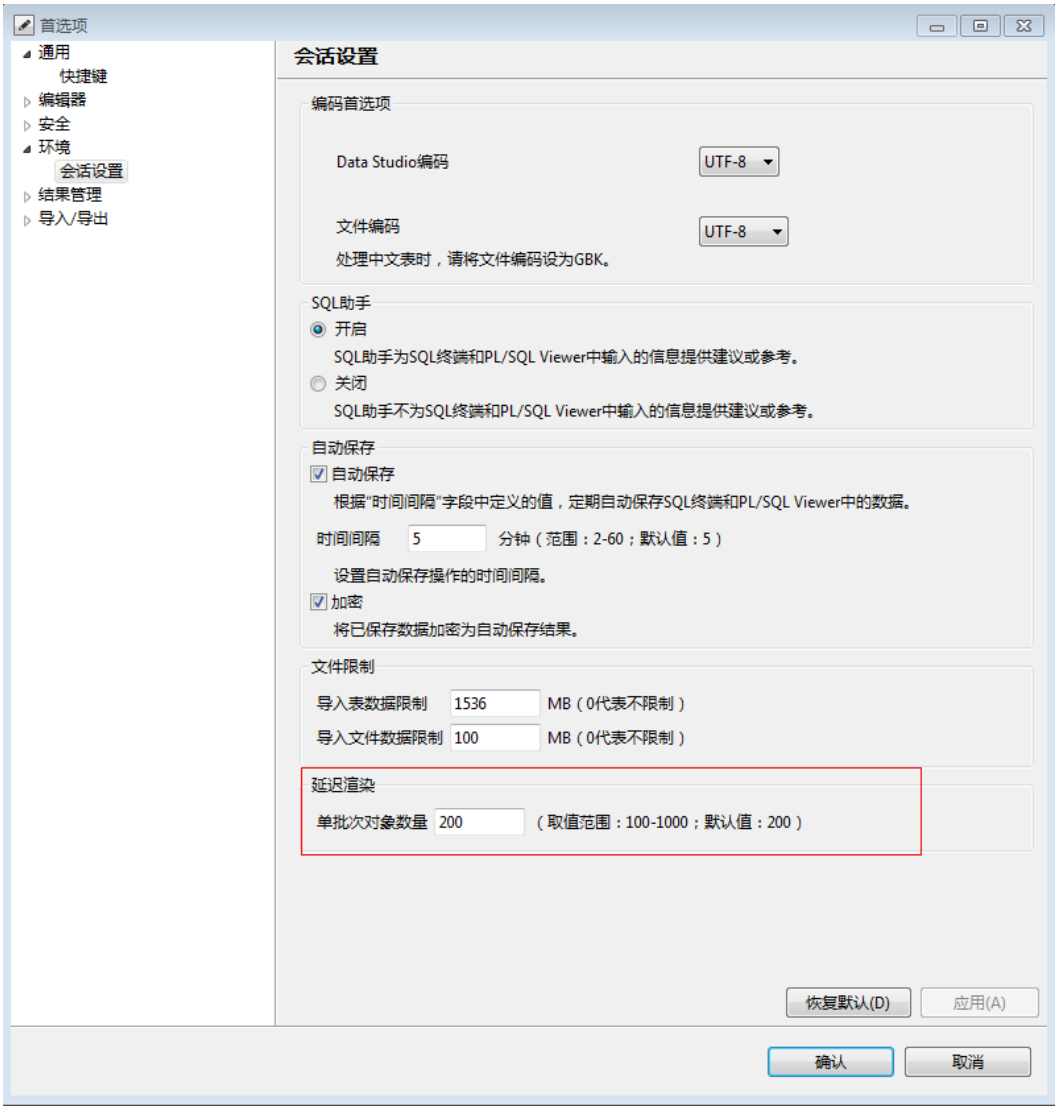
步骤 1 从主菜单选择“设置 > 首选项”。

弹出“**首选项**”对话框。

步骤 2 选择“环境 > 会话设置”。

弹出“**会话设置**“窗格。

“**延迟渲染**”区域显示“**单批次对象数量**”。



步骤 3　设置批量渲染时单批次中的对象数量。取值范围是 100-1000，默认值为 200。

如果取值超出范围，则工具报错“超出范围（100-1000）”。

步骤 4　单击“**确认**”。

----结束

4.22.4 结果管理

本章节介绍如何使用“查询结果”自定义设置查询结果的列宽、获取的记录数以及对结果中列标题或行号的复制。

查询结果

设置查询结果的列宽：

步骤 1　从主菜单中选择“设置 > 首选项”。

弹出“**首选项**”对话框。

步骤 2 选择“结果管理 > 查询结果”。

显示“**查询结果**”窗格。

步骤 3 选择所需选项。

列宽定制选项：

选项	结果
内容长度	选择此选项会按照查询结果内容的长度设置列宽。
自定义长度	选择此选项会将用户输入的值设为列宽。 说明 取值范围为 100 到 500。

步骤 4 单击“**确认**”。

说明

若要恢复默认值，可在“查询结果”窗格中单击“恢复默认”。默认值为“内容长度”。

----结束

设置要从查询结果中获取的记录数。

步骤 1 从主菜单中选择“设置 > 首选项”。

弹出“**首选项**”对话框。

步骤 2 选择“结果管理 > 查询结果”，会显示“**查询结果**”窗格。

步骤 3 选择所需项。

选项	结果
获取所有记录	选择此项可获取查询结果中的所有记录。
获取指定数量的记录	选择此项可设置查询结果中需要获取的记录数。 说明 取值范围为 100 到 5000。

步骤 4 单击“**确认**”。

说明

若要恢复默认值，可在“查询结果”窗格中单击“恢复默认”。默认值为“获取指定数量的记录”(1000)。

----结束

设置首选项，从查询结果中复制列名称和行号。

步骤 1 从主菜单中选择“设置 > 首选项”。

弹出“**首选项**”对话框。

步骤 2 选择“结果管理 > 查询结果”。显示“**查询结果**”框格。

步骤 3 选择所需项目。

选项	结果
包含列标题	选择此选项可从查询结果中复制列标题。
包含行号	选择此选项可从查询结果中复制所选内容和行号。

步骤 4 单击“**确认**”。

说明

若要恢复默认值，可在“查询结果”窗格中单击“恢复默认”。默认值为“包含列标题”。

----结束

设置首选项，决定结果集窗口的打开方式。

步骤 1 从主菜单中选择“设置 > 首选项”。

弹出“**首选项**”对话框。

步骤 2 选择“结果管理 > 结果窗口”。

步骤 3 选择所需项目。

选项	结果
覆盖结果集	如果当前有打开的结果集窗口，关闭之后打开新的结果集窗口。
保留当前结果集	打开新的结果集窗口，同时保留已打开的结果集窗口。

步骤 4 单击“**确认**”。

----结束

编辑表数据

设置保存编辑表数据的操作如下：

步骤 1 从主菜单中选择“设置 > 首选项”。显示“**首选项**”对话框。

步骤 2 选择结果“结果管理 > 编辑表数据”。显示“**编辑表数据**”窗格。

根据需要，选择所需选项：

表4-28 编辑表数据

数据库类型	自动提交	重用连接	表数据保存模式	操作
GaussDB (DWS)	打开	打开	保存有效数据	保存并提交所有有效数据，忽略不正确的数据。
	打开	打开	数据出错时，不保存任何数据	发生错误时，不保存任何数据。
	打开	关闭	保存有效数据	保存并提交所有有效数据，忽略不正确的数据。
	打开	关闭	数据出错时，不保存任何数据	发生错误时，不保存任何数据。
	关闭	打开	保存有效数据	发生错误时，不保存任何数据。用户需要执行 Commit 或 Rollback 命令。
	关闭	打开	数据出错时，不保存任何数据	发生错误时，不保存任何数据。用户需要执行 Commit 或 Rollback 命令。

步骤 3 单击“**确定**”。

说明

在“编辑表数据”窗格单击“恢复默认”，恢复默认值。默认值为“保存有效数据”。

----结束

结果数据编码

用户可在编辑、查看和查询结果窗口中设置是否显示数据编码类型。

按照以下步骤设置是否显示编码选项：

步骤 1 在主菜单中选择“设置 > 首选项”。

显示“**首选项**”对话框。

步骤 2 选择“结果管理 > 查询结果”。显示“**查询结果**”页签。

步骤 3 选择“**包含结果数据编码**”，在编辑表、查看表、查询结果时显示“**字符编码**”下拉列表。

步骤 4 单击“**确认**”。

说明

- 单击“结果管理”窗格中的“恢复默认”可恢复默认值。“包含结果数据编码”默认不选中。
- 要使更改生效，须再次执行编辑表、查看表属性或执行查询操作。

----结束

4.22.5 安全

本节介绍如何自定义密码和安全免责声明的显示情况。

永久保存密码

用户可设置是否在连接窗口中显示永久保存密码的选项。

执行以下步骤修改永久保存密码选项的显示情况：

步骤 1 在主菜单中选择“设置 > 首选项”。

显示“**首选项**”对话框。

步骤 2 选择“安全 > 密码”。显示“**密码**”页签。

步骤 3 选择所需选项。自定义选项如下表所示：

选项	说明
是	若选择该选项，则连接窗口的“保存密码”下拉列表中会显示“永久保存”选项。
否	若选择该选项，则连接窗口的“保存密码”下拉列表中不会显示“永久保存”选项，且已保存的密码会删除。

步骤 4 单击“**确定**”，会显示“**Data Studio 即将重启**”对话框。

步骤 5 单击“**是**”重启 Data Studio。如果正在进行导入、导出、执行操作，Data Studio 会显示“**进程正在运行**”对话框。

单击“**强制重启**”，放弃操作并重启 Data Studio。

步骤 6 重新启动 Data Studio；或单击“**确定**”继续正在进行的操作。

说明

若要恢复默认值，可在“密码”窗格单击“恢复默认”。默认值为“否”。

----结束

密码过期

本节介绍如何在密码过期后使用密码设置以继续/停止使用 Data Studio。

按照如下步骤在密码过期时修改 Data Studio 的行为：

步骤 1 从主菜单中选择“设置 > 首选项”。

显示“**首选项**”对话框。

步骤 2 选择“安全 > 密码”。显示“**密码**”页签。

步骤 3 选择所需选项。自定义选项如下表所示：

选项	说明
是	选择该选项后，您可以在密码过期后登录 Data Studio。 说明 将显示一条消息，通知您密码已过期，并且在以下情况下某些操作可能无法正常工作： • 建立新的连接。 • 编辑连接。 • 如果连接配置文件中没有其他数据库连接，则在创建数据库时连接到该数据库。 • 在该连接配置文件中没有其他数据库连接时连接到数据库。
否	如果选择此选项，一旦密码过期，您将无法登录到 Data Studio。将显示一条消息，通知您密码已过期。

步骤 4 单击“**确认**”。显示“**Data Studio 即将重启**”对话框。

步骤 5 单击“**是**”重启 Data Studio。如果正在进行导入、导出、执行操作，Data Studio 会显示“**进程正在运行**”对话框。

步骤 6 单击“**强制重启**”会放弃操作并重启 Data Studio。单击“**确定**”会继续正在进行的操作。

说明

默认值为“是”。

----结束

安全免责声明

用户可设置是否对不安全的连接/文件操作显示安全免责声明。

执行以下步骤修改安全免责声明的显示情况：

步骤 1 在主菜单中选择“设置 > 首选项”。

显示“**首选项**”对话框。

步骤 2 选择“安全 > 安全免责声明”，会显示“**安全免责声明**”页签。

步骤 3 选择所需选项。自定义选项如下表所示：

选项	说明
启用	若选择该选项，则用户每次尝试建立不安全的连接或执行文件操作时，会显示安全免责声明。
禁用	若选择该选项，则用户每次尝试建立不安全的连接或执行文件操作时，不会显示安全免责声明。用户需认识到不安全连接可能带来的安全问题。

步骤 4 单击“**确定**”，会显示“**Data Studio 即将重启**”对话框。

步骤 5 单击“**是**”重启 Data Studio。如果正在进行导入、导出、执行操作，Data Studio 会显示“**进程正在运行**”对话框。

步骤 6 单击“**强制重启**”会放弃操作并重启 Data Studio。单击“**确定**”会继续正在进行的操作。

说明

从“安全声明”窗格单击“恢复默认”以恢复为默认值。默认值为“启用”。

----结束

4.23 性能规格

Data Studio 在对象浏览器上加载和操作的性能直接取决于要加载的对象数量，包括表、视图、列等。

内存消耗也取决于加载对象的数量。

为了提高加载对象的性能和内存使用效率，建议将对象分割为多个命名空间，并避免使用包含大量对象、过度倾斜的命名空间。默认情况下，Data Studio 会为登录的用户加载 search_path 集中的命名空间。其他命名空间和包含的对象仅在需要时加载。

为了提高性能，建议加载所有对象，不要基于用户权限进行加载。表 4-29 提供有关对象浏览器中列出对象所需的最低访问权限的具体信息。

表4-29 最低权限要求

对象类型	权限类型	对象浏览器 - 最低权限类型
数据库	Create, Connect, Temporary/Temp, All	Connect
模式	Create, Usage, All	Usage

对象类型	权限类型	对象浏览器 - 最低权限类型
表	Select, Insert, Update, Delete, Truncate, References, All	Select
列	Select, Insert, Update, References, All	Select
视图	Select, Insert, Update, Delete, Truncate, References, All	Select
序列	Usage, Select, Update, All	Usage
函数	Execute, All	Execute

为了提高“查找”/“查找和替换”的操作性能，建议将超过 10000 字符的单行断开为多个短行。

以下性能测试的观测项和结果有助于了解 Data Studio 的各方面性能：

可配置的推荐最大内存（当前版本）		1.4 GB
性能（数据库中存在大小为 150 KB 的表和视图，各包含 3 列，使用最大内存配置）：		
>	刷新对象浏览器中的命名空间所需时间	15s
>	初始加载和扩展对象浏览器中所有表/视图所需的时间	90s-120s
>	后续加载和扩展对象浏览器中所有表/视图所需的时间	<10s
>	所用总内存	700 MB

说明

此处的性能数据仅供参考。实际性能可能因使用场景而异。

4.24 安全管理

4.24.1 概述

须知

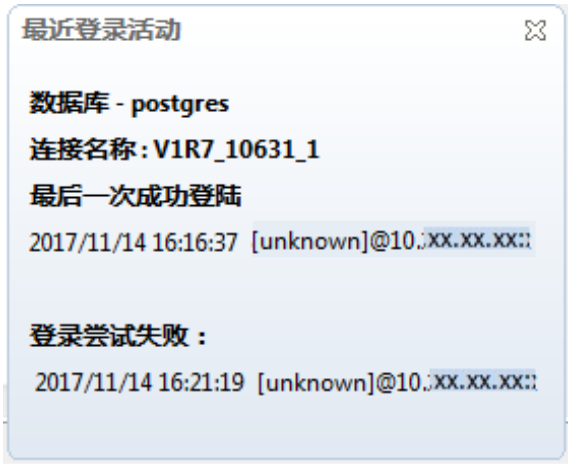
请务必使用最新的补丁更新操作系统和相关软件（详情请参见 4.1.4 系统要求），以防漏洞和其他安全问题。

本章节描述 Data Studio 的安全管理信息。

4.24.2 登录历史

如下信息对 Data Studio 的安全管理至关重要：

- 用户登录数据库后，Data Studio 会弹出一个窗口，描述最近一次的成功登录信息以及在最近两次成功登录数据库期间失败的登录尝试。



说明

如果弹出消息“未获取到上次登录信息”，则表示连接的数据库不支持显示上次登录信息。

4.24.3 密码到期通知

如下信息对 Data Studio 的安全管理至关重要：

- 系统在密码到期前 7 天开始会向您发送密码更换提醒。如果密码过期，请联系数据库管理员重置密码。
- 密码必须 90 天更换一次。

4.24.4 确保应用程序内存数据安全

如下信息对 Data Studio 的安全管理至关重要：

在受信任的环境中运行 Data Studio 时，必须防止恶意软件扫描或访问用于存储应用程序数据 (包括敏感信息) 的内存。

4.24.5 保存数据加密

如下信息对 Data Studio 的安全管理至关重要：

可以从“**首选项**”页面启用加密选项来对自动保存的数据进行加密。有关加密的具体步骤，请参见备份查询/函数/过程。

4.24.6 SQL 历史记录

如下信息对 Data Studio 的安全管理至关重要：

- 历史执行 SQL 脚本未加密。
- “历史执行 SQL”列表不显示包含如下关键字的敏感查询：
  - Alter Role
  - Alter User
  - Create Role
  - Create User
  - Identified by
  - Password
- 部分查询语法示例列举如下：
  - ALTER USER name [WITH] option [...]]
  - CREATE USER name [[WITH] option [...]]
  - CREATE ROLE name [[WITH] option [...]]
  - ALTER ROLE name [[WITH] option [...]]

4.24.7 SSL 证书

须知

证书的使用信息仅供参考。有关证书和管理证书及相关文件的安全指南的详细信息，请参见数据库服务器文档。

Data Studio 可以使用安全套接字层[SSL]选项连接到数据库。4.8.2 添加连接需要下列文件：

#	证书/密钥	说明
1	客户端 SSL 证书	由系统/数据库管理员提供。
2	客户端 SSL 密钥	由系统/数据库管理员提供。
3	Root 证书	由系统/数据库管理员提供。

SSL 证书生成和服务器配置

生成证书的步骤如下：

步骤 1 **搭建 CA 环境**：假设已创建 omm 用户，且 CA 路径为 test。

以 root 用户登录 SUSE Linux 操作系统，并切换到 omm 用户。

执行如下命令：

```
mkdir test
cd /etc/ssl
```

将配置文件 openssl. cnf 拷贝到 test 目录下。

命令如下：

```
cp openssl.cnf ~/test
cd ~/test
```

在 test 文件夹下建立 CA 环境。

在 demoCA./demoCA/newcerts./demoCA/private 目录下新建一个文件夹。

命令如下：

```
mkdir ./demoCA ./demoCA/newcerts ./demoCA/private
chmod 777 ./demoCA/private
```

创建 serial 文件，并将 01 写入其中。

命令如下：

```
echo '01'>./demoCA/serial
```

创建 index.txt 文件。

命令如下：

```
touch /home/omm/test/demoCA/index.txt
```

修改配置文件 openssl.cnf 中的参数。

命令如下：

```
dir = /home/omm/test/demoCA
default_md = sha256
```

CA 环境搭建完成。

步骤 2 **生成根私钥**：生成 CA 私钥。

命令如下：

```
openssl genrsa -aes256 -out demoCA/private/cakey.pem 2048
```

生成 2048-bit 的 RSA 私钥。

步骤 3 **生成根证书请求文件**：根证书应用文件名为 server.req。

命令如下：

```
openssl req -config openssl.cnf -new -key demoCA/private/cakey.pem -out
demoCA/careq.pem
```

输入 demoCA/private/cakey.pem 的口令。

输入 root 私钥密码。

系统会要求输入证书请求中需包含的信息。

您所输入的即为所谓的“区分名称”或“DN”。

其中一些字段可以不填。

一些字段会显示一个默认值，输入“.”可使字段为空。将如下信息填入生成的服务器和客户端证书。

```
Country Name (2 letter code) [AU]:CN
State or Province Name (full name) [Some-State]:shanxi
Locality Name (eg, city) []:xian
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Abc
Organizational Unit Name (eg, section) []:hello
-Common name can be any name
Common Name (eg, YOUR name) []:world
-Email is optional.
Email Address []:
A challenge password []:
An optional company name []:
```

步骤 4 **生成自签名根证书**。

命令如下：

```
openssl ca -config openssl.cnf -out demoCA/cacert.pem -keyfile
demoCA/private/cakey.pem -selfsign -infiles demoCA/careq.pem
```

使用 openssl.cnf 中的配置。

输入 demoCA/private/cakey.pem 的口令。

输入 root 私钥密码。

检查请求与签名是否匹配。

```
Signature ok
Certificate Details:
Serial Number: 1 (0x1)
Validity
Not Before: Feb 28 02:17:11 2017 GMT
Not After : Feb 28 02:17:11 2018 GMT
Subject:
countryName = CN
stateOrProvinceName = shanxi
organizationName = Abc
organizationalUnitName = hello
commonName = world
X509v3 extensions:
X509v3 Basic Constraints:
CA:FALSE
Netscape Comment:
OpenSSL Generated Certificate
X509v3 Subject Key Identifier:
F9:91:50:B2:42:8C:A8:D3:41:B0:E4:42:CB:C2:BE:8D:B7:8C:17:1F
X509v3 Authority Key Identifier:
keyid:F9:91:50:B2:42:8C:A8:D3:41:B0:E4:42:CB:C2:BE:8D:B7:8C:17:1F
Certificate is to be certified until Feb 28 02:17:11 2018 GMT (365 days)
```

```
Sign the certificate? [y/n]:y
1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

已下发名为 demoCA/cacert.pem 的 CA 根证书。

步骤 5 **生成服务器证书私钥**：生成名为 server.key 的私钥文件。

命令如下：

```
openssl genrsa -aes256 -out server.key 2048
```

步骤 6 **生成服务器证书请求文件**：生成服务器证书请求文件 server.req。

命令如下：

```
openssl req -config openssl.cnf -new -key server.key -out server.req
```

输入 server.key 的口令。

系统会要求输入证书请求中需包含的信息。

您所输入的即为所谓的“区分名称”或“DN”。

其中一些字段可以不填。

一些字段会显示一个默认值，输入“.”可使字段为空。

配置如下信息，确保与创建 CA 时内容一致。

```
Country Name (2 letter code) [AU]:CN
State or Province Name (full name) [Some-State]:shanxi
Locality Name (eg, city) []:xian
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Abc
Organizational Unit Name (eg, section) []:hello
-Common name can be any name
Common Name (eg, YOUR name) []:world
Email Address []:
-- The following information is optional.
A challenge password []:
An optional company name []:
```

步骤 7 **生成服务器证书**：将 demoCA/index.txt.attr 属性设为“no”。

```
vi demoCA/index.txt.attr
```

下发生成的服务器证书请求文件，下发成功后，会生成一个正式的服务器证书 server.crt。

```
openssl ca -config openssl.cnf -in server.req -out server.crt -days 3650 -md sha256
```

使用/etc/ssl/openssl.cnf 中的配置。

输入./demoCA/private/cakey.pem 的口令：

检查请求与签名是否匹配。

```
Signature ok
Certificate Details:
Serial Number: 2 (0x2)
```

```
Validity
Not Before: Feb 27 10:11:12 2017 GMT
Not After : Feb 25 10:11:12 2027 GMT
Subject:
countryName = CN
stateOrProvinceName = shanxi
organizationName = Abc
organizationalUnitName = hello
commonName = world
X509v3 extensions:
X509v3 Basic Constraints:
CA:FALSE
Netscape Comment:
OpenSSL Generated Certificate
X509v3 Subject Key Identifier:
EB:D9:EE:C0:D2:14:48:AD:EB:BB:AD:B6:29:2C:6C:72:96:5C:38:35
X509v3 Authority Key Identifier:
keyid:84:F6:A1:65:16:1F:28:8A:B7:0D:CB:7E:19:76:2A:8B:F5:2B:5C:6A
Certificate is to be certified until Feb 25 10:11:12 2027 GMT (3650 days)
-- Choose y to sign and issue the certificate.
Sign the certificate? [y/n]:y
-- Select y, the certificate singing and issuing is complete.
1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

启用私钥密码保护：若服务器私钥的密码保护未启用，用户需执行 gs_guc 命令加密密码。

```
gs_guc encrypt -M server -K root私钥密码 -D ./
```

使用 gs_guc 加密后，会生成私钥密码保护文件 server.key.cipher 和 server.key.rand。

步骤 8 **生成客户端证书和私钥**：生成客户端私钥。

```
openssl genrsa -aes256 -out client.key 2048
```

生成客户端证书请求文件。

```
openssl req -config openssl.cnf -new -key client.key -out client.req
```

生成的客户端证书请求文件签发下发后，会生成正式的客户端证书文件 client.crt。

```
openssl ca -config openssl.cnf -in client.req -out client.crt -days 3650 -md sha256
```

说明

若服务器的 pg_hba.conf 文件中 METHOD 被设置为 cert，客户端必须使用 License 文件 client.crt 中配置的用户名 username（普通用户）连接数据库。若 METHOD 被设置为 md5 或 sha256，客户端则没有此用户名限制。

若不删除客户端私钥的密码保护，则需要使用 gs_guc 对密码进行加密。

```
gs_guc encrypt -M client -K root私钥密码 -D ./
```

使用 gs_guc 加密后，会生成私钥密码保护文件 client.key.cipher 和 client.key.rand。

----结束

替换证书

在 LibrA 中配置 SSL 连接所需的默认安全证书和私钥。已从 CA 获取服务器和客户端的正式证书和秘钥。

步骤 1 准备证书和秘钥。服务器上的配置文件名称约定如下：

```
l Certificate name: server.crt
l Key name: server.key
l Key password and encrypted file: server.key.cipher and server.key.rand
Conventions for configuration file names on the client:
l Certificate name: client.crt
l Key name: client.key
l Key password and encrypted file: client.key.cipher and client.key.rand
l Certificate name: cacert.pem
l Names of files on in the revoked certificate list: sslcrl-file.crl
```

步骤 2 创建压缩包：

压缩包名称：db-cert-replacement.zip

压缩包格式：ZIP

压缩包文件列表：*server.crt, server.key,server.key.cipher, server.key.rand, client.crt, client.key, client.key.cipher,client.key.rand, cacert.pem*.

若需要配置证书撤销列表（CRL）， 压缩包文件列表必须包含 sslcrl-file.crl。

命令如下：

```
zip db-cert-replacement.zip client.crt client.key client.key.cipher client.key.rand
server.crt server.key server.key.cipher server.key.rand
zip -u ../db-cert-replacement.zip cacert.pem
```

步骤 3 调用证书替换接口替换证书。将准备好的压缩包 db-cert-replacement.zip 上传至集群用户的任一路径，如，/home/gaussdba/test/db-cert-replacement.zip。

执行如下命令替换 Coodinator 中的证书：

```
gs_om -t cert --cert-file=/home/gaussdba/test/db-cert-replacement.zip
```

Starting SSL cert files replace.

Backing up old SSL cert files.

Backup SSL cert files on BLR1000029898 successfully.

Backup SSL cert files on BLR1000029896 successfully.

Backup SSL cert files on BLR1000029897 successfully.

Backup gds SSL cert files on successfully.

BLR1000029898 replace SSL cert files successfully.

BLR1000029896 replace SSL cert files successfully.

BLR1000029897 replace SSL cert files successfully.

Replace SSL cert files successfully.

Distribute cert files on all coordinators successfully.

可以使用 gs_om -t cert --rollback 命令远程调用接口和 gs_om -t cert --rollback -L 命令。

----结束

客户端配置

步骤 1 对客户端密钥文件执行以下命令

```
openssl pkcs8 -topk8 -inform PEM -outform DER -in Client.key -out client.pk8
```

步骤 2 将前面创建的“client.pk8”，“client.crt”，“cacert.pem”文件复制到客户端。

说明

在 Data Studio 上选择“**客户端 SSL 秘钥**”时，该秘钥文件不可选，只能选择*.pk8 文件。然而，下载后的证书文件不包含该 pk8 文件。

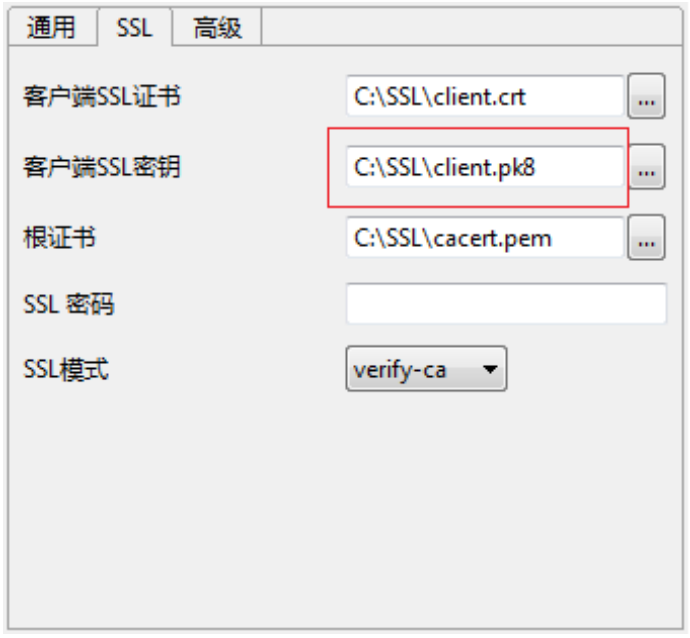
步骤 3 在服务器上对客户端配置“**双向**”SSL 认证。

```
hostssl     all          all          10.18.158.95/32       cert
```

在服务器上对客户端配置“**单向**”SSL 认证。

```
hostssl     all          all          10.18.158.95/32       sha256
```

步骤 4 登录 Data Studio 时，密码在双向 SSL 认证过程中未生效。



需要输入 SSL 密码。

----结束

4.25 故障处理

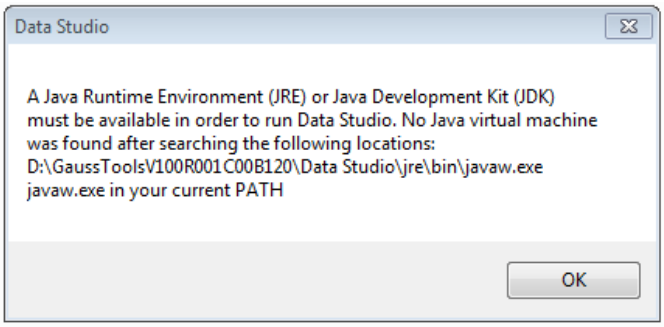
1. **Data Studio 长时间无法打开。**

   解决方法：检查是否未找到 JRE。验证环境中配置的 Java 路径。所支持的 Java JDK 版本，参见 4.1.4 系统要求。

2. **双击 Data Studio.exe 文件后，Data Studio 无法打开且显示 Java 运行错误。**

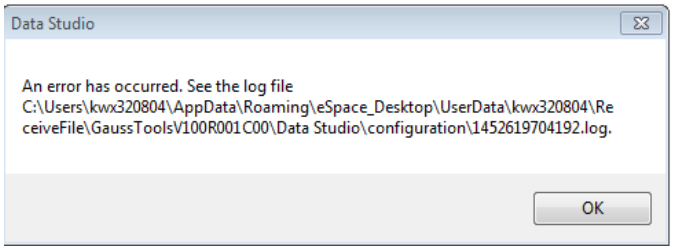
   解决方法：

   - 无 JRE：

     检查与操作系统位数对应的 JRE（Java 运行环境）或 JDK（Java 开发套件）版本 1.8.0_141 或以上是否已在系统中安装，并设置 Java Home 路径。如果安装了多个 Java 版本，请参照 4.2 安装配置 Data Studio 在配置文件中设置-vm 参数。这是运行 Data Studio 的前提条件。

   - 老版本 JRE：

     查询已安装的 JRE 或 JDK 版本。如果系统中安装的是旧版本，会引起该错误上报。将 JRE 版本更新到与操作系统位数对应的 1.8.0_141 或以上版本。

   - 不兼容 Java


Data Studio
Java was started but returned exit code=13
C:\ProgramData\Oracle\Java\javapath\javaw.exe
-Dosgi.requiredJavaVersion=1.8
-Xms40m
-Xmx1200m
-Dfile.encoding=UTF8
-Dds.encoding=UTF8
-jar C:\Users\sWX436505\Documents\Docs\English\DS\September 2017 Release\Local Binary\Sequence\eclipse\\plugins/org.eclipse.equinox.launcher_1.3.100.v20150511-1540.jar
-os win32
-ws win32
-arch x86_64
-showsplash
-launcher C:\Users\sWX436505\Documents\Docs\English\DS\September 2017 Release\Local Binary\Sequence\eclipse\Data Studio.exe
-name Data Studio
--launcher.library C:\Users\sWX436505\Documents\Docs\English\DS\September 2017 Release\Local Binary\Sequence\eclipse\\plugins/org.eclipse.equinox.launcher.win32.win32.x86_64_1.1.300.v20150602-1417\eclipse_1611.dll
-startup C:\Users\sWX436505\Documents\Docs\English\DS\September 2017 Release\Local Binary\Sequence\eclipse\\plugins/org.eclipse.equinox.launcher_1.3.100.v20150511-1540.jar
--launcher.overrideVmargs
-exitdata 2944_80
-clearPersistedState
-fetchResultData=1000
-viewTableFetchSize=200
-consoleLineCount=5000
-orderByColumn=-1
-enablePermanentPasswordSaveOption=false
-enableSecurityWarning=true
-logfolder=.
-loginTimeout=180
-data @none
-focusOnFirstResult=false
-vm C:\ProgramData\Oracle\Java\javapath\javaw.exe
-vmargs
-Dosgi.requiredJavaVersion=1.8
-Xms40m
-Xmx1200m
-Dfile.encoding=UTF8
-Dds.encoding=UTF8
-jar C:\Users\sWX436505\Documents\Docs\English\DS\September 2017 Release\Local Binary\Sequence\eclipse\\plugins/org.eclipse.equinox.launcher_1.3.100.v20150511-1540.jar
OK

检查系统安装的 JRE 或 JDK 版本。如果安装的 Java 版本位数与系统不兼容，会导致该错误。将 JRE 版本更新到与操作系统位数对应的 1.8.0_141 或以上版本。

建议运行 BAT 文件以检查 Java 版本兼容性，然后打开 Data Studio。详情请参见 4.3 快速入门。

3. **运行 StartDataStudio.bat 文件时显示如下信息。**

   解决方法：

信息	解决方案
您试图在如下环境运行 32 位 Data Studio： • 64 位操作系统 • Microsoft Windows 7 专业版 • 64 位 Java 1.8 JDK（不兼容） 请安装 32 位 Java 1.8。	安装 32 位 Java 1.8。
支持 Data Studio 的最低 Java 版本为 1.8。使用 Data Studio 前需安装 Java 1.8。	安装与操作系统位数对应的 Java 1.8。
您试图在如下环境运行 64 位 Data Studio： • 64 位操作系统 • Microsoft Windows 7 专业版 • 32 位 Java 1.8 JDK（不兼容） 请安装 64 位 Java 1.8。	安装 64 位 Java 1.8。
您试图在如下环境运行 64 位 Data Studio： • 32 位操作系统 • Microsoft Windows 7 专业版 • 32 位 Java 1.8 JDK（不兼容） 安装 32 位 Data Studio。	安装 32 位 Data Studio。

4. **所有输入内容正确，但 Data Studio 无法连接到服务器。**

   解决方法：检查服务器是否在指定 IP 地址及端口运行。通过 gsql 连接指定用户，检查其可用性。

5. **Data Studio 使用过程中的连接问题。**

   解决方法：Data Studio 使用过程中的连接问题，下例说明。

   创建数据库连接。

   执行查询。

当任一数据库（PostgreSQL）出现连接异常，该连接关闭。当数据库连接关闭时，所有打开的过程和函数窗口也会关闭。

系统显示错误提示，“对象浏览器”导航树显示数据库状态：

说明

只有当前数据库会中断。其他数据库仍保持连接状态，或重新连接。

重新连接数据库继续执行查询。

6. **通过 Java 应用获取包含中文批注的过程时，中文字符不可见。应如何处理？**

   解决方法：在“首选项 > 会话设置 > Data Studio 编码”和“文件编码”中将编码设置为 GBK，以便可以正常显示中文字符。

7. **连接到数据库，在“SQL 终端”上加载大量 SQL 查询和数据时，Data Studio 可能出现“Out Of Memory（内存不足）”或“Java Heap Error（Java 堆错误）”错误。应该怎么解决？**

   解决方法：Data Studio 已用尽所分配的最大 Java 内存时，提示“Out of Memory'”或“Java Heap Error”。缺省情况下，Data Studio.ini 配置文件（位于 Data Studio 安装路径下）包含表项“-Xmx1200m”。其中 1200m 代表 1200MB，为 Data Studio 可使用的最大 Java 内存。Data Studio 的内存占用率取决于 Data Studio 使用过程中用户获取的数据的大小。

   要解决该问题，可以扩展 Java 内存大小到理想的值。例如，可更新“-Xmx1200m”为“-Xmx2000m”，重新启动 Data Studio。如果更新后的内存用尽，同样的问题可能还会发生。

   说明

   - 对于 8GB RAM 的 32 位 Data Studio，Xmx 参数的值不得超过 2044。对于 8GB RAM 的 64 位 Data Studio，Xmx 参数的值不得超过 6000。该上限可能随用户的当前内存用量变化。

     例如：

     -Xms1024m

     -Xmx1800m

   - Data Studio 在 SQL 终端中支持的最大文件大小取决于 Data Studio.ini 文件中 Xmx 参数的值以及可用内存。

8. **如果执行的 SQL 查询返回大量数据，Data Studio 提示“Insufficient Memory（内存不足）”错误。应如何处理？**

   解决方法：Data Studio 会断开连接文件中指定的数据库。重新建立连接并继续操作。

9. **导出 DDL 或数据时为什么会收到导出失败的消息？**

   解决方法：这可能是由于以下原因：

   - 选择了无效的客户端 SSL 证书和/或客户端 SSL 密钥文件。请选择正确的文件，然后重试。有关详情，请参见 4.8.2 添加连接中创建连接相关的内容。
   - 数据库中对象的标识可能已更改。检查对象的标识是否已更改，然后重试。
   - 您可能没有足够的权限。联系数据库管理员获取所需权限。

10. **在执行显示 DDL 操作时，为什么会收到消息，提示显示 DDL 失败？**

    解决方法：这可能是由于以下原因：

- 选择了无效的客户端 SSL 证书和/或客户端 SSL 密钥文件。请选择正确的文件，然后重试。有关详情，请参见 4.8.2 添加连接中创建连接相关的内容。
- 数据库中对象的标识可能已更改。检查对象的标识是否已更改，然后重试。
- 您可能没有足够的权限。联系数据库管理员获取所需权限。

11. **在执行显示 DDL 或导出 DDL 操作时，为什么会收到以下错误消息？**

    “无法启动此程序，因为计算机中丢失 MSVCRT100.dll。尝试重新安装该程序以解决问题。”

    解决方案：显示或导出 DDL 时需执行 gs_dump.exe，这需要 Microsoft VC Runtime Library 文件 msvcrt100.dll。

    要解决此问题，请将 msvcrt100.dll 文件从\Windows\System32 文件夹复制到 \Windows\SysWOW64 文件夹。

12. **尝试建立连接时，为什么不显示已保存的连接详细信息？**

    解决方法：如果 User Data 文件夹下的 Profile 文件夹不可用或被手动修改，可能导致该问题。请确保 Profile 文件夹存在且其名称符合要求。

13. **关闭并重新打开 Data Studio 时，为什么 SQL 查询历史记录信息会丢失？**

    解决方法：如果 User Data 文件夹下的 Profile 文件夹丢失或被手动修改，可能导致该问题。请确保 Profile 文件夹存在且其名称符合要求。

14. **尝试修改“语法高亮”设置时，为什么会提示保存失败？**

    解决方法：如果 Preferences 文件不存在或被名称修改，可能导致该问题。请重新启动 Data Studio 以解决该问题。

15. **如果 Data Studio 处于空闲状态，而 Data Studio.log 文件状态为“No more handles（没有更多句柄）”，应如何处理？**

    解决方法：重新启动 Data Studio。

16. **如果在编辑表格后发 303 生错误，导致我无法继续进行修改，会发生什么？**

    解决方案：之前编辑的所有数据将会丢失。请关闭“编辑数据”窗口并重新进行修改。

17. **为什么在操作无误的情况下不断提示我“粘贴的单元格数量与所选单元格数量不匹配”？**

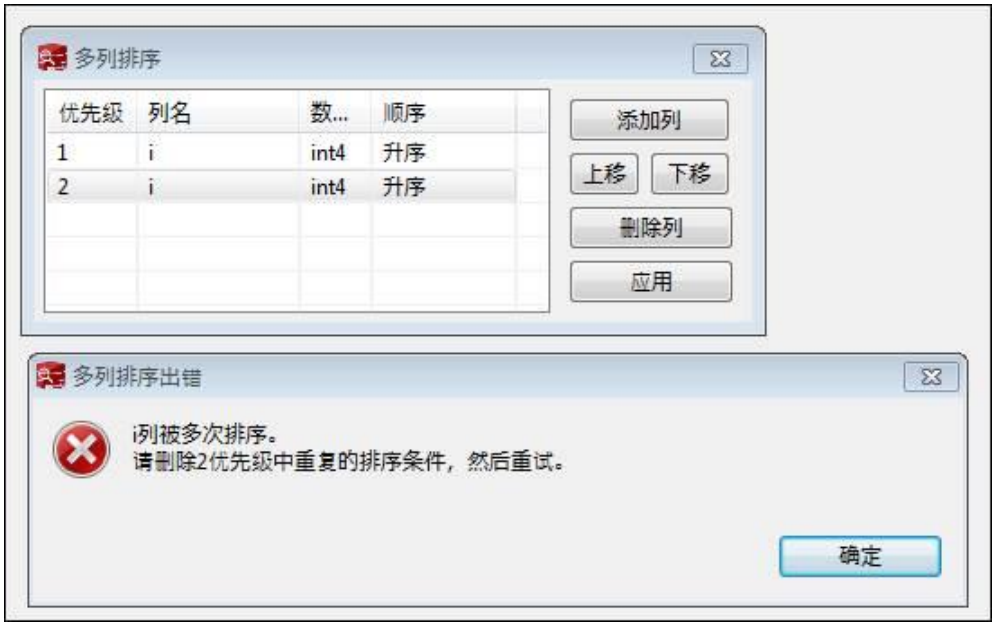
    解决方案： 如果选择“首选项 > 查询结果”后，设置了包含列标题，则会发生这种情况。此时所选单元格也包含列标题单元格。修改设置禁用包含列标题选项，然后重试。

18. **为什么“重用连接”选项禁用时，无法编辑临时表？**

    答：“重用连接”选项禁用后，工具会创建新会话，而临时表仅能在已有连接中编辑。要编辑临时表，请启用“重用连接”选项。有关详情，请参见 4.20.12 管理 SQL 终端连接。

19. **在多列排序对话框中多次添加同一列时会怎样？**

    答：如果用户在多列排序对话框中多次添加同一列且已经点击了“应用”按钮，系统弹出以下提示消息。用户需要单击“OK”然后选择非重复的列进行排序。



20. **未指定列名且已点击“应用”时会怎样？**

    答：系统弹出以下提示消息。用户需要设置有效的列名并再次单击“应用”，之后不会弹出此消息。

21. 当多个表查询正在 SQL 终端窗口运行时，单击“取消”会怎样？

    答：取消正在执行的表查询可能会导致控制台显示未创建的表名。此时，建议将该表删除，以便对具有相同名称的表进行操作。

22. 当用户因安全密钥被破解无法登录 Data Studio 时该怎么做？

    解决方案：按照以下步骤生成新的安全秘钥。

    a. 选择文件夹“Datastudio”>“Userdata”，删除其中的 Security folder 文件夹。

    b. 重启 Data Studio。

    c. 创建新的安全文件夹，重新生成密钥。

d. 重新输入密码登陆 Data Studio。

4.26 FAQs

1. **如果连接失败，需要检查哪些方面？**

   解答：检查以下几个方面：

   - 验证连接属性，检查连接属性输入是否正确。
   - 检查服务器和客户端版本是否兼容。
   - 检查 *database\pg_hba.conf* 文件是否正确配置。更多细节，参加服务器手册。
   - 检查 Data Studio.ini 文件是否正确配置。

2. **当用户通过 SSL 证书尝试和另一服务器建立连接时，为什么连接成功了？**

   解答：如果不同服务器使用相同 SSL 证书，那么第二次连接应成功，因为证书会缓存。

   当用户通过不同的 SSL 证书尝试和另一服务器建立连接时，由于证书不匹配连接失败。

3. **当用户右键点击过程并在“对象浏览器”窗口中进行刷新，过程本身不可见。原因是什么？**

   解答：当用户放弃功能并重新创建该功能时，这个问题可能出现。刷新主文件夹，在“**对象浏览器**”窗口中查看过程。

4. 如果关键错误在数据库会话过程中发生并且操作无法继续，如何处理？

   解答：关键错误可能发生在以下情景。检查：

   - 连接是否长时间空闲并且超时。
   - 服务器是否在运行。
   - 服务器是否有足够的内存并且“无内存”问题是否有上报服务器。

5. **限制条件是什么？**

   解答：限制条件用来限制表格中每列中不需要的数据插入。用户可以对任何表格的一列或多列创建限制条件。保持表格中的数据完整。

   支持的 3 种限制条件如下：

   - 主键 限制
   - 唯一 限制
   - 检查 限制

6. **索引是什么？**

   解答：索引是表格选择列中数据的复本，搜索非常高效。索引包含低级别磁盘块或直连连接到源数据所在行。

7. **Data Studio 文件的默认编码格式是什么？**

   解答：导入、导出的文件和系统文件使用的是系统的默认编码格式，该默认格式通过“设置 > 首选项”进行配置。默认编码格式为 UTF-8。

8. **我尝试打开 Data Studio 时，显示 Data Studio 不支持打开多个实例。原因是什么？**

解答：Data Studio 不支持同一用户打开多个实例。

9. **尝试对对象执行 DDL 操作时，任务无限期运行且无法取消任务。原因是什么？**

   解答：如果在同一对象上执行其他 DML/DDL 活动操作，则可能发生这种情况。此时需关闭对象上的所有 DML/DDL 活动操作，然后重试。如果问题仍然存在，可能是由于另一个用户正在对该对象执行 DML/DDL 操作。请等待一段时间后重试。可参考 4.4 Data Studio 用户界面自定义表数据查看事务的行为。

10. **为什么导出的查询结果与“结果”选项卡中的数据不同？**

    解答：导出结果集数据时，会使用新连接重新执行查询。因此，导出的结果可能与“结果”选项卡中显示的数据不同。

11. **为什么上次登录信息显示为“未获取到上次登录信息”？**

    解答：连接到旧版本数据库服务器，或在数据库创建后首次登录时，会显示该消息。

12. **SQL 终端上的错误消息标记不正确。**

    解答：当服务器返回错误的行编号时，会发生这种情况。可在“消息”页签重新查看错误消息，并定位至对应编号的行修复错误。

13. **显示 DDL 和导出 DDL 时会显示已删除的列信息吗？**

    解答：是的，显示 DDL 和导出 DDL 操作会显示已删除的列信息。

14. **为什么修改-Xmx 参数后无法启动 Data Studio？**

    解答：如果-Xmx 的参数值无效，则可能发生该问题。详情请参见 4.2 安装配置 Data Studio。

15. **如果我打开了多个终端或页签，如何更快地访问终端？**

    解答：根据屏幕分辨率，打开的终端或选项卡的数量达到一定限制后，终端列表末尾的图标  会显示下拉选项。单击该图标并从下拉列表中选择所需的终端。如果此图标不可用，请根据工具提示来识别终端和选项卡。可输入以上“SQL 终端”的列名称的值，来搜索终端名称。例如：

    - *s 显示所有名称以 s 开头的终端
    - test 显示所有名称以 test 开头的终端
    - *2 显示所有名称以 2 开头的终端

16. **变更语言设置并重启 Data Studio 后，为什么界面语言没有改变？**

    解答：有时界面语言不会在重启后更改为所选语言。请手动重启 Data Studio，使界面显示所选语言。

17. **为什么页面不显示上次登录的详细信息？**

    解答：有时服务器在尝试获取上次登录详细信息时会返回错误。在这种情况下，不会弹出上次登录的消息。

18. **查看/导出 DDL 时，为什么中文字符有时会显示为乱码？**

    解答：中文显示为乱码是因为查看的 SQL 语句、DDL、对象名称或数据中包含中文，且 Data Studio 客户端字符编码未设置为 GBK。设置 > 首选项 >设置 >文件编码，设置 Data Studio 客户端字符编码为 GBK。数据库编码和文件编码组合详细信息，请参考表 4-30。

    在 Windows 资源管理器中打开/查看导出的文件：对于使用 UTF-8 编码导出的文件，可双击查看，或右键单击文件并选择“打开”进行查看。对于以 GBK 编码导

出的文件，必须使用 Microsoft Excel 的导入外部数据功能（“数据 > 获取外部数据 > 自文本”）。

表4-30 支持的文件编码组合

数据库编码	Data Studio 文件编码	是否支持表名包含中文	是否支持表名为英文
GBK	GBK	**是**	**是**
GBK	UTF-8	否 - 乱码	否 - 乱码
UTF-8	GBK	否 - 提示导出失败	否 - 乱码
UTF-8	UTF-8	**是**	**是**
UTF-8	LATIN1	否 - 提示导出失败	**是**
SQL_ASCII	GBK	**是**	**是**
SQL_ASCII	UTF-8	否 - 乱码	否 - 乱码

19. **为什么会收到“不支持 GBK 和 LATIN1 相互转换”的错误信息？**

    解答：如果 Data Studio 和所选的数据库编码不兼容，则会出现此消息。选择兼容的编码来解决该问题。兼容编码的详细信息，请参考表 4-31。

表4-31 兼容的编码格式

Data Studio 文件编码	数据库编码	是否兼容
UTF-8	GBK	是
	LATIN1	是
	SQL_ASCII	是
GBK	UTF-8	是
	LATIN1	否
	SQL_ASCII	是
SQL_ASCII	UTF-8	是
	LATIN1	是
	GBK	是

20. **为什么编译和执行的 PL/SQL 过程会被保存为 PL/SQL 函数？**

解答：数据库不区分 PL/SQL 函数和过程。对数据库而言，所有过程都是函数。因此 PL/SQL 过程会被保存为 PL/SQL 函数。

21. **为什么分布键无法编辑？**

    解答：分布键仅能在执行第一次插入操作时编辑。

22. **在编辑表数据的时候，如果没有在默认值列输入值，默认值是否会添加到数据库服务器？**

    解答：默认值会添加到服务器，但不会在保存“编辑表数据”页签后显示。使用“编辑表数据”页签中的刷新选项或再次重新打开该表可查看添加的默认值。

23. **为什么删除/修改表数据时系统提示**找到了多个匹配的行？

    解答：在删除/修改数据行时，根据“自定义唯一键”或“使用所有列”，系统发现其他的行需要修改或删除。如果选择“自定义唯一键”，则会删除/修改所选列中与待删除/修改数据完全匹配的行。如果选择“使用所有列”，则会删除/修改所有列中与待删除/修改数据匹配的行。因此，如果选择“是”，匹配“自定义唯一键”或“使用所有列”的重复记录将被删除/修改。如果选择“否",未保存的行会被标记修改。

24. 为什么当我右键单击一个文本框会看到其他上下文菜单选项？

    解答：Windows 7 提供了附加的上下文菜单选项，如“从右到左阅读顺序”和“显示 Unicode 控制字符”等，以便您的键盘支持从左到右和从右到左的输入方式。

25. **哪些对象不支持批量导出 DDL？哪些不支持批量导出 DDL 和数据？**

    解答：以下对象不支持批量导出 DDL/DDL 和数据：

    导出 DDL：

    连接，数据库，外表，序列，列，索引，约束，分区，函数/过程组，普通表组，视图组，模式组，系统表组

    导出 DDL 和数据：

    连接，数据库，命名空间，外表，序列，列，索引，约束，分区，函数/过程，视图，普通表组，模式组，系统表组

26. **在启用“重用连接”选项并禁用“自动提交”选项的情况下，如果修改并保存了查询的结果集，能否在 SQL 终端中提交查询？**

    解答：不行。要提交查询，必须在 SQL 终端中执行 COMMIT 命令。

自动提交	重用连接	保存结果集
启用	启用	保存
启用	禁用	保存
禁用	启用	不保存
禁用	禁用	不支持

27. **在新的 SQL 终端窗口中查询临时表时，结果集中显示了错误的表信息。为什么会出现该问题？**

解答：在新的 SQL 终端窗口中查询临时表时，如果禁用了“重用连接”选项，且数据库中存在与该临时表同名的普通表、分区表或外表，结果集中会显示同名表的信息。

说明

如果启用“重用连接”选项，则结果集会显示该临时表的信息，无论是否存在同名表。

28. **对于锁定的对象，以下哪些操作不在后台运行，但需要手动终止？**

解答：对于已在其他操作中锁定的对象，以下操作不在后台运行：

操作	
重命名表	创建约束
设置表模式	创建索引
设置表说明	添加列
重命名分区	-

29. **将表数据导出到 Excel 时，是否需要限制行和列大小？**

A: 需要。xlsx 格式最多支持 100 万行和 16384 列，xls 格式最多支持 64000 行和 256 列。

5 GDS 并行数据加载工具

5.1 安装配置和启动 GDS

操作场景

GaussDB(DWS)提供了数据服务工具 GDS 来帮助分发待导入的用户数据及实现数据的高速导入。GDS 需部署到数据服务器上。

数据量大，数据存储在多个服务器上时，在每个数据服务器上安装配置、启动 GDS 后，各服务器上的数据可以并行入库。GDS 在各台数据服务器上的安装配置和启动方法相同，本节以一台服务器为例进行说明。

背景信息

1. GDS 支持在如下的操作系统中安装：

   鲲鹏平台：

   - Community Enterprise Operating System 7.6。
   - EulerOS 2.0 SP8。
   - Red Hat Enterprise Linux Server release 7.5。
   - 中标麒麟 7.5/7.6。

   x86 平台：

   - 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. GDS 的版本需与集群版本保持一致（如：GDS V100R008C00 版本与 DWS 1.3.X 版本配套），否则可能会出现导入导出失败或导入导出进程停止响应等情况。

因此请勿使用历史版本的 GDS 进行导入。数据库版本升级后，请按照操作步骤中的办法下载 GaussDB(DWS)软件包解压缩自带的 GDS 进行安装配置和启动。在导入导出开始时，GaussDB(DWS)也会进行两端的版本一致性检测，不一致时会打屏显示报错信息并终止对应操作。

GDS 的版本号的查看办法为：在 GDS 工具的解压目录下执行如下命令。

```
gds -V
```

数据库版本的查看办法为：连接数据库后，执行如下 SQL 命令查看。

```
SELECT version();
```

操作步骤

步骤 1 在使用 GDS 导入/导出数据前，请先参考《数据仓库服务用户指南》的“教程：使用 GDS 导入数据 > 步骤 1：准备 ECS 作为 GDS 服务器”中的步骤：“准备弹性云服务器作为 GDS 服务器”、“下载 GDS 工具包和 SSL 证书”。

步骤 2 以 root 用户登录待安装 GDS 的数据服务器，创建存放 GDS 工具包的目录。

```
mkdir -p /opt/bin/dws
```

步骤 3 将 GDS 工具包上传至上一步所创建的目录中。

以上传 SUSE Linux 版本的工具包为例 ，将 GDS 工具包“dws_client_8.1.x_suse_x64.zip”上传至上一步所创建的目录中。

步骤 4 （可选）如果使用 SSL 加密传输，请一并上传 SSL 证书至步骤 2 所创建的目录下。

步骤 5 在工具包所在目录下，解压工具包。

```
cd /opt/bin/dws
unzip dws_client_8.1.x_suse_x64.zip
```

步骤 6 创建 GDS 专有用户及其所属的用户组。此用户用于启动 GDS 及读取源数据。

```
groupadd gdsgrp
useradd -g gdsgrp gds_user
```

步骤 7 分别修改工具包和数据源文件目录属主为 GDS 专有用户。

```
chown -R gds_user:gdsgrp /opt/bin/dws/gds
chown -R gds_user:gdsgrp /input_data
```

步骤 8 切换到 gds_user 用户。

```
su - gds_user
```

若当前集群版本为 8.0.x 及以前版本，请跳过步骤 9，直接执行步骤 10。

若当前集群版本为 8.1.x 版本，则正常执行以下步骤。

步骤 9 执行环境依赖脚本。（仅 8.1.x 版本适用）

```
cd /opt/bin/dws/gds/bin
source gds_env
```

步骤 10 启动 GDS 服务。

GDS 是绿色软件，解压后启动即可。GDS 启动方式有两种。

方式一：直接使用“gds”命令，在命令项中设置启动参数。

方式二：将启动参数写进配置文件“gds.conf”后， 使用“gds_ctl.py”命令启动。

对于集中一次性导入的场景推荐使用第一种方式。对于需要隔段时间再次导入的场景，推荐使用第二种方式以配置文件的形式提升启动效率。

- 方式一：直接使用“gds”命令，启动 GDS。
  - 非 SSL 模式传输数据的情况下，启动 GDS。

    ```
    gds -d dir -p ip:port -H address_string -l log_file -D -t worker_num
    ```

    示例：

    ```
    /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
    ```

  - 使用 SSL 加密方式传输数据的情况下，启动 GDS。

    ```
    gds -d dir -p ip:port -H address_string -l log_file -D
    -t worker_num --enable-ssl --ssl-dir Cert_file
    ```

    示例：

    以步骤 4 中 SSL 证书以上传至/opt/bin 为例，命令如下。

    ```
    /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/
    ```

  命令中的斜体部分请根据实际替换。

  - **-d** *dir*：保存有待导入数据的数据文件所在目录。本教程中为“/input_data/”。
  - **-p** *ip:port*：GDS 监听 IP 和监听端口。默认值为：127.0.0.1，需要替换为能跟 GaussDB(DWS)通信的万兆网 IP。监听端口的取值范围：1024~65535。默认值为：8098。本教程配置为：192.168.0.90:5000。
  - **-H** *address_string*：允许哪些主机连接和使用 GDS 服务。参数需为 CIDR 格式。此参数配置的目的是允许 GaussDB(DWS)集群可以访问 GDS 服务进行数据导入。所以请保证所配置的网段包含 GaussDB(DWS)集群各主机。
  - **-l** *log_file*：存放 GDS 的日志文件路径及文件名。本教程为“/opt/bin/dws/gds/gds_log.txt”。
  - **-D**：后台运行 GDS。仅支持 Linux 操作系统下使用。
  - **-t** *worker_num*：设置 GDS 并发线程数。GaussDB(DWS)及数据服务器上的 I/O 资源均充足时，可以加大并发线程数。

    GDS 是根据导入事务并发数来决定服务运行线程数的。也就是说即使启动 GDS 时设置了多线程，也并不会加速单个导入事务。未做过人为事务处理时，一条 INSERT 语句就是一个导入事务。
  - **--enable-ssl**：启用 SSL 加密方式传输数据。
  - **--ssl-dir** *Cert_file*：SSL 证书所在目录。需与步骤 4 中的证书保存目录保持一致。
  - 关于更多参数的设置信息请参考《数据仓库服务工具指南》中的“GDS 并行数据加载工具 > gds 命令简介”。
- 方式二：将启动参数写进配置文件“gds.conf”后， 使用“gds_ctl.py”命令启动。

a. 使用如下命令，进入 GDS 工具包的“config”目录下，配置“gds.conf”文件。“gds.conf”配置详细信息请参考表 5-1。

```
vim /opt/bin/dws/gds/config/gds.conf
```

示例：

配置“gds.conf”文件如下：

```
<?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>
```

配置文件信息如下：

- 数据服务器所在 IP 为 192.168.0.90，GDS 监听端口为 5000。
- 数据文件存放在“/input_data/”目录下。
- 错误日志文件存放在“/err”目录下。该目录需要拥有 GDS 读写权限的用户自行创建。
- 单个数据文件大小为 100MB。
- 每个错误日志大小为 100MB。
- 日志保存在“/log/gds_log.txt”文件中。该目录需要拥有 GDS 读写权限的用户自行创建。
- 只允许 IP 为 10.10.0.*的节点进行连接。
- GDS 进程以后台方式运行。
- 递归数据文件目录。
- 指定并发导入工作线程数目为 2。

b. 执行如下命令启动 GDS 并确认 GDS 是否启动成功。

```
python3 gds_ctl.py start
```

示例：

```
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.
```

----结束

gds.conf 参数说明

表5-1 gds.conf 配置说明

属性	说明	取值范围
name	标识名。	-
ip	监听 ip 地址。	IP 需为合法 IP 地址。 IP 的默认值：127.0.0.1
port	监听端口号。	取值范围：1024~65535，正整数。 默认值：8098。
data_dir	数据文件目录。	-
err_dir	错误日志文件目录。	默认值：数据文件目录
log_file	日志文件路径。	-
host	设置允许连接到 GDS 的主机 IP 地址（参数为 CIDR 格式，仅支持 linux 系统）。	-
recursive	是否递归数据文件目录。	取值范围： • true：递归 。 • false：不递归。 默认值：false。
daemon	是否以 DAEMON（后台）模式运行。	取值范围： • true：以 DAEMON 模式运行。 • false：不以 DAEMON 模式运行。 默认值：false。
parallel	导入工作线程并发数目。	取值范围：0~32，正整数。 默认值：1。

5.2 停止 GDS

操作场景

待导入数据成功后，停止 GDS。

操作步骤

步骤 1 以 gds_user 用户登录安装 GDS 的数据服务器。

步骤 2 请根据启动 GDS 的方式，选择停止 GDS 的方式。

- 若用户使用“gds”命令启动 GDS，请使用以下方式停止 GDS。
  a. 执行如下命令，查询 GDS 进程号。

  ```
  ps -ef|grep gds
  ```

  示例：其中 GDS 进程号为 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. 使用“kill”命令，停止 GDS。其中 128954 为上一步骤中查询出的 GDS 进程号。

  ```
  kill -9 128954
  ```

----结束

5.3 GDS 导入示例

示例：多数据服务器并行导入

规划数据服务器与集群处于同一内网，数据服务器 IP 为 192.168.0.90 和 192.168.0.91。数据源文件格式为 CSV。

1. 创建导入的目标表 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. 以 root 用户登录每台 GDS 数据服务器，在两台数据服务器上，分别创建数据文件存放目录“/input_data”。以下以 IP 为 192.168.0.90 的数据服务器为例进行操作，剩余服务器上的操作与它一致。

   ```
   mkdir -p /input_data
   ```

3. （可选）创建用户及其所属的用户组。此用户用于启动 GDS。若该类用户及所属用户组已存在，可跳过此步骤。

```
groupadd gdsgrp
useradd -g gdsgrp gds_user
```

4. 将数据源文件均匀分发至相应数据服务器的“/input_data”目录中。
5. 修改每台数据服务器上数据文件及数据文件目录“/input_data”的属主为gds_user。以下以 IP 为 192.168.0.90 的数据服务器为例，进行操作。

```
chown -R gds_user:gdsgrp /input_data
```

6. 以 gds_user 用户登录每台数据服务器上分别启动 GDS。

   其中 GDS 安装路径为“/opt/bin/dws/gds”，数据文件存放在“/input_data/”目录下，数据服务器所在 IP 为 192.168.0.90 和 192.168.0.91，GDS 监听端口为 5000，以后台方式运行。

   在 IP 为 192.168.0.90 的数据服务器上启动 GDS。

```
/opt/bin/dws/gds/gds -d /input_data -p 192.168.0.90:5000 -H 10.10.0.1/24 -D
```

   在 IP 为 192.168.0.91 的数据服务器上启动 GDS。

```
/opt/bin/dws/gds/gds -d /input_data -p 192.168.0.91:5000 -H 10.10.0.1/24  -D
```

7. 创建外表 tpcds.foreign_tpcds_reasons 用于接收数据服务器上的数据。

   其中设置**导入模式信息**如下所示：

   - 导入模式为 Normal 模式。
   - 由于启动 GDS 时，设置的数据源文件存放目录为“/input_data”，GDS 监听端口为 5000，所以设置参数“location”为“gsfs://192.168.0.90:5000/* | gsfs://192.168.0.91:5000/*”。

   设置**数据格式信息**是根据导出时设置的详细数据格式参数信息指定的，参数设置如下所示：

   - 数据源文件格式（format）为 CSV。
   - 编码格式（encoding）为 UTF-8。
   - 字段分隔符（delimiter）为 E'\x08'。
   - 引号字符（quote）为 0x1b。
   - 数据文件中空值（null）为没有引号的空字符串。
   - 逃逸字符（escape）默认和 quote 相同。
   - 数据文件是否包含标题行（header）为默认值 false，即导入时数据文件第一行被识别为数据。

   设置**导入容错性**如下所示：

   - 允许出现的数据格式错误个数（PER NODE REJECT LIMIT 'value'）为 unlimited，即接受导入过程中所有数据格式错误。
   - 将数据导入过程中出现的数据格式错误信息（LOG INTO error_table_name）写入表 err_tpcds_reasons。

   根据以上信息，创建的外表如下所示：

```
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. 通过外表 tpcds.foreign_tpcds_reasons，将数据导入目标表 tpcds.reasons。

```
INSERT INTO tpcds.reasons SELECT * FROM tpcds.foreign_tpcds_reasons;
```

9. 查询错误信息表 err_tpcds_reasons，处理数据导入错误。详细请参见 5.6 处理错误表。

```
SELECT * FROM err_tpcds_reasons;
```

10. 待数据导入完成后，以 gds_user 用户登录每台数据服务器，分别停止 GDS。

    以下以 IP 为 192.168.0.90 的数据服务器为例，停止 GDS。其中 GDS 进程号为 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
```

示例：多线程导入

规划数据服务器与集群处于同一内网，数据服务器 IP 为 192.168.0.90，导入的数据源文件格式为 CSV，同时导入 2 个目标表。

1. 在数据库中创建导入的目标表 tpcds.reasons1 和 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. 以 root 用户登录 GDS 数据服务器，创建数据文件存放目录“/input_data”，以及子目录“/input_data/import1/”和“/input_data/import2/”。

```
mkdir -p /input_data
```

3. 将目标表 tpcds.reasons1 的数据源文件存放在数据服务器“/input_data/import1/”目录下，将目标表 tpcds.reasons2 的数据源文件存放在目录“/input_data/import2/”下。

4. （可选）创建用户及其所属的用户组。此用户用于启动 GDS。若该用户及所属用户组已存在，可跳过此步骤。

```
groupadd gdsgrp
useradd -g gdsgrp gds_user
```

5. 修改数据服务器上数据文件及数据文件目录“/input_data”的属主为 gds_user。

```
chown -R gds_user:gdsgrp /input_data
```

6. 以 gds_user 用户登录数据服务器上启动 GDS。

其中 GDS 安装路径为“/gds”，数据文件存放在“/input_data/”目录下，数据服务器所在 IP 为 192.168.0.90，GDS 监听端口为 5000，以后台方式运行，设定并发度为 2，并设定递归文件目录。

```
/gds/gds -d /input_data -p 192.168.0.90:5000 -H 10.10.0.1/24  -D -t 2 -r
```

7. 在数据库中创建外表 tpcds.foreign_tpcds_reasons1 和 tpcds.foreign_tpcds_reasons2 用于接收数据服务器上的数据。

   以下以外表 tpcds.foreign_tpcds_reasons1 为例，讲解设置的导入外表参数信息。

   其中设置的**导入模式信息**如下所示：

   - 导入模式为 Normal 模式。
   - 由于启动 GDS 时，设置的数据源文件存放目录为“/input_data/”，GDS 监听端口为 5000，实际存放数据源文件目录为“/input_data/import1/”，所以设置参数“location”为“gsfs://192.168.0.90:5000/import1/*”。

   设置的**数据格式信息**是根据导出时设置的详细数据格式参数信息指定的，参数设置如下所示：

   - 数据源文件格式（format）为 CSV。
   - 编码格式（encoding）为 UTF-8。
   - 字段分隔符（delimiter）为 E'\x08'。
   - 引号字符（quote）为 0x1b。
   - 数据文件中空值（null）为没有引号的空字符串。
   - 逃逸字符（escape）默认和 quote 相同。
   - 数据文件是否包含标题行（header）为默认值 false，即导入时数据文件第一行被识别为数据。

   设置的**导入容错性**如下所示：

   - 允许出现的数据格式错误个数（PER NODE REJECT LIMIT 'value'）为 unlimited，即接受导入过程中所有数据格式错误。
   - 将数据导入过程中出现的数据格式错误信息（LOG INTO error_table_name）写入表 err_tpcds_reasons1。
   - 当数据源文件中一行的最后一个字段缺失（fill_missing_fields）时，自动设置为 NULL。

   根据以上信息，创建的外表 tpcds.foreign_tpcds_reasons1 如下所示：

```
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';
```

   参考以上设置，创建的外表 tpcds.foreign_tpcds_reasons2 如下所示：

```
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. 通过外表 tpcds.foreign_tpcds_reasons1 和 tpcds.foreign_tpcds_reasons2 将数据分别导入 tpcds.reasons1 和 tpcds.reasons2。

```
INSERT INTO tpcds.reasons1 SELECT * FROM tpcds.foreign_tpcds_reasons1;
INSERT INTO tpcds.reasons2 SELECT * FROM tpcds.foreign_tpcds_reasons2;
```

9. 查询错误信息表 err_tpcds_reasons1 和 err_tpcds_reasons2，处理数据导入错误。详细请参见 5.6 处理错误表。

```
SELECT * FROM err_tpcds_reasons1;
SELECT * FROM err_tpcds_reasons2;
```

10. 待数据导入完成后，以 gds_user 用户登录数据服务器，停止 GDS。

    其中 GDS 进程号为 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

背景信息

gds 可以为 GaussDB(DWS)提供导入导出数据的功能。更多详细内容可参考《开发指南》的“导入数据”和“导出数据”章节。

语法

```
gds [ OPTION ] -d DIRECTORY
```

其中，-d、-H 是必选参数，option 项是可选参数。gds 将 DIRECTORY 中的文件数据提供给 GaussDB(DWS)访问。

在启动 GDS 服务前， 请确定使用的 GDS 版本和数据库的版本保持一致，否则数据库会提示错误并终止导入导出操作，因此请注意 GDS 工具和数据库的版本务必严格匹配。具体版本可通过-V 参数进行查看。

参数说明

- -d dir

  设置待导入数据文件的目录。在 gds 进程权限允许的条件下，-d 指定的目录会自动被创建。

- -p ip:port

  设置 gds 监听 IP 和监听端口。

  IP 的取值范围：IP 需为合法 IP 地址。

IP 的默认值：127.0.0.1。

监听端口的取值范围：1024~65535，正整数。

监听端口的默认值：8098。

- -l log_file

  设置日志文件。本次特性添加了日志自动切分的功能。当设置-R 参数后 gds 会根据设置的值重新生成新的文件，以此来避免单个日志文件过大的问题。

  生成规则：gds 默认只识别后缀是 log 的文件重新生成日志文件。

  例如，当-l 参数指定为 gds.log，-R 指定为 20MB 的时候，当 gds.log 大小达到 20MB 后就会新创建一个 "gds-2020-01-17_115425.log"文件。

  当-l 指定的日志文件没有以 log 为后缀，例如：" gds.log.txt"， 则新创建的日志文件名为" gds.log-2020-01-19_122739.txt"。

  gds 启动时会检测-l 参数设置的日志文件是否存在，如果存在则根据当前日期时间新生成一个日志文件，不会覆盖之前的日志文件。

- -H address_string

  设置允许哪些主机连接到 gds，参数需为 CIDR 格式，仅支持 linux 系统。需要配置多个不同网段时，使用“,”分隔。例如：-H 10.10.0.0/24,10.10.5.0/24。

- -e dir

  设置导入时产生的错误日志存放路径。

  默认值：数据文件目录。

- -E size

  设置导入产生的错误日志的上限值。

  取值范围：0<size<1TB，请使用正整数+单位的形式进行取值设置，单位支持 KB、MB 和 GB。

- -S size

  设置导出单个文件大小上限。

  取值范围：1MB<size<100TB，请使用正整数+单位的形式进行取值设置，单位支持 KB、MB 和 GB。如果使用 KB，取值需要大于 1024KB。

- -R size

  设置-l 指定的 gds 单个日志文件大小上限。

  取值范围：1MB<size<1TB，请使用正整数+单位的形式进行取值设置，单位支持 KB、MB 和 GB。如果使用 KB，取值需要大于 1024KB。

  默认值：16MB

- -t worker_num

  设置导入导出工作并发线程数目。

  取值范围：0<worker_num<=200，正整数

  默认值：8

  推荐值：普通文件导入导出场景取值：CPU 核数*2；管道文件导入导出场景取值：64。

说明

当管道文件导入导出场景并发较大时，该值应不低于业务并发数。

- -s status_file

  设置状态文件，仅支持 linux 系统。
- -D

  后台运行 gds，仅支持 linux 系统。
- -r

  递归遍历目录（外表目录下的子目录）下文件，仅支持 linux 系统。
- -h

  显示帮助信息。
- --enable-ssl

  使用 SSL 认证的方式与集群通信。
- --ssl-dir Cert_file

  在使用 SSL 认证方式时，指定认证证书的所在路径。
- --debug-level

  设置 GDS 端的 debug 日志级别，以控制 GDS debug 相关的日志输出。

  取值范围：0、1、2

  - 0：仅打印导入导出相关的文件列表，日志量小，推荐在系统处于正常状态时使用设置。
  - 1：打印日志的完整信息，增加各节点的连接信息、session 转换信息和一些数据统计。
  - 2：打印详细的交互日志以及所属状态，输出较大量的 debug 日志信息，以帮助故障定位分析。推荐仅在故障定位时开启。

  默认值：1
- --pipe-timeout

  设置 GDS 操作管道文件的等待超时时间。

  说明

  - 该参数的设置是为了避免人为或程序自身问题造成管道文件的一端长时间不读取或者不写入，导致管道另一端的读取或写入操作 hang 住。
  - 该参数表示的超时时间不是指 GDS 一个导入导出任务的最长时间，而是 GDS 对管道文件的每一次 read/open/write 的最大超时时间，当超过--pipe-timeout 参数设置时间会向前端报错。

  取值范围：大于 1s。请使用正整数+单位的形式进行取值设置，单位支持 s、m 和 h。如：1 小时可以设置为 3600s、60m 或者 1h。

  默认值：1h/60m/3600s
- --pipe-size

  设置 GDS 管道文件导入/导出时所使用的文件容量。

  取值范围：大于 1K。

  默认值：操作系统允许的最大值，可以通过命令 cat /proc/sys/fs/pipe-max-size 查看。

  说明

  该参数只能在 Linux 内核版本不低于 2.6.35 的环境下使用。

示例

数据文件存放在“/data” 目录，IP 为 192.168.0.90，监听端口为 5000。

```
gds -d /data/ -p 192.168.0.90:5000 -H 10.10.0.1/24
```

数据文件存放在“/data/” 目录下的任意子目录，IP 为 192.168.0.90，监听端口为 5000。

```
gds -d /data/ -p 192.168.0.90:5000 -H 10.10.0.1/24 -r
```

数据文件存放在“/data/ ”目录，IP 为 192.168.0.90，监听端口为 5000，以后台方式运行，将日志保存在“/log/gds_log.txt”文件中，指定并发导入工作线程数目为 32。

```
gds -d /data/ -p 192.168.0.90:5000 -H 10.10.0.1/24 -l /log/gds_log.txt -D  -t 32
```

数据文件存放在“/data/ ”目录，IP 为 192.168.0.90，监听端口为 5000，只允许 IP 为 10.10.0.*的节点进行连接。

```
gds -d /data/ -p 192.168.0.90:5000 -H 10.10.0.1/24
```

数据文件存放在“/data/ ”目录，IP 为 192.168.0.90，监听端口为 5000，只允许 IP 为 10.10.0.*的节点进行连接，设定为使用 SSL 认证的方式与集群通信，证书文件存放在/certfiles/目录。

```
gds -d /data/ -p 192.168.0.90:5000 -H 10.10.0.1/24 --enable-ssl --ssl-dir
/certfiles/
```

说明

- 1 个 GDS 在同一时刻，只能为 1 个集群提供导入导出服务；
- 为满足安全要求，请通过-p 显式指定监听 ip 和监听端口。
- 证书文件包括根证书文件 cacert.pem，以及二级证书文件 client.crt 和秘钥文件 client.key。
- 在加载证书时，需要使用密码保护文件 client.key.rand 和 client.key.cipher。

5.5 gds_ctl.py

背景信息

在配置了 gds.conf 的情况下，就可通过 gds_ctl.py 控制 gds 的启动和停止。

前置条件

只支持在 Linux 系统执行该命令。执行前，需确保目录结构如下：

|----gds

|----gds_ctl.py

|----config

|-------gds.conf

|-------gds.conf.sample

或

|----gds

|----gds_ctl.py

|-------gds.conf

|-------gds.conf.sample

“gds.conf”的内容：

```
<?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>
```

“gds.conf”配置说明：

- name：标识名。
- ip：监听 ip 地址。
- port：监听端口号。
  取值范围：1024~65535，正整数。
  默认值：8098。
- data_dir：数据文件目录。
- err_dir：错误日志文件目录。
- log_file：日志文件路径。
- host：允许哪些主机连接到 gds。
- recursive：是否递归数据文件目录。
  取值范围：
  - true 为递归数据文件目录。
  - false 为不递归数据文件目录。
- daemon：是否以 DAEMON 模式运行，
  取值范围：
  - true 为以 DAEMON 模式运行。
  - false 为不以 DAEMON 模式运行。
- parallel：导入导出工作线程并发数目。
  默认并发数目为 8，最大为 200。

语法

```
gds_ctl.py [ start | stop all | stop [ ip: ] port | stop | status ]
```

描述

当配置了“gds.conf”，可通过 gds_ctl.py 启动/停止 gds。

参数说明

- start
  启动 gds.conf 中配置的 gds。
- stop
  关闭当前用户有权限关闭的经配置文件启动的 gds 运行实例。
- stop all
  关闭当前用户有权限关闭的所有 gds 运行实例。
- stop [ip:] port
  关闭当前用户有权限关闭的特定 gds 运行实例。如果启动时指定了 ip:port，那么停止需要指定相应的 ip:port；如果启动时未指定 IP，只指定 port，则停止只需指定相应的 port 即可。如启动和停止指定不同的信息，则停止失败。
- status
  查询通过 gds.conf 启动的 gds 实例的运行状态。

示例

启动 gds。

```
python3 gds_ctl.py start
```

停止由配置文件启动的 gds。

```
python3 gds_ctl.py stop
```

停止所有当前用户有权限关闭的 gds。

```
python3 gds_ctl.py stop all
```

停止当前用户有权限关闭的，由[ip:]port 指定的 gds。

```
python3 gds_ctl.py stop 127.0.0.1:8098
```

查询 gds 状态。

```
python3 gds_ctl.py status
```

5.6 处理错误表

操作场景

当数据导入发生错误时，请根据本文指引信息进行处理。

查询错误信息

数据导入过程中发生的错误，一般分为数据格式错误和非数据格式错误。

- 数据格式错误

在创建外表时，通过设置参数“LOG INTO error_table_name”，将数据导入过程中出现的数据格式错误信息写入指定的错误信息表 error_table_name 中。您可以通过以下 SQL，查询详细错误信息。

```
SELECT * FROM error_table_name;
```

错误信息表结构如表 5-2 所示。

表5-2 错误信息表

列名称	类型	描述
nodeid	integer	报错节点编号。
begintime	timestamp with time zone	出现数据格式错误的时间。
filename	character varying	出现数据格式错误的数据源文件名。 当 GDS 导入时，同时会包括对应 GDS 服务端的 IP 地址端口信息。
rownum	bigint	在数据源文件中，出现数据格式错误的行号。
rawrecord	text	在数据源文件中，出现数据格式错误的原始记录。
detail	text	详细错误信息。

- 非数据格式错误

  对于非数据格式错误，一旦发生将导致整个数据导入失败。您可以根据执行数据导入过程中，界面提示的错误信息，帮助定位问题，处理错误表。

处理数据导入错误

根据获取的错误信息，请对照下表，处理数据导入错误。

表5-3 处理数据导入错误

错误信息	原因	解决办法
missing data for column "r_reason_desc"	1. 数据源文件中的列数比外表定义的列数少。 2. 对于 TEXT 格式的数据源文件，由于转义字符（\）导致 delimiter（分隔符）错位或者 quote（引号字符）错位造成的错误。 **示例：**目标表存在 3 列字段，导入的数据如下所示。由于存在转义字符“\”，分隔符“|”被转义为	1. 由于列数少导致的报错，选择下列办法解决： • 在数据源文件中，增加列“r_reason_desc”的字段值。 • 在创建外表时，将参数“fill_missing_fields”设置为“on”。即当导入过程中，若数据源文件中一行数据的最后一个字段缺失，则把最后一个字段的

错误信息	原因	解决办法
	第二个字段的字段值，导致第三个字段值缺失。 `BE|Belgium||1`	值设置为 NULL，不报错。 2. 对由于转义字符导致的错误，需检查报错的行中是否含有转义字符（\）。若存在，建议在创建外表时，将参数“noescaping”（是否不对'\'和后面的字符进行转义）设置为 true。
extra data after last expected column	数据源文件中的列数比外表定义的列数多。	• 在数据源文件中，删除多余的字段值。 • 在创建外表时，将参数“ignore_extra_data”设置为“on”。即在导入过程中，若数据源文件比外表定义的列数多，则忽略行尾多出来的列。
invalid input syntax for type numeric: "a"	数据类型错误。	在数据源文件中，修改输入字段的数据类型。根据此错误信息，请将输入的数据类型修改为 numeric。
null value in column "staff_id" violates not-null constraint	非空约束。	在数据源文件中，增加非空字段信息。根据此错误信息，请增加“staff_id”列的值。
duplicate key value violates unique constraint "reg_id_pk"	唯一约束。	• 删除数据源文件中重复的行。 • 通过设置关键字“DISTINCT”，从 SELECT 结果集中删除重复的行，保证导入的每一行都是唯一的。 `INSERT INTO reasons SELECT DISTINCT * FROM foreign_tpcds_reasons;`
value too long for type character varying(16)	字段值长度超过限制。	在数据源文件中，修改字段值长度。根据此错误信息，字段值长度限制为 VARCHAR2(16)。

6 DSC SQL 语法迁移工具

6.1 概述

当客户选择切换到 DWS 数据库后可能会面临数据库的迁移任务，数据库迁移包括用户数据迁移和应用程序 sql 脚本迁移，其中，应用程序 sql 脚本迁移是一个复杂、高风险且耗时的过程。

DSC（Database Schema Convertor）是一款运行在 Linux 或 Windows 操作系统上的命令行工具，致力于向客户提供简单、快速、可靠的应用程序 sql 脚本迁移服务，通过内置的语法迁移逻辑解析源数据库应用程序 sql 脚本，并迁移为适用于 GaussDB(DWS)数据库的应用程序 sql 脚本。

DSC 不需要连接数据库，可在离线模式下实现零停机迁移，迁移过程中还会显示迁移过程状态，并用日志记录操作过程中发生的错误，便于快速定位问题。

迁移对象

DSC 支持迁移 Teradata、Oracle、Netezza、MySQL、DB2 数据库的对象有：

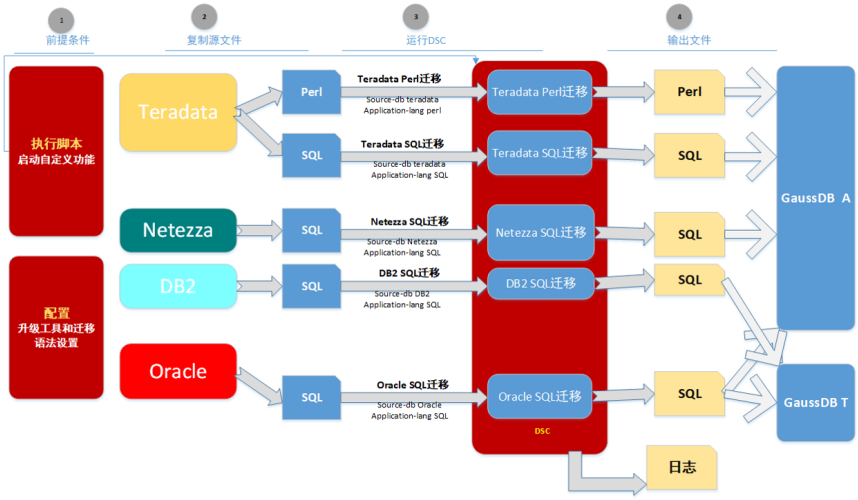
- Oracle、Teradata、Netezza、MySQL、DB2 支持的通用对象：SQL 模式，SQL 查询
- 仅 Oracle 和 Netezza 支持的对象：PL/SQL
- 仅 Teradata 支持的对象：包含 BTEQ 和 SQL_LANG 脚本的 Perl 文件

迁移流程

DSC 迁移 sql 脚本流程如下：

1. 从 Teradata 或 Oracle 数据库导出待迁移的 sql 脚本到已安装了 DSC 的 Linux 或 Windows 服务器。
2. 使用 DSC 工具进行语法迁移，命令中指定输入文件路径、输出文件路径以及日志路径。
3. DSC 自动将迁移后的 sql 脚本和日志信息归档在指定路径中。

图6-1 DSC 处理流程



6.2 支持的关键词和特性

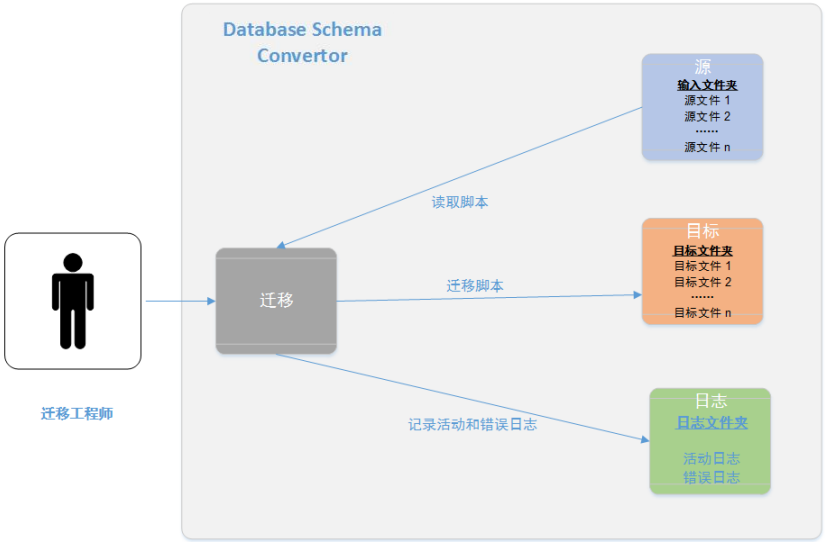
本节提供了 DSC 的系统上下文。DSC 是一款命令行工具，支持数据迁移工程师将 Teradata/Oracle/Netezza/MySQL/DB2 源数据库脚本迁移至 GaussDB(DWS)。

DSC 从输入文件夹读取脚本。迁移流程结束后，迁移后脚本保存到输出文件夹。工具还会在日志文件夹中生成操作日志和错误日志。

DSC 迁移最关键的是 sql 脚本关键字的迁移规则，本节提供 Teradata、Oracle、Netezza 和 MySQL 迁移工具支持的关键词和特性，及关键字迁移规则的配置参数，请根据源数据库、目的数据库和使用场景进行配置。

不支持迁移的关键词在迁移后会通过错误消息记录在错误日志中，错误日志还包含文件详细信息，如发生错误的文件名和查询位置。详情请参见 6.14 日志参考和 6.15 DSC 故障处理。

图6-2 DSC 的系统上下文



6.3 DSC 约束和限制

DSC 的约束和限制如下：

通用

- DSC 仅用于语法迁移，不支持数据迁移。
- 如果在将 IN/NOT IN 操作符转换为 EXISTS/NOT EXISTS 时，子查询的 SELECT 子句包含 aggregate 函数，则迁移的脚本可能发生问题。

Teradata

- DSC 无法区分 SQL 语句中的 mod 是否是关键字，当 mod 并非关键字时也会被替换为%。

  例如：DSC 将 SQL 语句中的 mod 字符串迁移为%。

  ```
  SELECT sal
   FROM employee
   WHERE name LIKE 'mod (%';
  ```

  该查询转换如下：

  ```
  SELECT sal
   FROM employee
   WHERE name LIKE 'mod (%';
  ```

- 如果含有 FORMAT 参数的 case 语句未用半角括号“()”括起来，该语句不会处理。

  例如：

  ```
  case when column1='0' then column1='value' end (FORMAT 'YYYYMMDD')as alias1
  ```

在该示例中，case when column1="0", column1="value" end 未用半角括号括起，因此不会处理该语句。

- 如果 Teradata 查询中同时使用 SELECT *和 QUALIFY 子句，迁移的查询会为 QUALIFY 子句多返回一列。

  例如：

  Teradata 查询

  ```
  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;
  ```

  迁移后的查询

  ```
  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;
  ```

  在迁移后的查询中，ROW_NUMBER( ) OVER(PARTITION BY Agt_Num ,Agt_Modif_Num ORDER BY NULL) AS ROW_NUM1 列为额外返回的一列。

- 子查询或函数内不支持对表的命名引用。

  例如，如果输入的查询包含表名（例如 foo），迁移工具不会将基于该表命名的引用从子查询（foo.fooid）中迁移到该表中，或从调用该子查询的函数（getfoo(foo.fooid)）中迁移到该表中。

  ```
  SELECT * FROM foo
   WHERE foosubid IN (
                   SELECT foosubid
                     FROM getfoo(foo.fooid) z
                    WHERE z.fooid = foo.fooid
                    );
  ```

Oracle

- 在如下场景中不处理 ROWID：

  ```
  SELECT empno,ename ,d.deptno,d.rowid  FROM ( SELECT * FROM employees where
  deptno is not null) e  , dept d WHERE d.deptno = e.deptno;
  ```

- 窗口函数（RANK，ROW_NUMBER）中的 aggregate 在字段列表中不可用。

  例如：

  ```
  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;
```

- 如果在 SELECT 语句中指定了星号（*），并且未指定 VALUES 子句，则 INSERT FIRST 或 INSERT ALL 查询的迁移将无法正常运行。
- DSC 只支持在 WHERE 子句的末尾指定 ROWNUM 条件。

  例如：

```
SELECT ROWNUM, ename,empid
FROM employees
WHERE empid = 10 AND deptno = 10 AND ROWNUM < 3
ORDER BY ename;
```

- 该工具不支持在 UPDATE 子句中使用 ROWNUM 函数。

  例如：

```
UPDATE tableName SET empno = ROWNUM
  where column = "value";
```

- 该工具不支持 INDEX 子句中基于列表分区的功能。

  例如：

```
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 不支持没有表名或表别名的 JOIN 条件。
- 存储过程和函数必须以“/”结尾。

6.4 系统要求（DSC）

支持的数据库

DSC 支持的源数据库如表 6-1 所示。

表6-1 支持的源数据库

数据库名称	数据库版本
Netezza	7.0
Teradata	13.1
Oracle	11g Release 2 12c Release 1
MySQL	5.6

软件要求

操作系统要求

DSC 兼容的操作系统如表 6-2 所示。

表6-2 兼容的操作系统

服务器	操作系统	版本
通用 x86 服务器	SUSE Linux Enterprise Server 11	SP1（SUSE11.1）
		SP2（SUSE11.2）
		SP3（SUSE11.3）
		SP4（SUSE11.4）
	SUSE Linux Enterprise Server 12	SP0（SUSE12.0）
		SP1（SUSE12.1）
		SP2（SUSE12.2）
		SP3（SUSE12.3）
	RHEL	6.4-x86_64（RedHat6.4）
		6.5-x86_64（RedHat6.5）
		6.6-x86_64（RedHat6.6）
		6.7-x86_64（RedHat6.7）
		6.8-x86_64（RedHat6.8）
		6.9-x86_64（RedHat6.9）
		7.0-x86_64（RedHat7.0）
		7.1-x86_64（RedHat7.1）
		7.2-x86_64（RedHat7.2）
		7.3-x86_64（RedHat7.3）
		7.4-x86_64（RedHat7.4）
	CentOS	6.4（CentOS6.4）
		6.5（CentOS6.5）
		6.6（CentOS6.6）
		6.7（CentOS6.7）
		6.8（CentOS6.8）

服务器	操作系统	版本
		6.9（CentOS6.9）
		7.0（CentOS7.0）
		7.1（CentOS7.1）
		7.2（CentOS7.2）
		7.3（CentOS7.3）
		7.4（CentOS7.4）
	Windows	7.0，10

其他软件要求

DSC 对其他软件版本的要求如表 6-3 所示。

表6-3 其他软件要求

软件	用途
JDK 1.8.0_141 及以上版本	用于运行 DSC 的软件版本。
Perl 5.8.8	用于迁移 Perl 文件。
Perl 5.28.2 及以上版本	用于在 Windows 中迁移 Perl 文件。

说明

此处的 Perl 5.8.8 为固定版本，考虑到语法兼容性，不能使用更高的版本。

6.5 安装 DSC

在使用 DSC 工具之前，必须在 Linux 或 Windows 服务器中安装工具，DSC 支持 Linux 64 位操作系统。DSC 支持其它操作系统的详情请见表 6-2。

前提条件

- 在 Linux 系统中请勿使用具有 root 权限的用户安装和操作 DSC。且该用户必须具有创建文件夹的权限，否则 install.sh 将执行失败。
- 请确保目标文件夹大小至少为输入文件夹中 SQL 文件大小的 4 倍。

  例如，输入文件夹中 SQL 文件大小为 100 KB，则目标文件夹至少需要 400 KB 空间处理 SQL 文件。

说明

- Linux 系统中查询目标文件夹的可用磁盘空间：

  ```
  df -P <folder path>
  ```

- Linux 系统中查询输入文件的大小，在输入文件所在路径下执行：

  ```
  ls -l
  ```

- 系统已安装 JRE 1.8 及以上版本和 Perl。有关软硬件环境的具体要求，请参见 6.4 系统要求（DSC）。

  执行以下步骤验证 Java 安装版本并设置 Java 路径。

  a. 验证 Java 安装是否符合要求。

  ```
  java -version
  ```

  b. 验证 java 路径是否设置，如果不正确请按照步骤重新设置。

  - Linux

    1) 验证 Java 路径是否设置。

    ```
    echo $JAVA_HOME
    ```

    2) 如果命令返回为空，请编辑当前用户的.bashrc 文件，输入如下内容，保存并退出。

    假设 Java 安装路径为“/home/user/Java/jdk1.8.0_141”。

    ```
    export JAVA_HOME=/home/user/Java/jdk1.8.0_141
    export PATH=$JAVA_HOME/bin:$PATH
    ```

    3) 激活 Java 环境变量。

    ```
    source ~/.bashrc
    ```

  - Windows

    1) 在“我的电脑”右键菜单中选择“属性”，弹出“系统”窗口。

    2) 单击“高级系统设置”，弹出“系统属性”对话框。

    3) 在“高级”页签中单击“环境变量”，弹出“环境变量”对话框。

    4) 选中“系统变量”中的“Path”环境变量，单击下方的“编辑”按钮，查看 Path 中是否包含 Java 安装路径。

    如果 Path 中未包含 Java 安装路径或路径不正确，请在原有内容的基础上增加本机的 Java 路径。

    假设 Java 安装路径为“C:\Program Files\Java\jdk1.8.0_141\bin”，Path 环境变量为 *“c:\windows\system32;*，则 Path 应该设置为 “*c:\windows\system32;***C:\Program Files\Java\jdk1.8.0_141\bin;**”。

操作步骤

DSC 是一款免安装工具，下载软件包后，用户解压软件包即可使用。

获取安装包方法如下。解压 **DSC.zip** 后再解压 **DSC.rar** 获取如表 6-4 所示的文件。

Windows:

步骤 1 下载软件包和校验文件。

- **DSC.zip**：软件包

- **DSC.zip.sha256.txt：**软件包检验

步骤 2 解压 DSC.zip 包。

得到 DSC 文件夹。

说明

解压 DSC.zip 时，可根据需要选择任意文件夹进行解压。

步骤 3 进入 DSC 目录。

步骤 4 找到并查看 DSC 目录中的文件。

解压出来的文件夹和文件说明如表 6-4 所示。

----结束

Linux 操作系统:

步骤 1 从 DSC.zip 包中提取文件。

```
sh install.sh
```

步骤 2 进入 DSC 目录。

```
cd DSC
```

步骤 3 查看 DSC 目录中的文件。

```
ls
config   lib   scripts   bin  input output runDSC.sh  runDSC.bat
```

----结束

表6-4 DSC 目录

文件或文件夹		说明
DSC	bin	dsc 相关 jar（可执行的）。
	config	DSC 工具的配置文件。
	input	输入文件夹
	lib	该文件夹中包括 DSC 正常运行所必须的库文件。
	output	输出文件夹
	scripts	该文件夹中包括 Oracle 和 Teradata 迁移的自定义配置脚本，用户可以直接执行 sql 文件启用所需功能。
	runDSC.sh	在 Linux 操作系统中运行应用程序。
	runDSC.bat	在 Windows 操作系统中运行应用程序。

文件或文件夹	说明
changelog	通知当前修改。
Install.sh	设置 DSC 的文件权限。
readme	在安装和配置前，阅读安装和配置操作说明。

说明

如果不再需要 DSC，可以通过删除 DSC 文件夹本身来卸载它。

6.6 配置 DSC

6.6.1 DSC 概述

DSC 提供配置文件和配置参数，用于控制迁移逻辑和迁移规则。迁移脚本之前，用户要根据场景和需求进行配置。涉及以下三种类型的配置：

1. 迁移前配置工具：用于在安装后 6.6.2 DSC 配置。
2. 迁移前配置迁移规则：用于配置 sql 脚本的迁移规则，针对 6.6.3 Teradata SQL 配置、6.6.4 Oracle SQL 配置、6.6.5 Teradata Perl 配置以及 6.6.6 MySQL SQL 配置迁移提供独立的配置文件。
3. 迁移后自定义配置：用于在目标数据库中 6.6.7 Netezza 配置，以支持目标数据库原本不支持的部分源数据库关键字。

6.6.2 DSC 配置

DSC 的配置包含如下内容：

- 设置 application.properties：用于配置工具的迁移行为，例如，是否要覆盖目标文件夹下的文件，是否对 sql 文件格式化。
- 设置 Java 内存分配：用户配置工具在迁移过程中可使用的内存资源，超出设置的内存，工具将显示错误消息并退出。

设置 application.properties

application.properties 文件中包括一系列应用配置参数，用于控制 DSC 在迁移数据库脚本时的行为，该文件中的参数为通用控制参数，适用于 Teradata、Oracle。

设置方法如下。

步骤 1 打开 config 文件夹中的 application.properties 文件。

步骤 2 根据实际需要修改 application.properties 文件中参数的值。

application.properties 文件中的参数解释见表 6-5。

说明

- 参数值不区分大小写。
- 除了列出的参数外，用户不得更改任何参数值。

步骤 3　保存后退出。

----结束

表6-5 application.properties 文件的配置参数

参数	说明	取值范围	默认值	样例
• formatterrequired	若该参数配置发生改变，工具将无法按预期运行。	• true • false	true	formatterrequired=true
• prevalidationFlag	若该参数配置发生改变，工具将无法按预期运行。	• true • false	true	prevalidationFlag=true
• commentSeparatorFlag	若该参数配置发生改变，工具将无法按预期运行。	• true • false	true	commentSeparatorFlag=true
• queryDelimiter	若该参数配置发生改变，工具将无法按预期运行。	NA	不适用	queryDelimiter=;
• blogicDelimiter	若该参数配置发生改变，工具将无法按预期运行。	NA	不适用	blogicDelimiter=/
• Timeout	工具迁移超时时间。 若该参数配置发生改变，工具将无法按预期运行。	-	4 hours	Timeout=4
• fileExtension	合法的文件扩展名，以逗号分隔。 如果此配置项 fileExtension 有修改，工具将不	• csv • txt • SQL	SQL	fileExtension=SQL

参数	说明	取值范围	默认值	样例
	能正常运行。 说明 导出的脚本必须带有如下后缀，如： • .sql • .txt • .fnc • .proc • .tbl • .tbs • .pl 等。			
• formattedSourceRequired	指定是否使用6.7.7 SQL Formatter 对源SQL 文件进行格式化。 若该参数设为true，则对输入文件的副本进行格式化并保存到*{输出路径}/formattedSource* 文件夹。	• true • false	true	formattedSourceRequired=true
• target_files	指定要在输出/目标文件中执行的操作。 Overwrite：用于覆盖输出文件夹中的现有文件。 指定是否必须覆盖输出文件夹中的文件。 Delete：用于删除目标文件夹中的所有文件。 Cancel：用于在输出/目标文件夹中存在文件时取消操作。	• overwrite • delete • cancel	overwrite	target_files=overwrite
• encodingFor	指定输入/源文件	• UTF8	基于区域	encodingFormat=

参数	说明	取值范围	默认值	样例
mat	的编码格式。 如果未设置该参数（或该参数被注释掉），则工具将基于区域设置使用默认编码。 说明 • 文件编码的自动检测功能并不准确。为确保正确的编码格式，请使用本参数指定格式。	• UTF16 • UTF32 • GB2312 • ASCII 等	设置的默认编码	UTF8
• NoOfThreads	指定用于迁移的线程数。	取决于可用的系统资源	3	NoOfThreads=3
• MaxFileSize Warning	指定输入文件大小告警阈值，单位为 B（字节）、KB、MB 或 GB 如果指定的值无效，那么将使用默认值。 如果指定的源文件大小超过指定的值，则会向用户显示以下警告消息： `**********` `*` `[WARNING] : Migration of the following files(>100KB) will take more time:` `bigfile001.SQL` `bigfile008.SQL` `**********` `*`	10 KB 1 GB	10MB	MaxFileSizeWarning=10MB
• MaxFileSize	允许输入文件的最大大小，如果	-	20MB	MaxFileSize=20 MB

参数	说明	取值范围	默认值	样例
	超过这个限制，文件迁移将被跳过。			
• MaxSqlLen	指定待迁移的单个查询的最大长度。 如果指定的值无效，则工具会将其重置为默认值，并且控制台上将显示以下错误消息： `The query length parameter (MaxSqlLen) value is out of range. Resetting to default value.` 如果输入查询超过指定的最大长度，则查询迁移会预验证失败。工具会跳过该查询，并记录以下错误消息： `2018-07-06 12:05:57,598 ERROR TeradataBulkHandler:195 Error occurred during processing of input in Bulk Migration. PreQueryValidation failed due to: Invalid termination; OR exclude keyword found in query; OR query exceeds maximum length (MaxSqlLen config parameter). filename.SQL for Query in position : xx`	1 .. 52,428,800 字节 （1 字节 - 50 MB）	1048576 （1 MB）	MaxSqlLen=1048576

参数	说明	取值范围	默认值	样例
• initialJVMMemory	设置初始内存	不适用	256 MB	initialJVMMemory=256MB 表明内存达到256MB 时，进程将启动。
• maxJVMMemory	设置最大内存	不适用	1024 MB	maxJVMMemory=2048m 表明内存达到2048 MB 时，进程将启动。

说明

- 如果为配置参数提供了错误或无效值，DSC 将采用该参数的默认值。
- 如果存在扩展名不受支持（如“.doc”），建议将此扩展名添加到“application.properties”文件的“fileExtension”配置参数中。

设置 Java 内存分配

DSC 支持通过参数控制 Java 虚拟机（JVM）的内存分配量，并预设默认值。

在迁移操作期间，如果内存使用超过设置的值，DSC 将提示“java.lang.OutOfMemoryError: GC overhead limit exceeded”错误消息并退出，此时用户可通过更改 application.properties 配置文件中的 initialJVMMemory 和 maxJVMMemory 的值，以分配更多内存。

说明

可用系统资源决定了内存分配量。

表6-6 JVM 内存分配的控制参数

参数	说明	推荐取值
Xms	指定初始内存分配量，单位为 MB。	该参数最小值为 256 MB，最大值取决于可用的系统资源。 默认值：256
Xmx	指定内存分配量的上限，单位为 MB。	该参数最小值为 1024 MB，最大值取决于可用的系统资源。 默认值：1024

6.6.3 Teradata SQL 配置

设置 Teradata 配置参数可在迁移 Teradata 数据库脚本时自定义迁移工具的行为。

打开 config 文件夹中的 features-teradata.properties 文件，并根据实际需要设置表 6-7 中的参数。

表6-7 features-teradata.properties 文件中的配置参数

参数	说明	取值范围	默认值	样例
• deleteToTruncate	该参数用于设置不含 WHERE 的 DELETE 语句迁移规则。 若该参数设为 true，则可将 DELETE 迁移为 TRUNCATE。若该参数设为 false，则不可将 DELETE 迁移为 TRUNCATE。	• true • false	false	deleteToTruncate=true
• distributeByHash	基于主索引中指定的字段，将数据分布在集群多个节点上。 若该参数设为 one，表示数据基于主索引的首个字段分布。 若该参数设为 many，表示数据基于所有主索引字段分布。 该功能通过指定 DISTRIBUTE BY 子句实现。 **说明** **该参数在 V100R002C60 版本中设置为 one，因为该版本不支持在 DISTRIBUTE BY 子句中指定多个字段。**	• one • many	many	distributeByHash=many
• extendedGroupByClause	该参数用于启用和禁用 3（grouping sets/cube/rollup）迁移。 若该参数设为 true，则可迁移 GROUP BY()。 若该参数设为 false，则	• true • false	false	extendedGroupByClause=false

参数	说明	取值范围	默认值	样例
	不可迁移 GROUP BY()。			
• inToExists	该参数可用于启用和禁用从 IN 和 NOT IN 转换到 EXISTS/NOT EXISTS 的查询优化。	• true • false	false	inToExists=false
• rowstoreToColumnstore	该参数将 rowstore（行存）表转换为 COLUMN STORE（列存）表。 如果该参数设为 true，则所有 rowstore 表脚本迁移时会转换为 columnstore 表。	• true • false	false	rowstoreToColumnstore=false
• session_mode	该参数用于在运行 CREATE TALBE 时设置默认表类型（SET/MULTISET）。 若该参数设为 Teradata，则默认表类型会配置为 SET。 若该参数设为 ANSI，则默认表类型会配置为 MULTISET。	• Teradata • ANSI	Teradata	session_mode=ANSI
• tdMigrateALIAS	该参数用于启用/禁用 ALIAS 迁移。 若该参数设为 true，则迁移 ALIAS。 若该参数设为 false，则不迁移 ALIAS。	• true • false	false	tdMigrateALIAS=true
• tdMigrateDOLLAR	该参数用于设置迁移工具行为，从而迁移名称以$（美元符号）开头的静态对象。该参数不适用于动态对象，这些对象的名称使用${}格式。 若该参数设为 true，则使用英文双引号（"）将以$开头的对象名称括起来。	• true • false	true	tdMigrateDOLLAR=true

参数	说明	取值范围	默认值	样例
	若该参数设为 false，则直接迁移以$开头的对象。 说明 详情请参见以$开头的对象名称。			
• tdMigrateLOCKoption	该参数是否迁移包含 LOCK 关键字的查询。 true 表示在迁移此类查询时注释掉 LOCK 功能（LOCK 到 ACCESS）。 false 表示不迁移此类查询。工具会跳过此查询，并记录以下消息： `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.` 说明 详情请参见 6.8.2.5 ACCESS LOCK。	• true • false	false	tdMigrateLOCKoption=true
• tdMigrateNULLIFZERO	该参数指定是否迁移 NULLIFZERO()。 若该参数设为 true，则迁移 NULLIFZERO()。 若该参数设为 false，则不迁移 NULLIFZERO()。	• true • false	true	tdMigrateNullIFZero=true
• tdMigrateVIEWCHECKOPTION	该参数指定是否迁移包含 CHECK OPTION 的视图。 若该参数设为 true，则迁移时注释掉此类视图。 若该参数设为 false，则不迁移此类视图。工具	• true • false	false	tdMigrateVIEWCHECKOPTION=true

参数	说明	取值范围	默认值	样例
	将按原样复制此查询并记录以下消息： `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.`			
• tdMigrateZEROIFNULL	该参数指定是否迁移 ZEROIFNULL()。 若该参数设为 true，则迁移 ZEROIFNULL()。 若该参数设为 false，则不迁移 ZEROIFNULL()。	• true • false	true	tdMigrateZEROIFNULL=true
• volatile	特定会话的 volatile 数据和表仅存储在该会话中。会话结束后，其数据和表会删除。 volatile 表可以是 6.8.2.1 表迁移表或 unlogged 表。 **说明** V100R002C60 仅支持 unlogged 表选项，不支持 local temporary 表。	• local temporary • unlogged	local temporary	volatile=unlogged
• tdMigrateCharsetCase	该参数指定是否迁移 CHARACTER SET 和 CASESPECIFIC。 若该参数设为 true，则迁移 CHARACTER SET 和 CASESPECIFIC 为注释掉的脚本。 若该参数设为 false，则不迁移 CHARACTER SET 和 CASESPECIFIC。工具按原样拷贝查询，并在错误日志文件中记录以下消息，包括查询细节	• true • false	false	tdMigrateCharsetCase=false **说明** 如果 tdminatecharsetcase = true，则注释该字符的特殊关键字。

参数	说明	取值范围	默认值	样例
	（如文件名和语句位置）： **CHARACTER SET** 和 **CASESPECIFIC** 是列级选项，Gauss 不提供等效语法。用户可以改写相应语句，或将 **tdMigrateCharsetCase** 参数设为 **true**，从而注释掉 **CHARACTER SET** 和 **CASESPECIFIC**。			
• terdataUtilities	是否支持迁移 Teradata 命令行工具。 支持以下参数值： • true • false	• true • false	true	terdataUtilities=true
• unique_primary_index_in_column_table	是否支持为列存表创建 unique 索引。	• true • false	true	unique_primary_index_in_column_table=true
• default_charset	是否支持迁移 default_charset。 支持以下参数值： • LATIN • UNICODE • GRAPHIC	• LATIN • UNICODE • GRAPHIC	LATIN	default_charset=LATIN
• mergeImplementation	指定 merge 类型： • 使用 WITH 子句 • 拆分查询 支持以下参数值： • With • Split • None	• With • Split • None	None	mergeImplementation=None
• dsqlSupport	是否支持 dsql。 支持以下参数值： • true • false	• true • false	false	dsqlSupport=false
• tdcolumnInSensitive	是否在迁移时删除包含双引号的列名称。	• true • false	false	tdcolumnInSensitive=false

参数	说明	取值范围	默认值	样例
	支持以下参数值： • true • false			
• tdMigrateCASE_N	指定分区关键字 CASE_N 的迁移方式。Gauss 不支持多级（嵌套）分区。 支持以下参数值： • comment • none	• comment • none	comment	tdMigrateCASE_N=comment
• tdMigrateRANGE_N	指定分区关键字 RANGE_N 的迁移方式。Gauss 不支持多级（嵌套）分区。 支持以下参数值： • comment • none • range	• comment • none • range	range	tdMigrateRANGE_N=range
• tdMigrateAddMonth	是否支持迁移 addMonth。 支持以下参数值： • true • false 若该参数设为 true，则迁移后为 mig_td_ext.ADD_MONTHS（添加 mig_td_ext）。否则，不支持迁移。	• true • false	false	tdMigrateAddMonth=false

6.6.4 Oracle SQL 配置

设置 Oracle 配置参数可在迁移 Oracle 数据库脚本时自定义迁移工具的行为。

打开 config 文件夹中的 features-oracle.properties 文件，并根据实际需要设置表 6-8 中的参数。

表6-8 features-oracle.properties 文件中的配置参数

参数	说明	取值范围	默认值	样例
• exceptionHandler	指定 PL/SQL 中的异常块是否必须被注释掉。 设为 True，表示必须被注释掉。 设为 False，表示保留原样。 说明 V100R002C60 版本不支持 exceptionHandler。	• true • false	false	exceptionHandler=TRUE
• TxHandler	指定 PL/SQL 中的提交或回退操作是否必须被注释掉。 设为 True，表示必须被注释掉。 设为 False，表示保留原样。	• True • False	True	TxHandler=True
• foreignKeyHandler	设置 DSC 对 Foreign Key 约束的处理方法。 设为 true，可在迁移时将语句注释掉。 设为 false，可跳过迁移，原样复制语句。	• true • false	true	foreignKeyHandler=true
• globalTempTable	该参数的值支持 GLOBAL 和 LOCAL。目标数据库当前不支 GLOBAL。	• GLOBAL • LOCAL	LOCAL	encodingFormat=LOCAL
• onCommitDeleteRows	该参数的值支持 DELETE 和 PRESERVE。当前版本 V100R008	• DELETE • PRESERVE	DELETE	onCommitDeleteRows=DELETE
• maxValInSequ	设置数据库可以	1-	922337203	maxValInSequen

参数	说明	取值范围	默认值	样例
ence	支持的最大序列值。目前，数据库支持的最大值是92233720368547 75807。	9223372036854775807	6854775807	ce=9223372036854775807
• mergeImplementation	设置 merge 语句迁移方法。 SPLIT：通过将 merge 语句拆分为单个查询进行优化。 WITH：使用 WITH 子句来迁移整个 merge 语句。	• WITH • SPLIT • None	WITH	mergeImplementation=None
• RemoveHashPartition	设置 DSC 对 HASH PARTITION 语句的处理方法。 设为 true，可在迁移时将语句注释掉。 设为 false，可跳过迁移，原样复制语句。	• true • false	true	RemoveHashPartition=false
• RemoveHashSubPartition	设置 DSC 对 HASH SUBPARTITION 语句的处理方法。 设为 true，可在迁移时将语句注释掉。 设为 false，可跳过迁移，原样复制语句。	• true • false	true	RemoveHashSubPartition=false
• RemoveListPartition	设置 DSC 对 LIST PARTITION 语句的处理方法。 设为 true，可在迁移时将语句注	• true • false	true	RemoveListPartition=false

参数	说明	取值范围	默认值	样例
	释掉。 设为 false，可跳过迁移，原样复制语句。			
• RemoveListSubPartition	设置 DSC 对 LIST SUBPARTITION 语句的处理方法。 设为 true，可在迁移时将语句注释掉。 设为 false，可跳过迁移，原样复制语句。	• true • false	true	RemoveListSubPartition=false
• RemoveRangeSubPartition	设置 DSC 对 RANGESUBPARTITION 语句的处理方法。 设为 true，可在迁移时将语句注释掉。 设为 false，可跳过迁移，原样复制语句。	• true • false	true	RemoveRangeSubPartition=false
• MigSupportSequence	设置 DSC 对 SEQUENCE 语句的处理方法。 设为 true，可将 CREATE 脚本转换为 INSERT 脚本。 设为 false，则无法迁移 CREATE 脚本。	• true • false	true	MigSupportSequence=false
• RemovePartitionTS	设置是否注释掉 PartitionTS 语句。 设为 true，注释 PartitionTS 语句。 设为 false，原样	• true • false	true	RemovePartitionTS=true

参数	说明	取值范围	默认值	样例
	保留 PartitionTS 语句。			
• BitmapIndexSupport	设置是否注释掉 BitmapIndex。 设为 COMMENT，注释 BitmapIndex。 设为 BTREE，原样保留 BitmapIndex。	• comment • btree	comment	BitmapIndexSupport=comment
• pkgSchemaNaming	支持以下参数值： 设为 true，将 schema1.package1#procedure1 迁移为 package1.procedure1。 设为 false，保留 schema1.package1#procedure1。	• true • false	true	pkgSchemaNaming=true
• plsqlCollection	支持以下参数值： • varray • localtable • none	• varray • localtable • none	varray	plsqlCollection=varray
• commentstorageparameter	是否注释掉表或索引中的存储参数。 设为 true，注释存储参数。 设为 false，原样保留存储参数。	• true • false	true	commentStorageParameter=true
• MigSupportForListAgg	是否迁移 ListAgg 语句。 设为 true，迁移 ListAgg 语句。 设为 false，不迁移 ListAgg 语句。	• true • false	true	MigSupportForListAgg=false

参数	说明	取值范围	默认值	样例
• MigSupportForRegexReplace	是否迁移RegexReplace 语句。 设为 true，迁移RegexReplace 语句。 设为 false，不迁移 RegexReplace 语句。	• true • false	true	MigSupportForRegexReplace=false
• commentPragmaAutomationTrans	是否注释掉表或索引中的AutomationTrans。 设为 true，注释掉AutomationTrans。设为 false，原样保留AutomationTrans。	• true • false	true	commentPragmaAutomationTrans=true
• supportJoinOperator	是否支持左/右外连接操作符（+）。 支持以下参数值： • true • false	• true • false	false	supportJoinOperator=false
• migInsertWithTableAlias	支持以下参数值： • true • false	• true • false	true	migInsertWithTableAlias=true
• varraySize	设置 Varray 数据类型的大小。	NA	1024	varraySize=1024
• varrayObjectSize	设置 Varray 对象数据类型的大小。	NA	10240	varrayObjectSize = 10240
• migrationScope	设置迁移方法：分包或整体迁移。	• pkgSplit • completeMigration	completeMigration	migrationScope=completeMigration
• migSupportConnectBy	支持以下参数	• true	true	migSupportConnectBy = true

参数	说明	取值范围	默认值	样例
	值： • true • false 设为 true，迁移 connectBy。	• false		
• migrate_ConnectBy_Unnest	是否迁移 migrate_ConnectBy_Unnest。 设为 true，迁移 connectBy 和 Unnest。 设为 false，原样保留。	• true • false	true	migrate_ConnectBy_Unnest=true
• extendedGroupByClause	是否迁移以下 GROUP BY 扩展功能： • grouping sets • cube • rollup	• true • false	false	extendedGroupByClause=false
• supportDupValOnIndex	是否迁移 DUP_VAL_ON_INDEX。	• UNIQUE_VIOLATION(V1R8) • OTHERS(older versions)	UNIQUE_VIOLATION	supportDupValOnIndex=UNIQUE_VIOLATION
• pkgvariable	是否迁移 pkgvariable。	• localtable • sys_set_context • none	localtable	pkgvariable = localtable
• addPackageNameList	支持以下参数值： • true • false 设置 package_name_list=<schema_name>，然后调用该模式。	• true • false	false	addPackageNameList = false
• addPackageTag	支持以下参数值：	• true • false	false	addPackageTag = true

参数	说明	取值范围	默认值	样例
	• true • false 设为 true，会在存储过程/函数声明中的 AS|IS 前面增加 PACKAGE。			
• addGrantLine	支持以下参数值： • true • false 设为 true，会在文件末尾的每个存储过程/函数前面增加 GRANT 行。	• true • false	false	addGrantLine = true
• MigDbmsLob	支持以下参数值： • true • false 设为 true，迁移 DBMS_LOB。 设为 false，不迁移 DBMS_LOB。	• true • false	false	MigDbmsLob=false
• uniqueConsForPartitonedTable	设置分区表的 unique 或主键约束。	• comment_partition • comment_unique • none	comment_partition	uniqueConsForPartitonedTable = comment_partition
• MigSupportForRegexFunc	MigSupportForRegexReplace 参数的可能取值。	• true • false	false	MigSupportForRegexFunc=false
• migSupportUnnest	migSupportUnnest 的可能取值。	• true • false	true	migSupportUnnest = true
• MDSYS.MBRCOORDLIST	将不支持的数据类型 MDSYS.MBRCOORDLIST 替换为用户定义的数据类型。	• None • Can enter any free text	None	MDSYS.MBRCOORDLIST=None

参数	说明	取值范围	默认值	样例
• MDSYS.SDO_GEOMETRY	将不支持的数据类型 MDSYS.SDO_GEOMETRY 替换为用户定义的数据类型。	• None • Can enter any free text	None	MDSYS.SDO_GEOMETRY=None
• GEOMETRY	将不支持的数据类型 GEOMETRY 替换为用户定义的数据类型。	• None • Can enter any free text	None Input should not migrate.	GEOMETRY=None
• tdMigrateAddMonth	DSC 支持 addMonth.可能的值。	• true • false	None	tdMigrateAddMonth=false IF TRUE THEN mig_ORA_ext.ADD_MONTHS (APPENDING mig_ORA_ext) OTHERWISE NOT APPEND. tdMigrateAddMonth=false

说明

DSC 提供了用于删除 PARTITIONS 和 SUBPARTITIONS 的参数，因为该工具暂时无法完全支持这些关键词。用户可以选择在迁移脚本时将这些参数注释掉，也可以选择原样保留。

6.6.5 Teradata Perl 配置

设置 Teradata Perl 配置参数可在迁移 Teradata Perl 数据库脚本时自定义迁移工具的行为。

打开 config 文件夹中的 perl-migration.properties 文件，并根据实际需要设置表 6-9 中的参数。

说明

- 参数值不区分大小写。
- 用户可以更改下表中 db-bteq-tag-name 和 db-tdsql-tag-name 的参数值。

表6-9 perl-migration.properties 文件中的配置参数

参数	说明	取值范围	默认值	样例
• db-bteq-tag-	指定要在 Perl 文	• bteq	bteq	db-bteq-tag-

参数	说明	取值范围	默认值	样例
name	件中处理的脚本。 BTEQ：仅处理 BTEQ 标记的脚本。			name=bteq
• db-tdsql-tag-name	指定待处理的脚本。 SQL_LANG：仅处理 SQL_LANG 标记的脚本。	sql_lang	sql_lang	db-tdsql-tag-name=sql_lang
• add-timing-on	指定是否通过添加脚本来计算执行时间。 如果启用，则为每个输入文件添加 add timing 脚本。	• true • false	false	add-timing-on=true
• remove-intermediate-files	指定在迁移完成后是否删除工具创建的中间 SQL 文件。 该文件包含 SQL 文件中的 BTEQ 和 SQL_LANG 语法，作为语法迁移工具的输入文件。 设为 true，表示删除中间文件。 设为 false，表示不删除中间文件。	• true • false	true	remove-intermediate-files=true
• migrate-variables	指定是否迁移分配给 Perl 变量的 SQL 命令。 Perl 文件可以包含分配了 SQL 命令的 Perl 变量，通过 PREPARE 和 EXECUTE 语句在 Perl 中执行。该工具还可	• true • false	true	migrate-variables=true

参数	说明	取值范围	默认值	样例
	以从 Perl 变量提取 SQL 命令进行迁移。 设为 true，表示对分配给 Perl 变量的 SQL 命令进行迁移。 设为 false，表示在迁移时跳过该 Perl 变量。 示例 1： 当参数设为 true 时，将 `$V_SQL = "CT X1(C1 INT,C2 CHAR(30))";` 迁移为： `CREATE TABLE X1(C1 INT,C2 CHAR(30));` 示例 2： **输入** `$onesql ="SELECT trim(tablename) from dbc.tables WHERE databasename = '${AUTO_DQDB}' and tablename like 'V_%' order by 1;"; $sth_rundq = $dbh->execute_query($onesql);` **输出** `$onesql ="SELECT TRIM( tablename ) FROM dbc.tables WHERE`			

参数	说明	取值范围	默认值	样例
	databasename = '${AUTO_DQDB}' AND tablename LIKE 'V_%' ORDER BY 1 ; "; $sth_rundq = $dbh->execute_query($onesql);			
• logging-level	指定 Perl 迁移日志的级别。 设为 error，表示仅记录错误日志。 设为 warning，表示记录错误及告警日志。 设为 info，表示记录错误、告警和活动日志。该级别包含所有日志信息。	• error • warning • info	info	logging-level=info
• log-file-count	指定保留日志文件的最大数量。文件总数包括正在使用的日志文件和已归档的日志文件。 如果新归档的日志文件超过了文件数上限，则会先删除最早保留的文件，直到成功保存指定数量的文件。	3 - 10	5	log-file-count=10
• log-file-size	指定日志文件的最高上限。 日志文件大小达到指定上限时，给文件名添加时	1MB - 10MB	5MB	log-file-size=10MB

参数	说明	取值范围	默认值	样例
	间戳并进行归档。 **示例：** `perlDSC_2018-07-08_16_12_08.log` 归档后，生成新的日志文件perlDSC.log。			
• migrate-executequery	指定是否迁移包含 SQL 内容的execute_query。 设为 true，表示迁移。 设为 false，表示不迁移。 示例： 设为 true 时，将 `my $rows1=$conn1->execute_query("sel ${selectclause} from ${databasename}.${tablename};");` 迁移为： `my $rows1=$conn1->execute query("SELECT ${selectclause} FROM ${databasename}.${tablename};");`	• true • false	true	migrate-executequery=true

6.6.6 MySQL SQL 配置

设置 MySQL 配置参数可在迁移 MySQL 数据库脚本时自定义迁移工具的行为。

打开 config 文件夹中的 features-mysql.properties 文件，并根据实际需要设置表 6-10 中的参数。

表6-10 features-mysql.properties 文件中的配置参数

参数	说明	取值范围	默认值	样例
• table.databaseAsSchema • table.defaultSchema	是否使用数据库名称作为schema 名称，如果数据库名称不存在，则使用用户定义schema， 如果用户定义schema 为空，则使用默认schema。	• true • false • public	• true • public	• table.databaseAsSchema=true • table.defaultSchema=public
• table.schema	用户设置的schema 名称，如果该参数不为空，则使用该参数，即使包含useDatabaseAsSchema = true，也会使用当前 schema 名称。	• schemaName	• 默认为空	• table.schema=
• table.orientation	默认数据存储方式，ROW：行存储，COLUMN：列存储。	• ROW • COLUMN	• ROW	• table.orientation=ROW
• table.type	默认的表类型，分区表 or 复制表，HASH，REPLICATION。	• HASH • REPLICATION	• HASH	• table.type=HASH
• table.tablespace	表空间选项。	• COMMENT • RESERVE	• RESERVE	• table.tablespace=RESERVE
• table.partition-key.choose.strategy	分区键选择策略。	• partitionKeyChooserStrategy	• partitionKeyChooserStrategy	• table.partition-key.choose.strategy=partitionKeyChooserStrategy
• table.partitio	分区键设置，	• 预留参数	• 默认为空	• table.partitio

参数	说明	取值范围	默认值	样例
n-key.name	如果为空，则按照默认策略选择，如果有多列，用逗号分隔，忽略列名称大小写。			n-key.name=
• table.compress.mode	创建新表时，需要在 CREATE TABLE 语句中指定关键字 COMPRESS，这样，当对该表进行批量插入时就会触发压缩特性。该特性会在页范围内扫描所有元组数据，生成字典、压缩元组数据并进行存储。指定关键字 NOCOMPRESS 则不对表进行压缩。	• COMPRESS • NOCOMPRESS	• NOCOMPRESS	• table.compress.mode=NOCOMPRESS
• table.compress.row • table.compress.column	指定表数据的压缩级别，它决定了表数据的压缩比以及压缩时间。一般来讲，压缩级别越高，压缩比也越大，压缩时间也越长；反之亦然。实际压缩比取决于加载的表数据的分布特征。行存表的有效值为 YES/NO，默认值为 NO。列存表的有效值为 YES/NO/LOW/	• YES • NO • YES • NO • LOW • MIDDLE • HIGH	• NO • LOW	• table.compress.row=NO • table.compress.column=LOW

参数	说明	取值范围	默认值	样例
	MIDDLE/HIGH，默认值为LOW。			
• table.compress.level	指定表数据同一压缩级别下的不同压缩水平，它决定了同一压缩级别下表数据的压缩比以及压缩时间。对同一压缩级别进行了更加详细的划分，为用户选择压缩比和压缩时间提供了更多的空间。总体来讲，此值越大，表示同一压缩级别下压缩比越大，压缩时间越长；反之亦然。	• 0 • 1 • 2 • 3	• 0	• table.compress.level=0
• table.database.template	数据库模板。	• 预留参数	• template0	table.database.template=template0

6.6.7 Netezza 配置

设置 Netezza 配置参数可在迁移 Netezza 数据库脚本时自定义迁移工具的行为。

打开 config 文件夹中的 features-netezza.properties 文件，并根据实际需要设置表 6-11 中的参数。

表6-11 features-netezza.properties 文件中的配置参数

参数	说明	取值范围	默认值	样例
• rowstoreToColumnstore	是否将 rowstore 迁移为 columnstore。	• true • false	false	rowstoreToColumnstore=false
• cstore_blob	cstore_blob 取值如下：	• bytea • none	bytea	cstore_blob=bytea

参数	说明	取值范围	默认值	样例
	• bytea • none			
• keywords_addressed_using_as	关键字“addressed_using_as”的取值如下： • OWNER • ATTRIBUTE • SOURCE • FREEZE	• OWNER • ATTRIBUTE • SOURCE • FREEZE	OWNER	keywords_addressed_using_as=OWNER
• keywords_addressed_using_doublequote	关键字“addressed_using_doublequote”的可能取值。	FREEZE	FREEZE	keywords_addressed_using_doublequote=FREEZE

6.7 使用 DSC

6.7.1 迁移流程

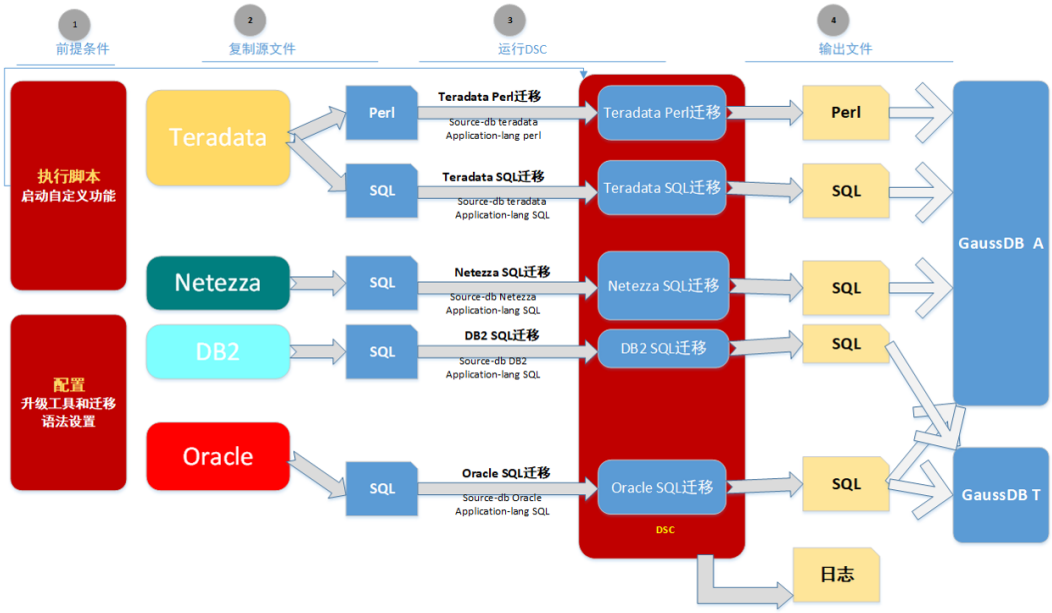
6.7.1.1 迁移流程概述

DSC 支持以下迁移场景：

- 迁移 Teradata SQL
- 迁移 Oracle SQL
- 迁移 Teradata Perl files
- 迁移 Netezza
- 迁移 MySQL SQL
- 迁移 DB2

DSC 迁移流程如图 6-3 所示。

图6-3 语法迁移流程



本节描述启动迁移过程前需要完成的前提条件。

执行自定义脚本

DSC 配置在 *DSC/scripts* 中包含如下自定义数据库脚本：

- *date_functions.sql* : Oracle 日期函数的自定义数据库脚本。
- *environment_functions.sql*：Oracle 环境函数的自定义数据库脚本。
- *string_functions.sql*：Oracle 字符串函数的自定义数据库脚本。
- *pkg_variable_scripts.sql*：Oracle 包变量函数的自定义数据库脚本。
- *sequence_scripts.sql*：Oracle 序列函数的自定义数据库脚本。
- *mig_fn_get_datatype_short_name.sql*：Teradata 函数的自定义数据库脚本。
- *mig_fn_castasint.sql* : CAST AS INTEGER 迁移的自定义数据库脚本。
- *vw_td_dbc_tables.sql*：DBC.TABLES 迁移的自定义数据库脚本。
- *vw_td_dbc_indices.sql*：DBC.INDICES 迁移的自定义数据库脚本。

需要通过这些脚本来支持目标数据库一个或多个版本中不存在的某些关键字。这些脚本在迁移之前必须在目标数据库中执行一次。

关于执行自定义数据库脚本的更多信息，请参考配置自定义数据库脚本。

使用以下任一方法在要执行迁移的所有目标 GaussDB(DWS)数据库中执行所需的脚本：

- 使用 **gsql** 连接到 GaussDB(DWS)数据库并将.sql 文件中的所有内容粘贴到 **gsql**，粘贴的内容将自动执行。

  执行如下命令连接到 GaussDB(DWS)数据库：

```
gsql -h <host_addr_xxx.xxx.xxx.xxx> -d <database_name> -U <user_name> -W
<password> -p <port_number> -r
```

- 使用 **gsql** 连接到 GaussDB(DWS)数据库并执行.sql 文件。

  执行如下命令连接到 GaussDB(DWS)数据库并执行.sql 文件：

```
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
```

- 使用 Data Studio 连接到 GaussDB(DWS)数据库，然后在 Data Studio 中打开并执行.sql 文件。
- 在迁移 Oracle PL/SQL（存储过程/函数）对象之前，先通过 Bulk 类型迁移所有 DDL 及 DML 命令，然后再通过 BLogic 类型迁移 PL/SQL 对象。

说明

如果迁移类型为 Bulk，则输入文件不允许包含 PL/SQL 对象。

同样，如果迁移类型为 BLogic，则输入文件不允许包含 DDL/DML 命令。

配置 DSC 和迁移属性

DSC 在 *DSC/config* folder 中包含如下配置文件：

- *application.properties*: DSC 的配置参数。
- *features-teradata.properties*: Teradata SQL 迁移的配置参数。
- *features-oracle.properties:* Oracle SQL 迁移的配置参数。
- *oracle-migration.properties:* Oracle (Beta) SQL 迁移的配置参数。
- *perl-migration.properties:* Perl 迁移的配置参数。
- *features-netezza.properties*: Netezza 迁移的配置参数。
- *features-mysql.properties:* MySQL SQL 迁移的配置参数。

更新配置参数的更多信息，请参考 6.6.2 DSC 配置,。

6.7.1.2 前提条件

执行自定义数据库脚本

执行数据库自定义脚本是为了支持目标数据库某些版本中不存在的关键字。这些脚本在迁移之前需在目标数据库中执行一次。

DSC/scripts 目录中的自定义脚本如表 6-12 所示。有关如何执行自定义脚本的详细信息，请参见配置自定义数据库脚本。

表6-12 自定义数据库脚本

自定义脚本	说明
date_functions.sql	Oracle 日期函数的自定义数据库脚本
environment_functions.sql	Oracle 环境函数的自定义数据库脚本

自定义脚本	说明
string_functions.sql	Oracle 字符串函数的自定义数据库脚本
pkg_variable_scripts.sql	Oracle 软件包变量函数的自定义数据库脚本
sequence_scripts.sql	Oracle 序列函数的自定义数据库脚本
mig_fn_get_datatype_short_name.sql	Teradata 函数的自定义数据库脚本
mig_fn_castasint.sql	用于迁移 CAST AS INTEGER 的自定义数据库脚本
vw_td_dbc_tables.sql	用于迁移 DBC.TABLES 的自定义数据库脚本
vw_td_dbc_indices.sql	用于迁移 DBC.INDICES 的自定义数据库脚本

表6-13 自定义数据库脚本 (Oracle 至 GaussDB T)

自定义脚本	说明
create_user_and_temptable_enable.sql	Oracle 软件包的创建用户和创建本地临时表的自定义数据库脚本
pkg_variable_scripts.sql	Oracle 包变量函数的自定义数据库脚本

按照以下步骤执行自定义数据库脚本：

步骤 1 通过以下任一方法在要执行迁移的所有目标数据库中执行所需脚本：

- 使用 **gsql**。
  - 使用 **gsql** 连接到目标数据库并将 SQL 文件中的所有内容粘贴到 **gsql**，粘贴的内容将自动执行。

    执行以下命令连接到数据库：

    ```
    gsql -h <host_addr_xxx.xxx.xxx.xxx> -d <database_name> -U <user_name> -W <password> -p <port_number> -r
    ```

  - 使用 **gsql** 连接到目标数据库并执行 SQL 文件。

    执行以下命令连接到数据库并执行 SQL 文件：

    ```
    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
    ```

- 使用 Data Studio。

  使用 Data Studio 连接到目标数据库，然后在 Data Studio 中打开并执行 SQL 文件。

----结束

配置自定义数据库脚本

用户可以使用自定义数据库的 SQL 脚本从 Teradata/Oracle 迁移那些不直接存在于目标数据库的关键字。

迁移之前，这些脚本必须在每个目标数据库中执行一次。

打开发布包中的 scripts 文件夹，文件目录如表 6-14 所示。

SQL 文件包含自定义迁移函数。GaussDB(DWS)数据库需要通过这些函数支持 Teradata/Oracle 的具体特性。

表6-14 DSC 自定义数据库脚本

文件夹	脚本文件	描述
-- scripts	-	文件夹：所有脚本
------ oracle	-	文件夹：Oracle 函数和脚本
-------- sequence	-	文件夹：配置 Oracle 序列的脚本
-	sequence_scripts.sql	脚本：启动 Oracle 序列的迁移。详情请参见 6.9.2.6 序列。
-------- package	-	文件夹：配置 Oracle 包变量的脚本
-	pkg_variable_scripts.sql	脚本：启动 Oracle 安装包变量的迁移。详情请参见 6.9.17.2 包变量。
-------- function	-	文件夹：配置 Oracle 系统函数的脚本
-	date_functions.sql	脚本：启动 Oracle 日期函数的迁移
-	environment_functions.sql	脚本：启动 Oracle 环境函数的迁移
-	string_functions.sql	脚本：启动 Oracle 字符串函数的迁移
------ teradata	-	文件夹：Teradata 函数和脚本
-------- view	-	文件夹：配置视图的脚本

文件夹	脚本文件	描述
-	vw_td_dbc_tables.sql	脚本：启动 Teradata 中 DBC.TABLES 的迁移
-	vw_td_dbc_indices.sql	脚本：启动 Teradata 中 DBC.INDICES 的迁移
-------- function	-	文件夹：配置 Teradata 系统函数的脚本
-X	mig_fn_get_datatype_short_name.sql	脚本：启动 Teradata 中 DBC.COLUMNS 的迁移
-	mig_fn_castasint.sql	脚本：启动 CAST AS INTEGER 的迁移
--------db_scripts	-	文件夹：启动 Teradata 自定义函数的脚本
-	mig_fn_get_datatype_short_name.sql	脚本：启动 Teradata 中 DBC.COLUMNS 的迁移
--------core	-	文件夹：Teradata 关键脚本
-	teradatacore.pm	脚本：执行 Perl 迁移的脚本

配置 DSC 和迁移属性

DSC 配置涉及 DSC/config 目录中的配置文件，请根据表 6-15 配置对应的参数。

表6-15 DSC 配置参数

迁移场景	配置文件	配置参数
6.7.2 Teradata SQL 迁移	• DSC：*application.properties* • 6.6.3 Teradata SQL 配置：*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**

迁移场景	配置文件	配置参数
		temporary/unlogged
6.7.3 Oracle SQL 迁移	• DSC：*application.properties* • 6.6.4 Oracle SQL 配置：*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
6.7.4 Teradata Perl 迁移	• DSC：*application.properties* • 6.6.5 Teradata Perl 配置：*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
6.7.6 MySQL SQL 迁移	• DSC：*application.properties* • 6.6.6 MySQL SQL 配置：*features-mysql.properties*	table.databaseAsSchema=true table.defaultSchema=public table.schema= table.orientation=ROW table.type=HASH table.partition-key.choose.strategy=com.partitionKeyChooserStrategy table.partition-key.name= table.compress.mode=NOCOMPRESS table.compress.level=0 table.compress.row=NO table.compress.column=LOW table.database.template=template0

迁移场景	配置文件	配置参数
6.7.5 Netezza SQL 迁移	• DSC: application.properties • 6.6.7 Netezza 配置: features-netezza.properties	`rowstoreToColumnstore=false`
6.12 DB2 语法迁移	• 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 准备工作

在迁移之前必须先创建输入文件夹和输出文件夹，并将待迁移的所有 SQL 脚本复制到输入文件夹中。Linux 系统操作如下：

步骤 1 创建输入和输出文件夹。您可以根据用户的首选项在任意位置创建文件夹。用户也可以使用默认的文件夹作为输入、输出，作为包的一部分提供。

```
mkdir input
mkdir output
```

危险

由于 DSC 批量无序地读取输出文件夹，因此，建议在迁移开始后不要对输入文件夹和文件进行任何修改，这些异常操作将影响 DSC 的输出结果。

步骤 2 将所有待迁移的 SQL 文件复制到输入文件夹。

说明

- 如果源文件的编码格式不是 UTF-8，请执行以下步骤：

- 打开 config 文件夹中的 application.properties 文件。

1. 将 application.properties 文件中 encodingFormat 参数值修改为所需的文件编码格式。

   DSC 支持 UTF-8、ASCII 以及 GB2312 编码格式。encodingFormat 的值不区分大小写。

- 如果需要获取 Linux 系统中源文件的编码格式，请在源文件所在服务器上执行以下命令：

  ```
  file -bi <Input file name>
  ```

----结束

6.7.1.4 使用 DSC 迁移

注意事项

- 启动迁移程序前，必须指定输出文件夹路径。输入文件夹路径、输出文件夹路径以及日志路径以空格隔开。输入文件夹路径不能包含空格。路径空格会导致 DSC 执行错误。详情请参见 6.15 DSC 故障处理。
- 如果输出文件夹中包含子文件夹或文件，DSC 会在执行迁移前将其删除或者根据用户设置（config 文件夹中 application.properties 配置文件）将其覆盖。已删除或覆盖的子文件夹或文件无法通过 DSC 恢复。
- 如果在同一台服务器上并发进行迁移（由同一个或不同 DSC 执行），不同的迁移任务必须使用不同的输出文件夹路径和日志路径。
- 用户可以通过可选参数指定日志存储路径。如果路径未指定，DSC 在 TOOL_HOME 下自动创建 log 文件夹。详情请参见 6.14 日志参考。

迁移方法

用户可在 Windows 和 Linux 操作系统中执行 **runDSC.sh** 或 **runDSC.bat** 命令进行迁移，各迁移场景的命令详见表 6-16。

表6-16 Windows 和 Linux 场景迁移

迁移场景	命令行参数
6.7.2 Teradata SQL 迁移	**> ./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]
6.7.3 Oracle SQL 迁移	**./runDSC.sh** **--source-db** Oracle

迁移场景	命令行参数
	[--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]
6.7.4 Teradata Perl 迁移	> ./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]
6.7.6 MySQL SQL 迁移	> ./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]
6.7.5 Netezza SQL 迁移	> ./runDSC.sh --source-db Netezza

迁移场景	命令行参数
	[--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]
6.12 DB2 语法迁移	**> ./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**]

说明

- 命令行参数说明：
- source-db 指定源数据库，参数值为 Teradata、Oracle，不区分大小写。
- conversion-type 指定迁移类型，为可选参数。DSC 支持以下迁移类型：

  -Bulk：迁移 DML 和 DDL 脚本。

  -BLogic：迁移业务逻辑，如存储过程和函数。BLogic 适用于 Oracle PL/SQL 及 Netezza。
- target-db 指定目标数据库，参数值为 GaussDB(DWS)。
- 命令回显说明：

  Migration process start time 和 Migration process end time 分别表示迁移开始时间和结束时间。Total process time 表示迁移总时长，单位为 ms。此外，迁移文件总数、处理器总数、已使用处理器数量、日志文件路径以及错误日志文件路径也会显示在控制台上。
- 关于命令行参数的详细信息，请参考 6.13.1 数据库模式迁移。

任务示例

- 示例：将 Oracle 数据库的 SQL 文件迁移到适用于 Linux 系统下的 GaussDB(DWS) 的 SQL 脚本中。

  ```
  ./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
  ```

- 示例：执行以下命令，将 Oracle 数据库的 SQL 文件迁移到适用于 Windows 操作系统下的 GaussDB(DWS)的 SQL 脚本中。

  ```
  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
  ```

控制台上显示迁移详情（包括进度和完成状态）：

```
********************** 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 查看输出文件和日志

查看并验证输出文件

迁移流程结束后，用户可使用对比工具（例如 BeyondCompare®）将输入文件与输出文件进行比较。为了简化对比过程，也可以先对源 SQL 文件进行 6.7.7 SQL Formatter。

1. 在 Linux 操作系统上运行以下命令以查看输出文件夹中的迁移文件。Windows 操作系统不再赘述。

   ```
   cd OUTPUT
   ls
   ```

   显示类似以下信息：

   ```
   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. 使用对比工具比较输入文件和输出文件，查看迁移后 SQL 文件的关键字是否符合目标数据库的要求。如果不符合，请联系技术工程师支持处理。

查看日志文件

所有执行及错误信息都会写入对应的日志文件。详情请参见 6.14 日志参考。

检查日志文件是否记录错误信息。如是，请参考 6.7.1.6 故障处理（DSC 迁移）。

6.7.1.6 故障处理（DSC 迁移）

迁移问题可分为：

- 工具执行问题：由于工具部分或全部执行失败导致的无输出或输出不正确的问题。要了解更多遗留问题及其解决方案，请参见 6.15 DSC 故障处理和 6.16 DSC 常见问题。
- 迁移语法问题：由于迁移工具无法正确识别或迁移语法的问题。要了解更多遗留问题，请参见 6.3 DSC 约束和限制。

6.7.2 Teradata SQL 迁移

工具支持从 Teradata 到 GaussDB(DWS)的迁移，包括模式、DML、查询、系统函数、类型转换等。

执行 Teradata SQL 迁移

执行以下命令设置源数据库、输入和输出文件夹路径、日志路径和应用程序语言：

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]
```

以示例文件夹路径为例，命令如下：

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
```

在工具执行时，控制台上会显示迁移汇总信息，包括迁移进度和完成状态。执行信息和错误会录入 6.14.2 SQL 迁移日志。

```
********************** 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 ***********************
```

有关如何使用工具进行 Teradata SQL 迁移，请参见 6.7.1.4 使用 DSC 迁移。

说明

迁移过程中，输入脚本的元数据保存在以下文件中，允许迁移调用这些元数据：

- Teradata 迁移：

  teradata-set-table.properties

- Oracle 迁移：
- global-temp-table.properties

1. global-temp-tables.properties
2. primary-key-constraints.properties
3. package-definition.properties
4. package-names-oracle.properties
5. create-types-UDT.properties

以下迁移场景时，需要清空上述文件：

- 不同文件的迁移
- 相同文件的迁移，但是参数配置不同

6.7.3 Oracle SQL 迁移

工具支持从 Oracle 到 GaussDB(DWS)的迁移，包括模式、DML、查询、系统函数、PL/SQL 等。

执行 Oracle SQL 迁移

执行以下命令设置源数据库、输入和输出文件夹路径、日志路径、应用程序语言和迁移类型：

Linux 操作系统：

```
./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 操作系统：

```
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>]
```

- 迁移不含 PL/SQL 语句的普通 DDL 语句(表、视图、索引、序列等)时，应使用 Bulk 模式(即，将 conversion-type 参数设为 Bulk)。

  以示例文件夹路径为例，将 conversion-type 参数设为 Bulk，命令如下：

  Linux 操作系统：

  ```
  ./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 操作系统：

  ```
  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
  ```

  在工具执行时，控制台上会显示迁移汇总信息，包括迁移进度和完成状态。执行信息和错误会录入 6.14.2 SQL 迁移日志。

  ```
  ********************** 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 ************************
  ```

- 迁移含 PL/SQL 语句的函数、过程、包等对象时，应使用 BLogic 模式（即，将 conversion-type 参数设为 BLogic）。

  以示例文件夹路径为例，将 conversion-type 参数设为 BLogic，命令如下：

  ```
  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
  ```

  在工具执行时，控制台上会显示迁移汇总信息，包括迁移进度和完成状态。执行信息和错误会录入 6.14.2 SQL 迁移日志。

  ```
  ********************** 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 ************************
```

注意，应将普通 DDL 脚本和 PL/SQL 脚本分置不同输入文件夹下迁移。

Oracle PACKAGE 迁移注意事项

1. 应将包规范(即包头)与包体分置于不同文件、相同输入路径中进行迁移。

2. 应先使用 Bulk 模式迁移普通 DDL 语句（包含 PACKAGE 脚本中引用到的全部表结构信息），以在 config/create-types-UDT.properties 文件中形成字典信息。之后再使用 Blogic 模式迁移包规范（即包头）与包体。具体解释如下：

在部分 Oracle PACKAGE 定义包规范时，使用了"tbName.colName%TYPE"语法以基于其他表对象声明自定义的记录类型。

```
例如
CREATE OR REPLACE PACKAGE p_emp
AS
    --定义 RECORD 类型
    TYPE re_emp IS RECORD(
        rno emp.empno%TYPE,
        rname emp.empname%TYPE
    );

END;
```

GaussDB 暂不支持通过"tbName.colName%TYPE"语法在 CREATE TYPE 命令中指定列数据类型，DSC 工具在迁移时需要构建含有诸如 emp 表信息的数据库上下文环境。由此需要先使用 DSC 工具迁移所有的建表脚本(即使用 Bulk 模式迁移普通 DDL 语句)，DSC 内部会自动生成相应的数据字典。当含有各种表信息的上下文环境构建完成后，可以使用 Blogic 模式迁移 Oracle PACKAGE，此时 re_emp 记录类型会根据 emp 表的列类型完成迁移。

```
期望输出
CREATE TYPE p_emp.re_emp AS (
    rno NUMBER(4),
    rname VARCHAR2(10)
);
```

有关如何使用 DSC 进行 Oracle SQL 迁移，请参见 6.7.1.4 使用 DSC 迁移。

6.7.4 Teradata Perl 迁移

概述

本节描述 Teradata Perl 文件迁移过程的详细信息。

请使用 **runDSC.sh** 或 **runDSC.bat** 命令并设置**--application-lang=perl**，将 Perl 文件中的 Teradata BTEQ 或 SQL_LANG 脚本迁移到兼容 Perl 文件的 GaussDB(DWS)中。迁移 Perl 文件后，可使用对比工具比较输入和输出文件进行验证。

Perl 文件迁移流程如下：

1. 完成 6.7.1.2 前提条件中的步骤。
2. 创建输入文件夹，并将待迁移 Perl 文件复制到该文件夹。例如：/migrationfiles/perlfiles
3. 执行 DSC 迁移 Perl 脚本，并将 db-bteq-tag-name 设为 BTEQ 或 db-tdsql-tag-name 设为 SQL_LANG。
   a. DSC 从 Perl 文件中提取 BTEQ 或 SQL_LANG 类型脚本。
      i. BTEQ 是标签名称，包含一组 BTEQ 脚本，可以通过 perl-migration.properties 文件中的 db-bteq-tag-name 参数来配置。
      ii. SQL_LANG 也是标签名称，包含 Teradata SQL 语句，可以通过 db-tdsql-tag-name 参数来配置。
   b. DSC 通过调用 Teradata SQL 来迁移提取到的 SQL 脚本。有关 Teradata SQL 迁移的详细信息，请参见 6.8 Teradata 语法迁移。
   c. Perl 文件嵌入迁移后脚本。
4. 在指定的输出文件夹中创建迁移后的 Perl 文件。如果未指定输出文件夹，则工具会在输入文件夹内创建一个名为 converted 的输出文件夹，例如：/migrationfiles/perlfiles/converted。

说明

- 包含 SQL 命令的 Perl 变量也可以通过 migrate-variables 参数迁移为 SQL。
- Perl v 5.10.0 及以上提供兼容能力。

执行 Perl 迁移

要迁移 Perl 文件，请指定--source-db Teradata 和--application-lang Perl 参数值，然后执行迁移工具。该工具支持迁移 BTEQ 和 SQL_LANG 脚本。请配置 db-bteq-tag-name 或 db-tdsql-tag-name 参数指定待迁移脚本。

执行以下命令设置源数据库、输入和输出文件夹路径、日志路径和应用程序语言：

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>]
```

以示例文件夹信息为例，命令如下：

```
./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
```

工具执行时，控制台上会显示迁移汇总信息，包括进度和完成状态。

```
********************* 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 ***********************
```

有关 Perl 迁移的配置参数，详情请参见 6.6.5 Teradata Perl 配置。

有关命令行参数，详情请参见 6.13.1 数据库模式迁移。

说明

- DSC 对输入文件进行格式化，并将格式化后的文件归档到输出文件夹中。用户可以直接比对该输出文件和输出文件。
- 确保输入路径中没有空格。如果存在空格，则 DSC 会报错。详情请参见 6.15 DSC 故障处理。
- 日志详情请参见 6.14 日志参考。
- 如果输出文件夹中包含子文件夹或文件，DSC 会在执行迁移前将其删除或者根据用户设置（config 文件夹中 application.properties 配置文件）将其覆盖。已删除或覆盖的子文件夹或文件无法通过 DSC 恢复。
- Process start time 和 Process end time 分别表示迁移流程的开始和结束时间。Process total time 是 DSC 完成整个流程所用的总时间（以毫秒为单位）。在控制台上还显示已迁移文件的总数、用户配置的处理器总数、已使用的处理器数量以及日志文件和错误日志文件的路径。
- 设置 perl-migration.properties 文件中的--add-timing-on 参数为 true，通过添加自定义脚本来计算语句执行时间。

  示例：

  输入

  ```
  $V_SQL2 = "SELECT T1.userTypeInd FROM T07_EBM_CAMP T1  WHERE T1.Camp_List_Id
  = '$abc'";
  $STH = $dbh->prepare($V_SQL2);
  $sth->execute();
  @rows = $sth->fetchrow();
  ```

  输出

  ```
  $V_SQL2 = "SELECT T1.userTypeInd FROM T07_EBM_CAMP T1  WHERE T1.Camp_List_Id
  = '$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();
```

- --input-folder 中指定的文件和文件夹不得拥有 GROUP 和 OTHERS 的写权限，即，--input-folder 参数指定的文件夹权限不得高于 755。出于安全考虑，如果输入文件或文件夹具有写入权限，则不会执行 DSC。
- 在并发迁移场景下，每次迁移的输入路径必须是唯一的。

最佳实践

为优化 Perl 文件迁移，建议遵循如下标准：

- BTEQ 脚本采用以下格式：

```
print BTEQ <<ENDOFINPUT;
TRUNCATE TABLE employee;
ENDOFINPUT
close(BTEQ);
```

- SQL_LANG 脚本采用以下格式：

```
my $sSQL=<<SQL_LANG;
TRUNCATE TABLE employee;
SQL_LANG
```

- 注释不包括如下内容：
  - print BTEQ <<ENDOFINPUT
  - ENDOFINPUT
  - close(BTEQ)
  - my $sSQL=<<SQL_LANG
  - SQL_LANG

6.7.5 Netezza SQL 迁移

工具支持从 Netezza 到 GaussDB(DWS)的迁移，包括模式、DML、查询、系统函数、PL/SQL 等。

执行以下命令设置源数据库、输入和输出文件夹路径、日志路径、应用程序语言以及迁移类型：

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>]
```

以示例文件夹路径为例，命令如下：

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
```

在工具执行时，控制台上会显示迁移汇总信息，包括迁移进度和完成状态。执行信息和错误会录入 6.14.2 SQL 迁移日志。

```
********************* 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 ************************
```

有关如何使用 DSC 工具进行 Netezza SQL 迁移，请参见 6.7.1.4 使用 DSC 迁移。

6.7.6 MySQL SQL 迁移

工具支持从 MySQL 到 GaussDB(DWS)的迁移，包括模式、DML、查询、系统函数、PL/SQL 等。

在 LINUX 中执行 MySQL 迁移

在 Linux 中执行以下命令开始迁移。用户需指定源数据库、输入和输出文件夹路径和日志路径；应用程序语言类型可以是 SQL 或 Perl， 默认为 SQL；迁移类型，可以是 Bulk 或 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>]
```

在工具执行时，控制台上会显示迁移汇总信息，包括迁移进度和完成状态。

```
********************* 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 ************************
```

在 Windows 中执行 MySQL 迁移

在 Windows 中执行以下命令开始迁移。用户需指定源数据库、输入和输出文件夹路径和日志路径；应用程序语言类型可以是 SQL 或 Perl， 默认为 SQL；迁移类型，可以是 Bulk 或 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>]
```

在工具执行时，控制台上会显示迁移汇总信息，包括迁移进度和完成状态。

```
********************* 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 ************************
```

- 执行以下命令以迁移对象，如函数、存储过程和包含 PL/SQL 语句的包。

  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
  ```

在工具执行时，控制台上会显示迁移汇总信息，包括迁移进度和完成状态。执行信息和错误会录入 6.14.2 SQL 迁移日志。

```
********************* 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 用于提高 SQL 的可读性。它通过添加/删除行和上下文缩进来对 SQL 语句进行格式化。SQL Formatter 通常用于对迁移后的输出文件进行格式化，也可用于输入文件。

可以使用 formattedSourceRequired 参数来决定是否对源 SQL 文件使用 SQL Formatter。若该参数设为 true，则对输入文件的副本进行格式化并保存到*{输出路径}\formattedSource* 文件夹。

SQL Formatter 还支持 Teradata SQL 和 Oracle SQL 迁移。Teradata Perl 文件迁移中的 SQL 脚本也是格式化的。Oracle（Beta）工具不支持 SQL Formatter。

输入: SQL FORMATTER

```
select
p1.parti_encode               ,p1.accting_type_cd              ,p1.prod_code
      ,p1.cust_type_cd              ,p1.accting_amt_type_cd              ,p1.acctin
g_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              ,p
1.after effect crdt gl num             ,coalesce( p2.start dt ,cast( '30001231' as
date format 'yyyymmdd' ) )              ,p1.accting_term              ,p1.etl_job
from              (                 select                          rank
(                           start_dt asc                         ) as
start_dt_id                        ,parti_encode                          ,accting_ty
pe_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_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_ty
pe_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 ;
```

输出

```
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_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 (
                    ) 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 语法迁移

6.8.1 Teradata 迁移概述

本节列出了语法迁移工具支持的 Teradata 特性，并针对每一特性提供了 Teradata 语法及相应的 GaussDB 语法。通过本节所列语法可以了解 Teradata 脚本的内部迁移逻辑。

本节还可以作为数据库迁移团队的参考，作为客户现场验证 Teradata 脚本迁移的参考。

6.8.2 模式对象

本节主要介绍 Teradata 模式对象的迁移语法。迁移语法决定了关键字/特性的迁移方式。

数据库模式是数据库内部的数据结构。DSC 可快捷地将模式从 Teradata 迁移到 GaussDB(DWS)。

本节包括以下内容：

表迁移、索引迁移、视图迁移、COLLECT STATISTICS、ACCESS LOCK、DBC.COLUMNS、DBC.TABLES、DBC.INDICES、SHOW STATS VALUES SEQUENCED，具体内容详见 6.8.2.1 表迁移~6.8.3 SHOW STATS VALUES SEQUENCED 章节。

6.8.2.1 表迁移

输入文件中包含表的专用关键词 MULTISET VOLATILE，但 GaussDB(DWS)不支持该关键词。因此，DSC 在迁移过程中用关键词 LOCAL TEMPORARY/UNLOGGED 替换该关键词。请通过 session_mode 参数为 CREATE TABLE 语句设置默认表类型（SET/MULTISET）。

详细内容查看以下节点：

CREATE TALBE

CHARACTER SET 和 CASESPECIFIC

VOLATILE

SET

MULTISET

TITLE

索引

约束

COLUMN STORE

PARTITION

ANALYZE

数据类型

支持指定部分列

CREATE TALBE

Teradata 的 CREATE TABLE （缩写关键字为 CT）语句用于创建表。

示例：

输入：CREATE TABLE

```
CT tab1 (
    id INT
);
```

输出

```
CREATE
    TABLE
        tab1 (
            id INTEGER
        )
;
```

执行 CREATE tab2 AS tab1 时，从 tab1 中复制表结构到 tab2。如果 CREATE TABLE 语句包含 WITH DATA 选项，则将 tab1 的数据也复制到 tab2 中。如果还包含 CREATE AS，则将源表中的约束行为保留到新表。

- 如果 session_mode 设为 Teradata，则必须删除目标表中的重复记录。该操作通过在迁移脚本中添加 MINUS 运算符实现。
- 如果 session_mode 设为 ANSI，则允许目标表中存在重复记录。

如果源表具有 PRIMARY KEY（主键）或 UNIQUE CONSTRAINT（唯一约束），则该表不包含任何重复记录。在这种情况下，不需要添加 MINUS 操作符删除重复的记录。

示例：

输入：CREATE TABLE AS WITH DATA（session_mode=Teradata）

```
CREATE TABLE tab2
   AS tab1 WITH DATA;
```

输出

```
BEGIN
   CREATE TABLE tab2 (
         LIKE tab1 INCLUDING ALL EXCLUDING PARTITION EXCLUDING RELOPTIONS
                 );

   INSERT INTO tab2
    SELECT * FROM tab1
         MINUS SELECT * FROM tab2;
END
;
/
```

输入：CREATE TABLE AS WITH DATA AND STATISTICS

```
CREATE SET VOLATILE TABLE tab2025
 AS ( SELECT * from tab2023 )
 WITH DATA AND STATISTICS
 PRIMARY INDEX (LOGTYPE, OPERSEQ);
```

输出

```
CREATE LOCAL TEMPORARY TABLE tab2025
 DISTRIBUTE BY HASH ( LOGTYPE, OPERSEQ )
 AS ( SELECT * FROM tab2023 );

 ANALYZE tab2025;
```

CHARACTER SET 和 CASESPECIFIC

CHARACTER SET 用于指定字符列的服务器字符集，CASESPECIFIC 指定字符数据比较及排序时的大小写情况。

可以使用 tdMigrateCharsetCase 参数来配置是否迁移 CHARACTER SET 和 CASESPECIFIC。如果该参数设为 false，则工具将跳过该查询的迁移并记录消息。

输入：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
;
```

输出

```
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)

;
```

输入：迁移支持的字符数据类型

在 Teradata 中，以下字符集支持以字符个数来衡量字符串数据类型的长度：

- LATIN
- UNICODE
- GRAPHIC

  不过，KANJISJIS 字符集支持以字节个数来衡量字符串数据类型的长度。

  以 COLUMN_NAME VARCHAR(100) CHARACTER SET UNICODE CASESPECIFIC COLUMN_NAME VARCHAR(100) CHARACTER SET LATIN CASESPECIFIC 为例，字符串最大支持 100 个字符（而不是字节）。

  在 GaussDB(DWS)中，字符串数据类型长度通过字节（而不是字符）来衡量。VARCHAR(100)和 VARCHAR2(100)最多支持 100 个字节（而不是字符）。 但是，NVARCHAR2(100)最大可支持 100 个字符。

  因此，如果 Teradata 使用 LATIN、UNICODE 或 GRAPHIC 字符集，VARCHAR 应迁移为 NVARCHAR。

```
CREATE TABLE tab1
(
    col1 VARCHAR(10),
    COL2 CHAR(1)
);
```

输出

```
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)
        ) ;
```

输入

```
CREATE TABLE tab1
(
    col1 VARCHAR(10) CHARACTER SET UNICODE,
    COL2 CHAR(1)
);
```

输出

```
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

输入文件中包含表的专用关键词 VOLATILE，但 GaussDB(DWS)不支持该关键词。因此，DSC 在迁移过程中用关键词 LOCAL TEMPORARY 替换该关键词。根据配置输入，Volatile 表在迁移中标记为本地临时表或无日志表。

输入：CREATE VOLATILE TABLE

```
 CREATE VOLATILE TABLE T1 (c1 int ,c2 int);
```

输出

```
CREATE
   LOCAL TEMPORARY TABLE
   T1 (
       c1 INTEGER
      ,c2 INTEGER
     )
;
```

输入：CREATE VOLATILE TABLE AS WITH DATA（session_mode=Teradata）

如果源表具有 PRIMARY KEY（主键）或 UNIQUE CONSTRAINT（唯一约束），则该表不包含任何重复记录。在这种情况下，不需要添加 MINUS 操作符删除重复的记录。

```
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;
```

输出

```
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 是 Teradata 的独有功能。它不允许重复的记录。它使用 MINUS 集合操作符来实现。DSC 支持 MULTISET 和 SET 表。SET 表支持与 VOLATILE 一起使用。

输入：SET TABLE

```
CREATE SET VOLATILE TABLE tab1 … ;
INSERT INTO tab1
SELECT expr1, expr2, …
  FROM tab1, …
 WHERE ….;
```

输出

```
CREATE LOCAL TEMPORARY TABLE tab1
… ; INSERT INTO tab1
 SELECT expr1, expr2, …
FROM tab1, …
WHERE ….
MINUS
SELECT * FROM tab1 ;
```

MULTISET

MULTISET 是一个普通表，所有数据库都支持这个表。迁移工具同时支持 MULTISET 和 SET 表。

MULTISET 表支持与 VOLATILE 一起使用。

输入：CREATE MULTISET TABLE

```
 CREATE VOLATILE MULTISET TABLE T1 (c1 int ,c2 int);
```

输出

```
CREATE
    LOCAL TEMPORARY TABLE
    T1 (
        c1 INTEGER
       ,c2 INTEGER
      )
;
```

TITLE

Teradata Permanent、Global Temporary 和 Volatile 表支持关键字 TITLE。在迁移过程中，TITLE 文本将被注释掉。

说明

如果 TITLE 文本拆分为多行，则在迁移后脚本中，换行符（ENTER）替换为空格。

输入：CREATE TABLE，使用 TITLE

```
CREATE TABLE tab1 (
  c1  NUMBER(2) TITLE 'column_a'
);
```

输出

```
CREATE TABLE tab1 (
  c1  NUMBER(2) /* TITLE 'column_a' */
);
```

输入：TABLE，使用多行 TITLE

```
CREATE TABLE tab1 (
  c1  NUMBER(2) TITLE 'This is a
very long title'
);
```

输出

```
CREATE TABLE tab1 (
  c1  NUMBER(2) /* TITLE 'This is a  very long title' */
);
```

输入：TABLE，使用列 TITLE

DSC 将列 TITLE 迁移为新的外部查询。

```
SELECT customer_id (TITLE 'cust_id')
FROM Customer_T
WHERE cust_id > 10;
```

输出

```
SELECT
        customer_id  AS "cust_id"
    FROM
        (
            SELECT
                    customer_id
                FROM
                    Customer_T
                WHERE
                    cust_id > 10
        )
;
```

输入：TABLE，使用列 TITLE 和 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;
```

输出

```
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 和 ALIAS

   如果使用 TITLE 并指定 ALIAS，则工具将按如下方式进行迁移：

- **TITLE with AS**：迁移为 AS alias。
- **TITLE with NAMED**：迁移为 NAMED alias。
- **TITLE with NAMED and AS**：迁移为 AS alias。

输入：TABLE TITLE，使用 NAMED 和 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 ;
```

输出

```
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
;
```

说明

目前，DSC 支持迁移初始 CREATE/ALTER 语句中的 TITLE 命令，但不支持后续对 TITLE 指定列的引用。例如，在下面的 CREATE TABLE 语句中，带有 TITLE Employee ID 的列 eid 在迁移后被注释掉，但是 SELECT 语句中对 eid 的引用将保持原样。

输入

```
CREATE TABLE tab1 ( eid INT TITLE 'Employee ID');
SELECT eid FROM tab1;
```

输出

```
CREATE TABLE tab1 (eid INT /*TITLE 'Employee ID'*/);
SELECT eid from tab1;
```

2. TITLE 和 CREATE VIEW

输入

```
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};
```

输出

```
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};
```

索引

CREATE TABLE 语句支持创建索引。DSC 支持带有主索引（PRIMARY INDEX）和唯一索引（UNIQUE INDEX）的 TABLE 语句。

该工具不会添加 DISTRIBUTE BY HASH 用于创建具有主键（PRIMARY KEY）和非唯一主索引的表。

输入：CREATE TABLE，使用 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 );
```

输出

```
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 );
```

说明

由于索引列列表（organic_name）不是 DISTRIBUTE BY 列列表（serial_no、organic_name）的超集，因此索引中删除了 UNIQUE。

输入：CREATE TABLE，使用主键和非唯一主索引（未添加 DISTRIBUTE BY HASH）

```
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 ) ;
```

输出

```
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 )
  )
;
```

约束

表中的约束应用于多列。DSC 支持以下约束：

- REFERENCES 约束/FOREIGN KEY：目前无法通过工具迁移。
- PRIMARY KEY 约束：可通过工具迁移。
- UNIQUE 约束：可通过工具迁移。

输入：CREATE TABLE，使用 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 ) ;
```

输出

```
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 );
```

输入

建表后，可使用 ALTER 语句为该表字段添加列级约束。

```
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) ;
```

输出

```
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) ;
```

说明

建表脚本中，需在所有列声明之后添加约束创建语法。

COLUMN STORE

表的存储方式可使用 CREATE TABLE 语句中的 WITH（ORIENTATION=COLUMN）从 ROW-STORE 转换为 COLUMN 存储。可使用 rowstoreToColumnstore 参数启用/禁用此功能。

输入：CREATE TABLE，修改存储模式为 COLUMN STORE

```
CREATE MULTISET VOLATILE TABLE tab1
     ( c1 VARCHAR(30) CHARACTER SET UNICODE
     , c2 DATE
     , ...
     )
 PRIMARY INDEX (c1, c2)
 ON COMMIT PRESERVE ROWS;
```

输出

```
CREATE LOCAL TEMPORARY TABLE tab1
    ( c1 VARCHAR(30)
    , c2 DATE
```

```
   , ...
   ) WITH (ORIENTATION = COLUMN)
 ON COMMIT PRESERVE ROWS
 DISTRIBUTE BY HASH (c1, c2);
```

PARTITION

工具不支持迁移分区/子分区，在迁移后脚本中注释掉以下分区/子分区的关键字：

- 范围分区/子分区
- 列表分区/子分区
- 哈希分区/子分区

场景 1：假设参数 tdMigrateCASE_N 和 tdMigrateRANGE_N 分别设置为 comment 和 range。

以下示例为 Teradata 建表脚本，指定嵌套分区。

输入：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 )
                        );
```

输出

```
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) );
```

场景 2：假设参数 tdMigrateCASE_N 和 tdMigrateRANGE_N 分别设置为 comment 和 range。

以下示例为 Teradata 建表脚本，指定嵌套分区。

输入

```
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 )
                       );
```

输出

```
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) );
```

场景 3：假设参数 tdMigrateCASE_N 和 tdMigrateRANGE_N 分别设置为非 comment 和 range 的值。

工具支持迁移，且不会注释掉分区语法。

输入

```
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 )
                        );
```

输出

```
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 )
                        ) */
;
```

场景 4：假设参数 tdMigrateCASE_N 和 tdMigrateRANGE_N 设为任意值。

以下示例为 Teradata 建表脚本，指定 RANGE_N 分区，未指定嵌套分区。

输入

```
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 )
                  );
```

输出

```
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) );
```

场景 5：假设参数 tdMigrateCASE_N 和 tdMigrateRANGE_N 分别设置为 comment 和 range。

以下示例为 Teradata 建表脚本，指定 CASE_N 分区，未指定嵌套分区。

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 )
                );
```

输出

```
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

输入：CREATE TABLE，使用 INDEX

```
CREATE TABLE EMP27 AS emp21 WITH DATA
PRIMARY INDEX (EMPNO) ON COMMIT PRESERVE ROWS;
```

输出

```
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);
```

数据类型

DSC 支持以下数据类型映射：

输入	输出
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

输入	输出
SMALLINT	SMALLINT
Character	**Character**
CHAR[(n)] / CHARACTER [(n)]	CHAR(n)
CLOB	CLOB
LONG VARCHAR	TEXT
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

示例：BYTEINT

输入

```
select cast(col as byteint) from tab;
```

输出

```
 SELECT CAST( col AS SMALLINT ) FROM tab ;
```

支持指定部分列

DSC 支持在执行 INSERT 期间指定部分列（非全部列）。当输入的 INSERT 语句不包含输入的 CREATE 语句中提到的所有列时会出现这种情况。在迁移时，会向这些列添加指定的默认值。

说明

session_mode 设为 Teradata 时支持此功能。

- INSERT-INTO-SELECT 中的 SELECT 语句不得包含以下内容:
- SET 操作符
- MERGE、使用 PERCENT 的 TOP、使用 TIES 的 TOP PERCENT

输入：TABLE，且 INSERT 语句中未指定 CREATE 中的全部列

```
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
;
```

输出

```
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 索引迁移

Teradata 中 CREATE INDEX 的列和表名的顺序和 GaussDB(DWS)中不同。使用参数 distributeByHash 配置数据在集群节点间的分布方式。该工具不会添加 DISTRIBUTE BY HASH 用于创建具有主键和非唯一主索引的表。

输入：主键非主索引的超集，且仅有 1 列匹配

```
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);
```

输出

```
CREATE TABLE     good_5 (
              column_1 INTEGER NOT NULL PRIMARY KEY
             ,column_2 INTEGER
             ,column_3 INTEGER NOT NULL
             ,column_4 INTEGER
         )
;
```

输入：主键非主索引的超集，且无匹配的列

```
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  );
```

输出

```
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   ) );
```

输入：不存在主键，且唯一索引有名称

```
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 );
```

输出

```
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 );
```

输入：CREATE TABLE，使用主键和非唯一主索引（未添加 DISTRIBUTE BY HASH）

```
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 ) ;
```

输出

```
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 视图迁移

CREATE VIEW （缩写关键字为 CV）和 SELECT 一同使用，用于创建视图。

Teradata 和 GaussDB(DWS)均支持关键词 VIEW，但 SELECT 语句在迁移过程中会用()括住。详情请参见下方图片。

通过 tdMigrateVIEWCHECKOPTIO....参数可以配置如何迁移包含 WITH CHECK OPTION 关键字的视图。如果该参数设置为 false，则工具跳过该查询并记录日志。

如果 CREATE VIEW 包含 LOCK 关键字，则工具根据 tdMigrateLOCKoption 的设置决定如何迁移 VIEW 查询。

输入：CREATE VIEW

```
CREATE VIEW DP_STEDW.MY_PARAM
AS
SELECT RUNDATE FROM  DP_STEDW.DATE_TBL   WHERE   dummy = 1;
```

输出

```
CREATE OR REPLACE VIEW DP_STEDW.MY_PARAM
AS
SELECT RUNDATE
  FROM  DP_STEDW.DATE_TBL
       WHERE   dummy = 1;
```

输入：CREATE VIEW，使用 FORCE 关键字

```
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';
```

输出

```
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

在 Teradata 中，REPLACE VIEW 语句用于创建新视图，或重建现有视图。DSC 将其迁移为 GaussDB(DWS)中兼容的 CREATE OR REPLACE VIEW 语句中。

输入：REPLACE VIEW

```
REPLACE VIEW DP_STEDW.MY_PARAM AS SELECT
        RUNDATE
    FROM
        DP_STEDW.DATE_TBL
    WHERE
        dummy = 1
;
```

输出

```
CREATE
OR REPLACE VIEW DP_STEDW.MY_PARAM AS (
    SELECT
            RUNDATE
        FROM
            DP_STEDW.DATE_TBL
        WHERE
            dummy = 1
)
;
```

输入：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);
```

输出

```
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

输入

```
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';
```

输出

```
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

通过 tdMigrateVIEWCHECKOPTIO...参数可以配置如何迁移包含 CHECK OPTION 关键字的视图。

如果源数据库中出现含有 CHECK OPRTION 关键词的视图，则工具在目标数据库中注释掉 CHECK OPRTION。

输入：VIEW，使用 CHECK OPTION

```
CV  mgr15 AS SEL *
FROM
   employee
WHERE
   manager_id = 15 WITH CHECK OPTION
;
```

输出（tdMigrateVIEWCHECKOPTION=True）

```
CREATE
    OR REPLACE VIEW mgr15 AS (
        SELECT
                *
            FROM
                employee
            WHERE
                manager_id = 15 /*WITH CHECK OPTION */
    )
;
```

输出（tdMigrateVIEWCHECKOPTION=False）

```
CV  mgr15 AS SEL *
FROM
   employee
WHERE
   manager_id = 15 WITH CHECK OPTION
;
```

VIEW WITH RECURSIVE

GaussDB(DWS)不支持 Teradata 关键词 RECURSIVE VIEW。因此，工具采用 VIEW WITH RECURSIVE 替代该关键词，如下图所示。

图6-4 输入视图：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
```

图6-5 输出视图

```
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

通过 tdMigrateLOCKOption 参数可以配置如何迁移包含 LOCK 关键字的查询。如果 tdMigrateLOCKOption 设置为 false，则该工具在迁移时将跳过该查询并记录日志。

输入：VIEW，使用 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 ) ;
```

输出

```
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

在 Teradata 中，COLLECT STAT 采集优化器统计信息，用于查询性能。GaussDB(DWS) 使用 ANALYZE 语句来替代 COLLECT STAT。

详情请参见 1。

输入：COLLECT STATISTICS

```
COLLECT STAT tab1 COLUMN (c1, c2);
```

输出

```
ANALYZE tab1 (c1, c2);
```

输入：COLLECT STATISTICS

```
COLLECT STATISTICS
  COLUMN (customer_id,customer_name)
, COLUMN (postal_code)
, COLUMN (customer_address)
ON customer_t;
```

输出

```
ANALYZE customer_t (
    customer_id
    ,customer_name
    ,postal_code
    ,customer_address
)
;
```

输入：COLLECT STATISTICS，使用 COLUMN

```
COLLECT STATISTICS
  COLUMN (
  Order_Date
  -- ,o_orderID
/*COLLECT
STATISTICS*/
  ,Order_ID
    )
ON order_t;
```

输出

```
ANALYZE order_t (
   Order_Date
    ,Order_ID
)
;
```

输入：COLLECT STATISTICS，使用 schemaname

```
COLLECT STATS COLUMN (
    empno
    ,ename
)
    ON ${schemaname}."usrTab1"
;
```

输出

```
ANALYZE ${schemaname}."usrTab1"
(
    empno
    ,ename
)
;
```

COLLECT STATISTICS

统计（COLLECT STATISTICS）基于抽样计算百分比。

输入：

```
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 ;
```

输出：

```
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 允许用户从可能已经锁定 READ 或 WRITE 的表中读取数据。

可以通过 tdMigrateLOCKoption 参数来配置如何对包含 LOCK 关键字的查询进行迁移。如果该参数设置为 false，工具将跳过该查询的迁移并记录日志。

输入：ACCESS LOCK（tdMigrateLOCKOption=True）

```
LOCKING TABLE tab1 FOR ACCESS
 INSERT INTO tab2
 SELECT …
   FROM …
  WHERE ...;
```

输出

```
/* LOCKING TABLE tab1 FOR ACCESS */
INSERT INTO tab2
SELECT …
  FROM …
 WHERE ...;
```

6.8.2.6 DBC.COLUMNS

DBC.COLUMNS 视图是一个表，包含有关表和视图列、存储过程、或宏参数的信息。其中包括以下列：DatabaseName、TableName、ColumnName、ColumnFormat、ColumnTitle、ColumnType、DefaultValue。在 GaussDB(DWS)中，这个表等效于 information_schema.columns 表。

说明

本特性要求一次性执行以下自定义脚本文件：
DSC/scripts/teradata/db_scripts/mig_fn_get_datatype_short_name.sql

有关文件执行的详细步骤，请参见 6.4 系统要求（DSC）和 6.7.1.2 前提条件。

迁移工具将以下 dbc.columns 列迁移为对应的 information_schema 列：

表6-17 dbc.columns 列迁移到 information_schema 列

dbc.columns	information_schema.columns
ColumnName	Column_Name
ColumnType	mig_fn_get_datatype_short_name (data_Type)
ColumnLength	character_maximum_length
DecimalTotalDigits	numeric_precision
DecimalFractionalDigits	numeric_scale
databasename	table_schema
tablename	table_name
ColumnId	ordinal_position

迁移 dbc.columns 时，假设以下条件成立：

- FROM 子句仅包含 dbc.columns 的 TABLE NAME。
- COLUMN NAME 为以下任一格式：column_name 或 schema_name.table_name.column_name。

以下场景不支持 dbc.columns 迁移：

- FROM 子句包含 dbc.columns 表名的别名（dbc.columns 别名）。
- dbc.columns 与其他表组合（FROM dbc.columns alias1，table1 alias2 OR dbc.columns alias1 join table1 alias2）。

说明

- 如果输入的 SELECT 语句直接包含 dbc.columns 的列名，则该工具会将输入的列名称迁移为别名。例如，输入列名称 DecimalFractionalDigits 会迁移为 numeric_scale，其别名为 DecimalFractionalDigits。

  示例：

  输入：

  ```
  SEL
        columnid
        ,DecimalFractionalDigits
      FROM
        dbc.columns
  ;
  ```

  输出：

  ```
  SELECT
        ordinal_position columnid
        ,numeric_scale DecimalFractionalDigits
      FROM
        information_schema.columns
  ;
  ```

- 关于表名和模式名称，迁移工具会将所有字符串值转换为小写。如果要区分大小写，使用双引号表示表/模式名称。在以下输入示例中，“Test”不会转换为小写。

  ```
  SELECT
          TableName
      FROM
          dbc . columns
      WHERE
          dbc.columns.databasename = '"Test"';
  ```

输入：dbc.columns table，指定所有支持列

```
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;
```

输出

```
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
;
```

输入：dbc.columns table，指定表名

```
SELECT
        TRIM( ColumnName )
        ,UPPER( dbc.columns.ColumnType )
    FROM
        dbc . columns
```

```
    WHERE
        dbc.columns.databasename = '"Test"'
    ORDER BY
        dbc.columns.ColumnId
;
```

输出

```
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

DSC 会将 dbc.tables 迁移为对应的 mig_td_ext.vw_td_dbc_tables。

示例：databasename 迁移为 mig_td_ext.vw_td_dbc_tables.schemaname。

输入

```
sel databasename,tablename FROM dbc.tables
WHERE tablekind='T' and trim(databasename) = '<dbname>'
AND
( NOT(TRIM(tablename) LIKE ANY  (<excludelist>))
);
```

输出

```
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 将 dbc.indices 迁移为对应的 mig_td_ext.vw_td_dbc_indices。

示例：databasename 迁移为 mig_td_ext.vw_td_dbc_tables.schemaname。

输入

```
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');
```

输出

```
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 > ] ) ) )
;
```

说明

在 dbc.indices 迁移过程中，查询应包含 AND indextype IN ('Q','P')。否则，工具不会迁移该查询，且会记录以下错误消息：

"Query/statement is not supported as indextype should be mentioned with values 'P' and 'Q'."

dbc.sessioninfoV

输入

```
select username,clientsystemuserid,clientipaddress,clientprogramname
  from dbc.sessioninfoV
 where sessionno = 140167641814784;
```

输出

```
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

输入

```
SELECT  username
,clientsystemuserid
 ,clientipaddress
,clientprogramname
```

```
 FROM
dbc.sessioninfo
WHERE
sessionno = lv_mig_session ;
```

输出

```
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，指定 FOR SESSION

输入

```
set query_band =
'AppName=${AUTO_SYS};JobName=${AUTO_JOB};TxDate=${TX_DATE};ScriptName=${script_name
};' for session ;
```

输出

```
set query_band =
'AppName=${AUTO_SYS};JobName=${AUTO_JOB};TxDate=${TX_DATE};ScriptName=${script_name
};' /* for session */;
```

SESSION

输入

```
select Session ;
should be migrated as below:
SELECT pg_backend_pid();
```

输出

```
SELECT pg_backend_pid();
```

6.8.3 SHOW STATS VALUES SEQUENCED

该命令显示 COLLECT STATISTICS 语句的结果以及相关统计信息，且 Gauss 无对应命令。考虑到该命令不影响功能，因此迁移时可直接注释掉。

输入

```
SHOW STATS VALUES SEQUENCED on "temp"."table"
```

输出

```
/*SHOW STATS VALUES SEQUENCED on "temp"."table"*/
```

6.8.4 DML（Teradata）

本节主要介绍 Teradata DML 的迁移语法。迁移语法决定了关键字/特性的迁移方式。

在 Teradata 中，如果某文件中包含 SELECT、INSERT、UPDATE、DELETE 和 MERGE 语句，则该文件中的 SQL 查询可迁移到 GaussDB(DWS)。

详见以下节点内容：

INSERT

SELECT

UPDATE

DELETE

MERGE

NAMED

ACTIVITYCOUNT

TIMESTAMP

INSERT

Teradata 的 INSERT（缩写关键字为 INS）语句用于向表中插入记录。DSC 支持 INSERT 语句。

Teradata SQL 中存在 INSERT INTO TABLE table_name 语法，但 GaussDB(DWS)不支持。GaussDB(DWS)仅支持 INSERT INTO table_name。DSC 工具需要去除关键词 TABLE。

输入

```
INSERT TABLE tab1
SELECT col1, col2
  FROM tab2
 WHERE col3 > 0;
```

输出

```
INSERT INTO tab1
SELECT col1, col2
  FROM tab2
 WHERE col3 > 0;
```

SELECT

1. **ANALYZE**

   Teradata 的 SELECT 命令（缩写关键字为 SEL）用于指定从哪一列中检索数据。

   在 GaussDB(DWS)中使用 ANALYZE 来收集优化器统计信息，这些统计信息将用于查询性能。

   输入：ANALYZE，使用 INSERT

   ```
   INSERT INTO employee(empno,ename)  Values (1,'John');
   COLLECT STAT on employee;
   ```

   输出

   ```
   INSERT INTO employee( empno, ename)
   SELECT 1 ,'John';
   ANALYZE employee;
   ```

   输入：ANALYZE，使用 UPDATE

```
UPD employee SET ename = 'Jane'
      WHERE ename = 'John';
COLLECT STAT on employee;
```

输出

```
UPDATE employee SET ename = 'Jane'
 WHERE ename = 'John';
ANALYZE employee;
```

输入：ANALYZE，使用 DELETE

```
DEL FROM employee WHERE ID > 10;
COLLECT STAT on employee;
```

输出

```
DELETE FROM employee WHERE ID > 10;
ANALYZE employee;
```

2. **子句顺序**

从 Teradata 迁移 SELECT 语句时，各子句（FROM、WHERE、HAVING 和 GROUP BY）可按任意顺序排列。如果语句的 FROM 子句之前包含作为 ALIAS 的 QUALIFY 子句，则 DSC 不会迁移该语句。

可以使用 tdMigrateALIAS 参数来配置 ALIAS 的迁移。

输入：子句顺序

```
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 ;
```

输出

```
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 <> '';
```

输入：子句顺序

```
SELECT
        TOP 10 *
    GROUP BY
        DeptNo
    WHERE
        empID < 100
FROM
        tbl_employee;
```

输出

```
SELECT
        *
    FROM
        tbl_employee
    WHERE
        empID < 100
    GROUP BY
        DeptNo LIMIT 10
;
```

说明

如果输入脚本的 FROM 子句之前包含作为 ALIAS 的 QUALIFY 子句，DSC 将不会迁移该语句，也不会逐字复制输入的语句。

输入：子句顺序，在 FROM 子句之前使用 QUALIFY 作为 ALIAS

```
SELECT
        *
    FROM
        table1
    WHERE
        abc = (
            SELECT
                    col1 AS qualify
                FROM
                    TABLE
                    WHERE
                        col1 = 5
        )
;
```

输出

```
SELECT
        *
    FROM
        table1
    WHERE
        abc = (
            SELECT
                    col1 AS qualify
                FROM
                    TABLE
                    WHERE
                        col1 = 5
        )
;
```

3. **扩展 Group By 子句**

   如果用户希望数据库根据 expr（s）的值对选定的行进行分组，则可指定 GROUP BY 子句。 如果此子句包含 CUBE，ROLLUP 或 GROUPING SETS 扩展，则除了常规分组之外，数据库还会生成超级聚合分组。这些特性在 GaussDB(DWS)中不可用，使用 UNION ALL 操作符可以实现类似的功能。

   可以使用 extendedGroupByClause 参数来配置扩展 GROUP BY 子句的迁移。

   输入：扩展 Group By 子句，使用 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 ;
```

   输出

```
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 <> '';
```

   输入：扩展 Group By 子句，使用 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);
```

   输出

```
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
      );
```

输入：扩展 Group By 子句，使用 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);
```

输出

```
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 和 SAMPLE 子句**

   Teradata 的 TOP 和 SAMPLE 子句在 GaussDB(DWS)中迁移为 LIMIT。

   a. TOP

      DSC 还支持迁移使用动态参数的 TOP 语句。

说明

- 对于包含 WITH TIES 的 TOP 语句，需要指定 ORDER BY 子句，否则工具不会迁移该语句，只会原样复制。
- 使用 TOP 和动态参数时：
- 按照以下形式输入动态参数：

  ```
  TOP :<parameter_name>
  ```

  可使用的字符包括：小写英文字母（a-z）、大写英文字母（A-Z）、数字（0-9）、下划线（_）

  输入：SELECT...TOP

  ```
  SELECT TOP 1 c1, COUNT (*) cnt
    FROM tab1
  ```

```
 GROUP BY c1
 ORDER BY cnt;
```

输出

```
SELECT c1, COUNT( * ) cnt
  FROM tab1
 GROUP BY c1
 ORDER BY cnt
 LIMIT 1;
```

输入：SELECT...TOP PERCENT

```
SELECT TOP 10 PERCENT c1, c2
  FROM employee
 WHERE ...
 ORDER BY c2 DESC;
```

输出

```
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);
```

输入：SELECT...TOP，使用动态参数

```
SELECT
         TOP :Limit WITH TIES c1
        ,SUM (c2) sc2
    FROM
        tab1
    WHERE
        c3 > 10
    GROUP BY
        c1
    ORDER BY
        c1
;
```

输出

```
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
;
```

输入：SELECT...TOP，使用动态参数和 TIES

```
 SELECT
         TOP :Limit WITH TIES Customer_ID
   FROM
        Customer_t
   ORDER BY
        Customer_ID
;
```

输出

```
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
;
```

输入：SELECT...TOP PERCENT，使用动态参数

```
SELECT
        TOP :Input_Limit PERCENT WITH TIES c1
        ,SUM (c2) sc2
    FROM
        tab1
    GROUP BY
        c1
    ORDER BY
        c1
;
```

输出

```
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

说明

工具仅支持在 SAMPLE 子句中使用单个正整数。

输入：SELECT...SAMPLE

```
SELECT c1, c2, c3
  FROM tab1
 WHERE c1 > 1000
SAMPLE 1;
```

输出

```
SELECT c1, c2, c3
  FROM tab1
 WHERE c1 > 1000
 LIMIT 1;
```

UPDATE

该工具支持和迁移 UPDATE 语句（缩写关键字为 UPD）。

输入：UPDATE，使用 TABLE ALIAS

```
UPDATE T1
  FROM tab1 T1, tab2 T2
   SET c1 = T2.c1
     , c2 = T2.c2
 WHERE T1.c3 = T2.c3;
```

输出

```
UPDATE tab1 T1
   SET c1 = T2.c1
     , c2 = T2.c2
  FROM tab2 T2
 WHERE T1.c3 = T2.c3;
```

输入：UPDATE，使用 TABLE ALIAS 和子查询

```
UPDATE t1
  FROM tab1 t1, ( SELECT c1, c2 FROM tab2
                  WHERE c2 > 100 ) t2
   SET c1 = t2.c1
 WHERE t1.c2 = t2.c2;
```

输出

```
 UPDATE tab1 t1
   SET c1 = t2.c1
  FROM ( SELECT c1, c2 FROM tab2
         WHERE c2 > 100 ) t2
 WHERE t1.c2 = t2.c2;
```

输入：UPDATE，使用 ANALYZE

```
UPD employee SET ename = 'Jane'
       WHERE ename = 'John';
COLLECT STAT on employee;
```

输出

```
UPDATE employee SET ename = 'Jane'
 WHERE ename = 'John';
ANALYZE employee;
```

DELETE

DELETE（缩写关键字为 DEL）是 ANSI 标准的 SQL 语法操作符，用于从表中删除记录。DSC 支持 Teradata 的 DELETE 语句及其缩写关键字 DEL。不包含 WHERE 子句的 DELETE 语句在 GaussDB(DWS)中被迁移为 TRUNCATE。通过 deleteToTruncate 参数可以配置是否启用/禁用此行为。

输入：DELETE

```
DEL FROM tab1
 WHERE a =10;
```

输出

```
DELETE FROM tab1
 WHERE a =10;
```

输入：DELETE，不使用 WHERE（如果 deletetoTruncate=TRUE，则迁移为 TRUNCATE）

```
DELETE FROM ${schemaname} . "tablename" ALL;
```

输出

```
TRUNCATE
    TABLE
        ${schemaname} . "tablename";
```

以下输入示例中，DELETE 和 FROM 子句引用相同表，按是否使用 WHERE 子句区分：

输入

```
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 )
     );
```

输出

```
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

输入

```
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 ;
```

输出

```
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

说明

6.5.0 及之后版本的 GaussDB 支持 MERGE 功能。

MERGE 是 ANSI 标准的 SQL 语法操作符，用于从一个或多个来源中选择行来更新或插入到表或视图中，可以指定更新或插入到目标表或视图的条件。

输入：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);
```

输出

```
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

Teradata 中的 NAMED 用于为表达式或列分配临时名称。用于表达式的 NAMED 语句在 GaussDB(DWS)中被迁移为 AS。用于列名的 NAMED 语句保留在相同的语法中。

输入：NAMED 表达式，迁移为 AS

```
SELECT Name, ((Salary + (YrsExp * 200))/12) (NAMED Projection)
  FROM Employee
 WHERE DeptNo = 600 AND Projection < 2500;
```

输出

```
SELECT Name, ((Salary + (YrsExp * 200))/12) AS  Projection
  FROM Employee
 WHERE DeptNo = 600 AND ((Salary + (YrsExp * 200))/12)  < 2500;
```

输入：NAMED AS，定义列名

```
SELECT product_id AS id
  FROM emp where pid=2 or id=2;
```

输出

```
SELECT product_id (NAMED "pid") AS id
  FROM emp where product_id=2 or product_id=2;
```

输入：NAMED( )，定义列名

```
INSERT INTO Neg100 (NAMED,ID,Dept) VALUES ('TEST',1,'IT');
```

输出

```
INSERT INTO Neg100 (NAMED,ID,Dept) SELECT 'TEST',1, 'IT';
```

输入：NAMED 别名，使用 TITLE 别名，不使用 AS

```
SELECT dept_name (NAMED alias1) (TITLE alias2 )
  FROM employee
 WHERE dept_name like 'Quality';
```

输出

```
SELECT dept_name
   AS alias1
  FROM employee
 WHERE dept_name like 'Quality';
```

输入：NAMED 别名，使用 TITLE 别名和 AS

DSC 将跳过 NAMED 别名和 TITLE 别名，仅使用 AS 别名。

```
SELECT sale_name (Named alias1 ) (Title alias2)
   AS alias3
  FROM employee
 WHERE sname = 'Stock' OR sname ='Sales';
```

输出

```
SELECT sale_name
   AS alias3
  FROM employee
 WHERE sname = 'Stock' OR sname ='Sales';
```

输入：NAMED，使用 TITLE

NAMED 和 TITLE 一起使用，通过逗号隔开。

```
SELECT customer_id (NAMED cust_id, TITLE 'Customer Id')
FROM Customer_T
WHERE cust_id > 10;
```

输出

```
SELECT cust_id AS "Customer Id"
FROM   (SELECT customer_id AS cust_id
              FROM   customer_t
              WHERE  cust_id > 10);
```

ACTIVITYCOUNT

输入

状态变量，返回嵌入式 SQL 中受 DML 语句影响的行数。

```
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;
```

输出

```
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

输入：TIMESTAMP，使用 FORMAT

FORMAT 短语设置特定 TIME 或 TIMESTAMP 列或值的格式。FORMAT 短语会覆盖系统格式。

```
SELECT 'StartDTTM' as a
          ,CURRENT_TIMESTAMP (FORMAT 'HH:MI:SSBMMMBDD,BYYYY');
```

输出

```
SELECT 'StartDTTM' AS a
          ,TO_CHAR( CURRENT_TIMESTAMP ,'HH:MI:SS MON DD, YYYY' ) ;
```

TIMESTAMP 类型装换：

输入

```
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) )
```

输出

```
COALESCE( a.Snd_Tm , CAST('0001-01-01 00:00:00' AS TIMESTAMP) )
```

6.8.5 查询迁移操作符

本节主要介绍 Teradata 查询迁移操作符的迁移语法。迁移语法决定了关键字/特性的迁移方式。

具体详见以下节点内容：

QUALIFY

ALIAS

FORMAT 和 CAST

缩写关键字迁移

以$开头的对象名称

QUALIFY

通常，QUALIFY 子句和 CSUM()、MDIFF()、ROW_NUMBER()、RANK()等分析函数（窗口函数）一同使用。子查询中会包含 QUALIFY 子句指定的窗口函数。迁移工具支持 QUALIFY 使用 MDIFF()、RANK()和 ROW_NUMBER()函数。QUALIFY 是 Teradata 扩展项，不是标准 ANSI 语法。QUALIFY 在 WHERE 和 GROUP BY 子句后执行，必须单独成行。

说明

只有当 SELECT 语句显式包含列名和/或表达式时，DSC 才允许在 ORDER BY 子句中指定该列名和/或表达式。

输入：QUALIFY

```
SELECT
       CUSTOMER_ID
      ,CUSTOMER_NAME
  FROM
      CUSTOMER_T QUALIFY row_number( ) Over( partition BY CUSTOMER_ID ORDER BY
POSTAL_CODE DESC ) = 1
;
```

输出

```
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
;
```

输入：QUALIFY，使用 MDIFF 和 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
;
```

输出

```
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
;
```

输入：QUALIFY，使用 ORDER BY 且 ORDER BY 中包含不存在于 SELECT 列表的列

```
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;
```

输出

```
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;
```

输入：QUALIFY，使用列别名且不应在 SELECT 列表中再次添加相应的列表达式

```
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;
```

输出

```
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 和 QUALIFY

输入

```
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
;
```

输出

```
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

所有数据库都支持 ALIAS。在 Teradata 中，定义 ALIAS 的语句允许其 SELECT 和 WHERE 子句引用 ALIAS。但是目标数据库的 SELECT 和 WHERE 语句不支持 ALIAS，因此 MT 将其迁移为实际的字段名称替换。

说明

比较操作符 LT、LE、GT、GE、EQ 和 NE 不得用作表别名或列别名。

工具支持列的 ALIAS 名称。如果 ALIAS 名称与列名称相同，则仅为该列而非表中其他列指定 ALIAS。在以下示例中，DATA_DT 列名称与 DATA_DT 别名之间存在冲突，工具不支持。

```
SELECT DATA_DT,DATA_INT AS DATA_DT FROM KK WHERE DATA_DT=DATE;
```

输入：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
;
```

输出：tdMigrateALIAS = FALSE

```
SELECT
        expression1 AS alias1
        ,CASE
            WHEN alias1 + Cx >= z
            THEN 1
            ELSE 0
        END AS alias2
    FROM
        tab1
    WHERE
        alias1 = y
;
```

输出：tdMigrateALIAS = TRUE

```
SELECT
        expression1 AS alias1
        ,CASE
            WHEN expression1 + Cx >= z
            THEN 1
            ELSE 0
        END AS alias2
    FROM
        tab1
    WHERE
        expression1 = y
;
```

FORMAT 和 CAST

Teradata 中，关键词 FORMAT 用于格式化列或表达式。例如，LPAD 中 FORMAT '9(n)' 和'z(n)'分别用'0'和空格(' ')表示。

数据类型转换可使用 CAST 或直接数据类型（[like (expression1)(CHAR(n))]）进行。该功能使用 CAST 实现。详情参见 6.8.11 类型转换和格式化。

输入：FORMAT 和 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
;
```

输出

```
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
;
```

输入：FORMAT 'Z(n)9'

```
SELECT
        standard_price (FORMAT 'Z(5)9') (CHAR( 6 ))
        ,max_price (FORMAT 'ZZZZZ9') (CHAR( 6 ))
    FROM
        product_t
;
```

输出

```
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
;
```

输入：FORMAT 'z(m)9.9(n)'

```
SELECT
        standard_price (FORMAT 'Z(6)9.9(2)') (CHAR( 6 ))
    FROM
        product_t
;
```

输出

```
SELECT
        CAST( TO_CHAR( standard_price ,'9999990.00' ) AS CHAR( 6 ) ) AS
standard_price
    FROM
        product_t
;
```

输入：CAST AS INTEGER

```
SELECT
        CAST( standard_price AS INTEGER )
    FROM
        product_t
;
```

输出

```
SELECT
       (standard_price)
    FROM
        product_t
;
```

输入：CAST AS INTEGER FORMAT

```
SELECT
        CAST( price11 AS INTEGER FORMAT 'Z(4)9' ) (
            CHAR( 10 )
        )
    FROM
        product_t
;
```

输出

```
SELECT
        CAST( TO_CHAR(  ( price11 ) ,'99990' ) AS CHAR( 10 ) ) AS price11
    FROM
        product_t
;
```

说明

新增以下 Gauss 函数来转换为 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;
```

缩写关键字迁移

表 6-18 列出了 Teradata 支持的缩写关键字及其语法在 GaussDB(DWS)中对应的语法。

表6-18 缩写关键字列表

Teradata 语法	对应的 GaussDB(DWS)语法
SEL	SELECT
INS	INSERT
UPD	UPDATE

Teradata 语法	对应的 GaussDB(DWS)语法
DEL	DELETE
CT	CREATE TABLE
CV	CREATE VIEW
BT	START TRANSACTION
ET	COMMIT

输入：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 ..
```

输出

```
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;
```

以$开头的对象名称

本节介绍如何迁移以**$**（美元符号）开头的对象名称。

下表具体描述了这些对象名称的迁移行为。这些行为可以通过 tdMigrateDollar 参数来设置。

详情请参见 IN/NOT IN 转换。

表6-19 以$开头的对象名称的迁移行为

tdMigrateDollar 设置	对象名称	迁移为
true	$V_SQL 静态对象名称	"$V_SQL"
true	${V_SQL}	${V_SQL}

tdMigrateDollar 设置	对象名称	迁移为
	动态对象名称	无变化：不支持动态对象名称
false	$V_SQL 静态对象名称	$V_SQL 无变化：参数设为 false
false	${V_SQL} 动态对象名称	${V_SQL} 无变化：参数设为 false

说明

任何以$开头的变量都被视为值。工具将通过添加 ARRAY 来进行迁移。

输入：以$开头的对象

```
SELECT $C1 from p11 where $C1 NOT LIKE ANY ($sql1);
```

输出（tdMigrateDollar = TRUE）

```
SELECT
       "$C1"
FROM
       p11
WHERE
       "$C1" NOT LIKE ANY (
       ARRAY[ "$sql1" ]
       )
;
```

输出（tdMigrateDollar = FALSE）

```
SELECT
       $C1
FROM
       p11
WHERE
       $C1 NOT LIKE ANY (
       ARRAY[ $sql1 ]
       )
;
```

输入：LIKE ALL/LIKE ANY 中以$开头的值

```
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);
```

输出

```
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 和 ORDER BY

输入

```
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
;
```

输出

```
 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 查询优化操作符

本节主要介绍 Teradata 查询优化操作符的迁移语法。迁移语法决定了关键字/特性的迁移方式。

可以使用 inToExists 参数来配置从 IN/NOT IN 的到 EXISTS/NOT EXISTS 的迁移行为。

该参数的值默认为 FALSE。要使用查询优化功能，需将该参数值设为 TRUE。

IN 和 NOT IN 转换

迁移到 GaussDB(DWS)的 SQL 查询时，包含 IN/NOT IN 参数的 Teradata 查询已经优化并转换为 EXISTS/NOT EXISTS 运算符。IN/NOT IN 操作符可对单列或多列使用。只有当 IN/NOT IN 语句存在于 WHERE 或 ON 子句中时，DSC 才会迁移该语句。接下来以 IN 到 EXISTS 的转换为例（同样适用于 NOT IN 到 NOT EXISTS 的转换）。

IN 到 EXISTS 的简单转换

在如下示例中，输入文件中提供了关键词 IN。为进行优化，该工具在迁移过程中将其替换为 EXISTS 关键词。

说明

- 嵌套 IN/NOT IN 的 IN/NOT IN 语句不支持迁移，迁移的脚本将失效。

```
UPDATE tab1
   SET b = 123
 WHERE b IN ('abc')
   AND b IN ( SELECT  i
              FROM tab2
              WHERE j NOT IN (SELECT m
                              FROM tab3
                             )
            )
;
```

迁移包含子查询的 IN/NOT IN 语句时，不支持迁移 IN/NOT IN 操作符和子查询（见示例）之间的注释。

示例：

```
SELECT *
  FROM categories
 WHERE category_name
    IN --comment
       ( SELECT category_name
          FROM categories1 )
 ORDER BY category_name;
```

- **IN/NOT IN 语句迁移，其中对象名称包含$和#**
- 如果 TABLE 名称或 TABLE ALIAS 以$（美元符号）开头，则工具不会对查询进行迁移。

```
SELECT Customer_Name
  FROM Customer_t $A
 WHERE Customer_ID IN( SELECT Customer_ID FROM Customer_t );
```

- 如果 COLUMN 名称以＃（hash）开头，则工具进行的查询迁移可能存在问题。

```
SELECT Customer_Name
  FROM Customer_t
 WHERE #Customer_ID IN( SELECT #Customer_ID FROM Customer_t );
```

输入：IN

```
SELECT ...
  FROM tab1 t
  WHERE t.col1 IN (SELECT icol1 FROM tab2 e)
 ORDER BY col1
```

输出

```
SELECT ...
  FROM tab1 t
  WHERE EXISTS (SELECT icol1
                FROM tab2 e
               WHERE icol1 = t.col1
              )
 ORDER BY col1;
```

输入：IN，使用多列以及 Aggregate 函数

```
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;
```

输出

```
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 系统函数和操作符

本节主要介绍 Teradata 系统函数和运算符的迁移语法。迁移语法决定了关键字/特性的迁移方式。

模式

带有模式名的数据库更改为“SET SESSION CURRENT_SCHEMA”。

Oracle 语法	迁移后语法
`DATABASE SCHTERA`	`SET SESSION CURRENT_SCHEMA TO SCHTERA`

分析函数

在 Teradata 中，分析函数统称为有序分析函数，它们为数据挖掘、分析和商业智能提供了强大的分析能力。

1. **ORDER BY 中的分析函数**

   输入：ORDER BY 子句中的分析函数

   ```
   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);
```

输出

```
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) ;
```

输入：GROUP BY 子句中的分析函数

```
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;
```

输出

```
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. **PARTITION BY 中的分析函数**

   当输入脚本的 PARTITION BY 子句中包含数值时，迁移脚本将原样保留该数值。

   输入：PARTITION BY 子句中的分析函数（包含数值）

```
SELECT
        Customer_id
        ,customer_name
        ,rank (
        ) over( partition BY 1 ORDER BY Customer_id )
        ,rank (customer_name)
    FROM
        Customer t
    GROUP BY
        1
;
```

   输出

```
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. **窗口函数**

   窗口函数在查询结果中执行跨行计算。DSC 支持以下 Teradata 窗口函数：

说明

DSC 仅支持 QUALIFY 子句使用一个窗口函数。如果 QUALIFY 使用多个窗口函数，可能会导致迁移失败。

4. **CSUM**

累计函数(CSUM)为一列数值计算运行或累计总数，建议在 QUALIFY 语句中使用 ALIAS。

输入：CSUM，使用 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'
;
```

输出

```
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'
;
```

输入：CSUM，使用 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;
```

输出

```
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
;
```

输入：CSUM，使用 GROUP BY 和 QUALIFY

```
SELECT c1, c2, c3, CSUM(c4, c3)
  FROM tab1
QUALIFY ROW_NUMBER(c4) = 1
GROUP BY 1, 2;
```

输出

```
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**

   MDIFF 函数基于预定的查询宽度计算一列的移动差分值。查询宽度即所指定的行数。建议在 QUALIFY 语句中使用 ALIAS。

   输入：MDIFF，使用 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;
```

   输出

```
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...)

   输入：RANK，使用 GROUP BY

```
SELECT  c1, c2, c3, RANK(c4, c1 DESC, c3) AS Rank1
  FROM  tab1
 WHERE  ...
 GROUP BY c1;
```

   输出

```
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...)

   输入：ROW NUMBER，使用 GROUP BY 和 QUALIFY

```
SELECT c1, c2, c3, ROW_NUMBER(c4, c3)
   FROM tab1
QUALIFY RANK(c4) = 1
  GROUP BY 1, 2;
```

输出

```
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（使用*****）**

   输入

```
ORDCADBRN VARCHAR(6) CHARACTER SET LATIN CASESPECIFIC TITLE '    ' COMPRESS
'******'
```

   输出

```
ORDCADBRN VARCHAR( 6 ) /* CHARACTER SET LATIN*/ /* CASESPECIFIC*/ /*TITLE '
'*/ /* COMPRESS  '******' */
```

比较和列表操作符

说明

比较操作符 LT、LE、GT、GE、EQ 和 NE 不得用作表别名或列别名。

以下内容介绍了支持的比较和列表操作符。

1. **^=和 GT**

   输入：比较操作（^=和 GT）

```
SELECT t1.c1, t2.c2
  FROM tab1 t1, tab2 t2
 WHERE t1.c3 ^= t1.c3
   AND t2.c4 GT 100;
```

   输出

```
SELECT t1.c1, t2.c2
  FROM tab1 t1, tab2 t2
```

```
WHERE t1.c3 <> t1.c3
  AND t2.c4 > 100;
```

2. **EQ 和 NE**

   输入：比较操作（EQ 和 NE）

```
SELECT t1.c1, t2.c2
  FROM tab1 t1 INNER JOIN tab2 t2
    ON t1.c2 EQ t2.c2
 WHERE t1.c6 NE 1000;
```

   输出

```
 SELECT t1.c1, t2.c2
  FROM tab1 t1 INNER JOIN tab2 t2
    ON t1.c2 = t2.c2
 WHERE
       t1.c6 <> 1000;
```

3. **LE 和 GE**

   输入：比较操作（LE 和 GE）

```
SELECT t1.c1, t2.c2
  FROM tab1 t1, tab2 t2
 WHERE t1.c3 LE 200
   AND t2.c4 GE 100;
```

   输出

```
 SELECT t1.c1, t2.c2
   FROM tab1 t1, tab2 t2
  WHERE t1.c3 <= 200
    AND t2.c4 >= 100;
```

4. **NOT=和 LT**

   输入：比较操作（NOT=和 LT）

```
SELECT t1.c1, t2.c2
  FROM tab1 t1, tab2 t2
 WHERE t1.c3 NOT= t1.c3
   AND t2.c4 LT 100;
```

   输出

```
SELECT t1.c1, t2.c2
  FROM tab1 t1, tab2 t2
 WHERE t1.c3 <> t1.c3
   AND t2.c4 < 100;
```

5. **IN 和 NOT IN**

   详情请参见 IN/NOT IN 转换。

   输入：IN 和 NOT IN

```
 SELECT c1, c2
   FROM tab1
  WHERE c1 IN 'XY';
```

   输出

```
SELECT c1, c2
  FROM tab1
 WHERE c1 = 'XY';
```

说明

GaussDB(DWS)支持 IN 和 NOT IN 操作符，特殊场景除外。

6. **IS NOT IN**

   输入：IS NOT IN

   ```
   SELECT c1, c2
     FROM tab1
    WHERE c1 IS NOT IN (subquery);
   ```

   输出

   ```
   SELECT c1, c2
     FROM tab1
    WHERE c1 NOT IN (subquery);
   ```

7. **LIKE ALL 和 NOT LIKE ALL**

   输入：LIKE ALL/NOT LIKE ALL

   ```
   SELECT c1, c2
     FROM tab1
    WHERE c3 NOT LIKE ALL ('%STR1%', '%STR2%', '%STR3%');
   ```

   输出

   ```
   SELECT c1, c2
     FROM tab1
    WHERE c3 NOT LIKE ALL (ARRAY[ '%STR1%', '%STR2%', '%STR3%' ]);
   ```

8. **LIKE ANY 和 NOT LIKE ANY**

   输入：LIKE ANY/NOT LIKE ANY

   ```
   SELECT c1, c2
     FROM tab1
    WHERE c3 LIKE ANY ('STR1%', 'STR2%', 'STR3%');
   ```

   输出

   ```
   SELECT c1, c2
     FROM tab1
    WHERE c3 LIKE ANY (ARRAY[ 'STR1%', 'STR2%', 'STR3%' ]);
   ```

表操作符

可以在查询的 FROM 子句中调用函数，该函数包含在表操作符内部。

输入：表操作符，使用 RETURNS

```
SELECT *
  FROM TABLE( sales_retrieve (9005) RETURNS ( store INTEGER, item CLOB, quantity
BYTEINT) ) AS ret;
```

输出

```
SELECT *
  FROM sales_retrieve(9005) AS ret (store, item, quantity);
```

6.8.8 数学函数

**

输入：**

```
expr1 ** expr2
```

输出

```
expr1 ^ expr2
```

MOD

输入：MOD

```
expr1 MOD expr2
```

输出

```
 expr1 % expr2
```

NULLIFZERO

可以使用 tdMigrateNULLIFZERO 参数来配置 NULLIFZERO 迁移。

输入：NULLIFZERO

```
SELECT NULLIFZERO(expr1) FROM tab1
WHERE … ;
```

输出

```
SELECT NULLIF(expr1, 0) FROM tab1
 WHERE … ;
```

ZEROIFNULL

可以使用 tdMigrateZEROIFNULL 参数来配置 ZEROIFNULL 迁移。

输入：ZEROIFNULL

```
SELECT ZEROIFNULL(expr1) FROM tab1
WHERE … ;
```

输出

```
SELECT COALESCE(expr1, 0) FROM tab1
 WHERE … ;
```

声明 Hexadecimal Character Literal 值

输入

```
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
;
```

输出

```
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
;
```

声明 Hexadecimal Binary Literal 值

输入

```
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 ;
```

输出

```
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 字符串函数（Teradata）

CHAR 函数

输入：CHAR

```
CHAR( expression1 )
```

输出

```
LENGTH( expression1 )
```

CHARACTERS 函数

输入：CHARACTERS

```
CHARACTERS( expression1 )
```

输出

```
LENGTH( expression1 )
```

INDEX

输入：INDEX

```
SELECT INDEX(expr1/string, substring)
 FROM tab1
WHERE … ;
```

输出

```
SELECT INSTR(expr1/string, substring)
 FROM tab1
WHERE … ;
```

STRREPLACE

输入：STRREPLACE

```
SELECT STRREPLACE(c2, '.', '')
 FROM tab1
WHERE ...;
```

输出

```
SELECT REPLACE(c2, '.', '')
 FROM tab1
WHERE ...;
```

OREPLACE

输入：OREPLACE

```
SELECT OREPLACE (c2, '.', '')
 FROM tab1
WHERE … ;
```

输出

```
SELECT REPLACE(c2, '.', '')
 FROM tab1
WHERE … ;
```

6.8.10 日期和时间函数

DATE

DSC 支持迁移 Teradata 的 SELECT 语句中包含 DATE FORMAT，使用 TO_CHAR 并以源格式显示日期。如果日期格式是一个表达式（例如：Start_Dt + 30）或者 WHERE 子句包含表达式（例如：WHERE Start_Dt > End_Dt），则不会执行此转换。

详情请参见转换类型为 DATE（无 DATE 关键字）。

说明

- 无论 SELECT 语句是否有列别名，都可以进行迁移。

- 子查询和内部查询不支持日期格式化，仅外部查询支持。
- 关于日期格式化，如果建表时使用了模式名称，则后续 SELECT 查询仍需包含模式名称。在以下示例中，SELECT 语句中的表 TEMP_TBL 不会迁移，原样保留。

```
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;
```

输入：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
;
```

输出

```
SELECT
        CASE
            WHEN SUBSTR( TO_CHAR( CAST( SUBSTR( '20180631' ,1 ,6 ) || '01' AS DATE )
+ abc_day - 1 ,'YYYYMMDD' ) ,1 ,6 ) = SUBSTR( '20180631' ,1 ,6 )
            THEN 1
            ELSE 0
        END
    FROM
        tab1
;
```

DSC 支持迁移日期值。如果输入 DATE 后又输入“YYYY-MM-DD”，则输出中的日期不会改变。 以下示例显示 DATE 到 CURRENT DATE 的转换。

输入：DATE

```
SELECT
      t1.c1
      ,t2.c2
  FROM
      $schema.tab1 t1
      ,$schema.tab2 t2
  WHERE
      t1.c3 ^ = t1.c3
      AND t2.c4 GT DATE
;
```

输出

```
SELECT
       t1.c1
       ,t2.c2
  FROM
       "$schema".tab1 t1
       ,"$schema".tab2 t2
 WHERE
       t1.c3 <> t1.c3
       AND t2.c4 > CURRENT_DATE
;
```

输入：DATE 和"YYYY-MM-DD"

```
ALTER TABLE
    $abc . tab1 ADD (
        col_date DATE DEFAULT DATE '2000-01-01'
    )
;
```

输出

```
ALTER TABLE
    "$abc" . tab1 ADD (
        col_date DATE DEFAULT DATE '2000-01-01'
    )
;
```

输入：DATE 减法

```
SELECT
        CAST( T1.Buyback_Mature_Dt - CAST( '${gsTXDate}' AS DATE FORMAT 'YYYYMMDD' )
AS CHAR( 5 ) )
    FROM
        tab1 T1
    WHERE
        T1.col1 > 10
;
```

输出

```
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

输入

```
ADD_MONTHS(CAST(substr(T1.GRANT_DATE,1,8)||'01'AS DATE FORMAT 'YYYY-MM-DD'),1)-1
```

输出

```
ADD_MONTHS(CAST(SUBSTR( T1.GRANT_DATE ,1 ,8 ) || '01' AS DATE ),1) - 1
```

TIMESTAMP

输入：TIMESTAMP

```
select CAST('20190811'||' '||'01:00:00'
AS TIMESTAMP(0)
FORMAT 'YYYYMMDDBHH:MI:SS'
) ;
```

输出

```
SELECT TO_TIMESTAMP( '20190811' || ' ' || '01:00:00' ,'YYYYMMDD HH24:MI:SS' ) ;
```

TIME FORMAT

输入

```
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)))
```

输出

```
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

输入

```
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')  ;
```

输出

```
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

输入

```
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' )
```

输出

```
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')

输入

```
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
         )
          )
/
```

输出

```
date_trunc('month', cast('${v_Trx_Dt}' as date))
date_trunc('year', cast('${v_Trx_Dt}' as date))
```

NEXT

输入：NEXT

```
SELECT c1, c2
 FROM tab1
 WHERE NEXT(c3) = CAST('2004-01-04' AS DATE FORMAT 'YYYY-MM-DD');
```

输出

```
 SELECT c1, c2
  FROM tab1
 WHERE c3 + 1 = CAST('2004-01-04' AS DATE);
```

6.8.11 类型转换和格式化

本节主要介绍 Teradata 类型转换和格式化的迁移语法。迁移语法决定了关键字/特性的迁移方式。

在 Teradata 中，FORMAT 关键词用于格式化字段/表达式。FORMAT '9(n)' 和'z(n)'分别使用 0 和空格（''）填充，即使用 LPAD 函数。数据类型转换可通过 CAST 或直接指定数据类型实现：[like (expression1)(CHAR(n))]。DSC 使用 CAST 完成。

DSC 支持如下类型转换和格式化：

- CHAR
- COLUMNS 和 COLUMN ALIAS
- 表达式
- INT
- DATE
- 转换数据类型 DAY 为 SECOND
- DECIMAL
- 时间间隔
- NULL
- 隐式类型转换

CHAR

输入：CHAR 转换

```
(expression1)(CHAR(n))
```

输出

```
CAST( (expression1) AS CHAR(n) )
```

COLUMNS 和 COLUMN ALIAS

输入：对某列进行类型转换和格式化时，应确保列名和别名相同

```
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;
```

输出

```
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;
```

表达式

输入：对表达式进行类型转换和格式化

```
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 ;
```

输出

```
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

输入：INT 转换

```
SELECT
        CAST( col1 AS INT ) (
            FORMAT '9(5)'
        )
    FROM
        table1
;
```

输出

```
SELECT
        LPAD( CAST( col1 AS INT ) ,5 ,'0' )
    FROM
        table1
;
```

输入：INT 转换

```
SELECT
        CAST( col1 AS INT ) (
            FORMAT '999999'
        )
    FROM
        table1
;
```

输出

```
SELECT
        LPAD( CAST( col1 AS INT ) ,6 ,'0' )
    FROM
        table1
;
```

输入：INT 转换

```
SELECT
        CAST( expression1 AS INT FORMAT '9(10)' )
```

```
    FROM
        table1
;
```

输出

```
SELECT
        LPAD( CAST( expression1 AS INT ) ,10 ,'0' )
    FROM
        table1
;
```

输入：INT 转换

```
SELECT
        CAST( expression1 AS INT FORMAT '9999' )
    FROM
        table1
;
```

输出

```
SELECT
        LPAD( CAST( expression1 AS INT ) ,4 ,'0' )
    FROM
        table1
;
```

DATE

在 Teradata 中对 DATE 进行格式转换时，使用 AS FORMAT。DSC 将添加 TO_CHAR 函数来保留指定的输入格式。

详情请参见 6.8.10 日期和时间函数。

输入：数据类型转换，不使用 DATE 关键字

```
SELECT
     CAST( CAST( '2013-02-12' AS DATE FORMAT 'YYYY/MM/DD' ) AS FORMAT 'DD/MM/YY' )
;
```

输出

```
SELECT
     TO CHAR( CAST( '2013-02-12' AS DATE ) ,'DD/MM/YY' )
;
```

转换数据类型 DAY 为 SECOND

输入：DAY 转换为 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;
```

输出

```
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

输入：DECIMAL 转换

```
SELECT
        standard_price (
            DECIMAL( 17 )
            ,FORMAT '9(17)'
        ) (
            CHAR( 17 )
        )
    FROM
        db_pvfc9_std.Product_t
;
```

输出

```
SELECT
        CAST( LPAD( CAST( standard_price AS DECIMAL( 17 ,0 ) ) ,17 ,'0' ) AS
CHAR( 17 ) )
    FROM
        db_pvfc9_std.Product_t
;
```

输入：DECIMAL 转换

```
SELECT
        standard_price (
            DECIMAL( 17 ,0 )
            ,FORMAT '9(17)'
        ) (
            VARCHAR( 17 )
        )
    FROM
        db_pvfc9_std.Product_t
;
```

输出

```
SELECT
        CAST( LPAD( CAST( standard_price AS DECIMAL( 17 ,0 ) ) ,17 ,'0' ) AS
VARCHAR( 17 ) )
    FROM
        db_pvfc9_std.Product_t
;
```

输入：DECIMAL 转换

```
SELECT
        customer_id (
            DECIMAL( 17 )
```

```
        ) (
            FORMAT '9(17)'
        ) (
            VARCHAR( 17 )
        )
    FROM
        db_pvfc9_std.Customer_t
;
```

输出

```
SELECT
        CAST( LPAD( CAST( customer_id AS DECIMAL( 17 ,0 ) ) ,17 ,'0' ) AS
VARCHAR( 17 ) )
    FROM
        db_pvfc9_std.Customer_t
;
```

时间间隔

DDL 和 DML 支持将数据类型转换为时间间隔。用户可在 VIEW、MERGE 和 INSERT 子查询和 SELECT 内进行转换。

输入：转换为时间间隔

```
SELECT TIME '06:00:00.00' HOUR TO SECOND;
```

输出

```
SELECT TIME '06:00:00.00';
```

输入：转换为时间间隔，使用 TOP

```
SELECT TOP 3 * FROM dwQErrDtl_mc.C03_CORP_AGENT_INSURE
WHERE Data_Dt > (SELECT TIME '06:00:00.00' HOUR TO SECOND);
```

输出

```
SELECT  * FROM dwQErrDtl_mc.C03_CORP_AGENT_INSURE WHERE  Data_Dt > (SELECT TIME
'06:00:00.00')  limit 3;
```

NULL

DSC 将 NULL(data_type)形式的表达式迁移为 CAST(NULL AS replacement_data_type)。

输入：NULL 转换

```
NULL(VARCHAR(n))
```

输出

```
CAST(NULL AS VARCHAR(n))
```

隐式类型转换

输入：隐式类型转换

```
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
 ;
```

输出

```
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 工具命令

BTEQ 工具提供以下命令：

LOGOFF 和 QUIT

LOGOFF 命令结束当前 RDBMS 会话，但不会退出 BTEQ 工具。

QUIT 命令结束当前 Teradata 数据库会话，并退出 BTEQ 工具。

输入

```
SELECT 'StartDTTM' as a
         ,CURRENT_TIMESTAMP (FORMAT 'HH:MI:SSBMMMBDD,BYYYY') ;
.LOGOFF;
.QUIT;
```

输出

```
SELECT 'StartDTTM' as a
         ,CURRENT_TIMESTAMP (FORMAT 'HH:MI:SSBMMMBDD,BYYYY') ;
.LOGOFF;
.QUIT;
```

说明

Gauss 不支持上述命令，需要加注释。

.IF ERRORCODE 和.QUIT

BTEQ 命令，指定.IF 和.QUIT。

输入

```
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 ;
```

输出

```
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

输入

```
.IF END_MONTH_FLAG <> 'Y' THEN .GOTO LABEL_1;
```

输出

```
IF END_MONTH_FLAG <> 'Y' THEN
            GOTO LABEL_1 ;
END IF ;
```

.QUIT

输入

```
.IF ERRORCODE <> 0 THEN .QUIT 12;
```

输出

```
IF ERRORCODE <> 0 THEN
        RAISE EXCEPTION '12' ;
END IF;
```

.IF 与.QUIT

将.IF 与.Quit 移入语句块。

Oracle 语法	迁移后语法
INSERT INTO EMP	DECLARE

Oracle 语法	迁移后语法
SELECT * FROM EMP; .IF ERRORCODE <> 0 THEN .QUIT 12;	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

输入

```
.if ERRORCODE = 0 then .RETURN;
```

输出

```
IF ERRORCODE = 0 THEN
        RETURN;
END IF;
```

.GOTO

输入

```
.IF END_MONTH_FLAG <> 'Y' THEN .GOTO LABEL_1;
```

输出

```
IF END_MONTH_FLAG <> 'Y' THEN
     GOTO LABEL_1;
END IF ;
```

Label

输入

```
.LABEL LABEL_1
```

输出

```
<<LABEL_1>>
```

ERRORCODE 3807

输入

```
SELECT end_mon AS END_MONTH_FLAG
  FROM tab2 ;

.IF END_MONTH_FLAG <> 'Y' THEN
       .GOTO LABEL_1;

.IF ERRORCODE = 3807 THEN
      .QUIT 8888;
```

输出

```
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（BTEQ 事务命令）

输入

```
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;
```

输出

```
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（BTEQ 事务命令）

输入

```
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;
```

输出

```
--
  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

将.EXPORT FILE 修改为 COPY，并将 teradata 实用程序设置为 false。

Oracle 语法	迁移后语法
.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';

当 teradata 实用程序设置为 true 时，应将.EXPORT FILE 修改为qCOPY，并移入语句块。

Oracle 语法	迁移后语法
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

使用 SQL_Lang 需处理多个标签名。

Oracle 语法	迁移后语法
#!/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";	#!/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

Oracle 语法	迁移后语法
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; 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 ($UNIT_FLAG eq "Z") {	="${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 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 ;

Oracle 语法	迁移后语法
$BTEQCMD = BTEQ_Z1($dbh); } else { print "\n"; return 1; } ETL::disconnectETL($dbh); return ETL::ExecuteBTEQ($BTEQCMD, $TX_DATE); } my $ret= main(); exit($ret);	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); 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);

Oracle 语法	迁移后语法
	return ETL::ExecuteBTEQ($BTEQCMD, $TX_DATE); } my $ret= main(); exit($ret);

6.8.13 DSQL

DSQL 变量-使用 AS

输入

```
/* 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 ;
```

输出

```
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 变量-不使用 AS

输入

```
/* 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 ;
```

输出

```
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 ;
/
```

通过 BTEQ 语句指定变量

输入

```
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
```

输出

```
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;
```

通过 SELECT EXTRACT FROM 指定变量

输入

```
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;
```

输出

```
BEGIN
   select EXTRACT (MONTH FROM cast('${TX_DATE}' as  date format 'YYYYMMDD')) INTO
CURR_MON;
…
END;
…
```

日期类型转换-指定 DSQL 变量

输入

```
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 ,'' )
  ;
```

输出

```
 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 语法迁移

6.9.1 Oracle 迁移概述

本节列出了语法迁移工具支持的 Oracle 特性，并且针对每个特性提供了 Oracle 语法及其相应的 GaussDB(DWS)语法。本节中列出的语法说明了 Oracle 脚本使用的内部迁移逻辑。

本节还可以作为数据库迁移团队的参考，作为客户现场验证 Oracle 脚本迁移的参考。

6.9.2 模式对象

本节主要介绍 Oracle 模式对象的迁移语法。迁移语法决定了关键字/功能的迁移方式。

本节包括以下内容：

表、临时表、全局临时表、索引、视图、序列、PURGE、数据库关键字，具体内容详见 6.9.2.1 表（Oracle）~6.9.2.8 数据库关键字章节。

6.9.2.1 表（Oracle）

CREATE TABLE

Oracle 的 CREATE TABLE 语句用于创建表。GaussDB 直接支持该语句，无需迁移

ALTER TABLE

Oracle 的 ALTER TABLE 语句用于新增、重命名、修改或删除表列。GaussDB 直接支持该语句，无需迁移。

PRIMARY KEY

Oracle 中如果存在两张表具有相同的主键字段，则在执行 ALTER TABLE 时需加上表名进行区分。

输入：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;
```

输出

```
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 约束

以下 ALTER TABLE 语句包含约束，如果在 GaussDB 直接调用会报错：Cannot create index whose evaluation cannot be enforced to remote nodes.

该约束迁移和 PRIMARY KEY 类似。如果已有 PRIMARY KEY/UNIQUE 约束，无需迁移，保持原样。

输入

```
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) ;
```

输出

```
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 约束

在以下包中声明局部变量时不支持 NULL 约束：

L_CONTRACT_DISTRIBUTE_STATUS
SAD_DISTRIBUTION_HEADERS_T.STATUS%TYPE NULL ;

输入

```
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;
/
```

输出

```
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 ;
/
```

未创建索引

如果 ALTER TABLE 中使用了 INDEX 或 STORAGE 参数，需要删掉。需要在 CREATE TABLE 中添加约束。

输入：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;
```

输出

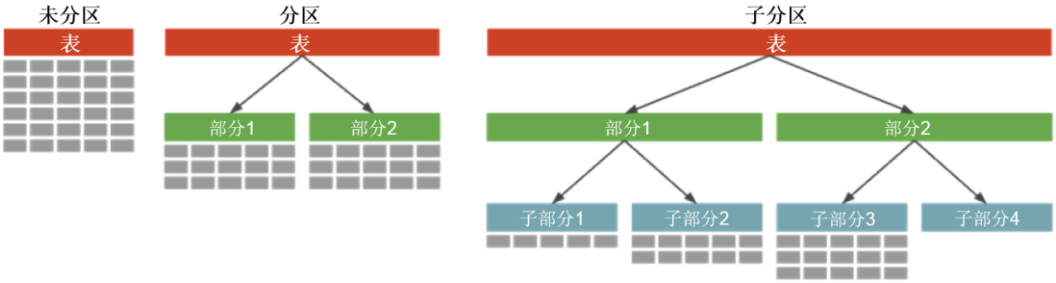
```
CREATE TABLE
CTP_ARM_CONFIG (
HOSTNAME VARCHAR2 (50)
,OPNAME VARCHAR2 (50)
,PARAMTYPE VARCHAR2 (2)
,PARAMVALUE NUMBER (
38
,0
)
```

```
,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 */
;
```

分区

大表和索引的维护越来越耗费时间和资源。同时，这些对象会导致数据访问性能明显降低。表和索引的分区可从各方面提升性能、便于维护。

图6-6 表的分区和子分区



DSC 支持范围分区。

该工具不支持以下分区/子分区（在迁移脚本中会被注释掉）：

- 列表分区
- Hash 分区
- 范围子分区
- 列表子分区
- Hash 子分区

未来可能会支持当前不支持的分区/子分区。该工具中，用户可设置配置参数，启用/禁用对不支持语句的注释功能。详情请参见表 6-8。

- **PARTITION BY HASH**

  Hash 分区是一种分区技术，其中 Hash 算法用于在不同分区（子表）之间均匀分配行。通常在无法进行范围分区时使用该技术，例如通过员工 ID、产品 ID 等进

行分区。DSC 不支持 PARTITION BY HASH 和 SUBPARTITION BY HASH，且会注释掉这些语句。

输入：HASH PARTITION

```
CREATE TABLE dept (deptno NUMBER, deptname VARCHAR(32)) PARTITION BY
HASH(deptno) PARTITIONS 16;
```

输出

```
CREATE TABLE dept ( deptno NUMBER ,deptname VARCHAR( 32 ) ) /* PARTITION BY
HASH(deptno) PARTITIONS 16 */ ;
```

输入：HASH PARTITION，不使用分区名

```
CREATE TABLE dept (deptno NUMBER, deptname VARCHAR(32))
     PARTITION BY HASH(deptno) PARTITIONS 16;
```

输出

```
CREATE TABLE dept (deptno NUMBER, deptname VARCHAR(32))
  /*    PARTITION BY HASH(deptno) PARTITIONS 16 */;
```

输入：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'))
  );
```

输出

```
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**

  列表分区是一种分区技术，在每个分区的说明中指定分区键的离散值列表。DSC 不支持 PARTITION BY LIST 和 SUBPARTITION BY LIST，且会注释掉这些语句。

  输入：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) );
```

  输出

```
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) ) */ ;
```

  输入：LIST PARTITION（使用 STORAGE 参数）

```
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 )
      );
```

输出

```
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 );
```

输入：LIST PARTITION TABLE，基于其他 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;
```

输出

```
CREATE TABLE tab1_list
 AS
 ( SELECT *
   FROM  tab1 );
```

输入：LIST PARTITION，使用 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;
```

输出

```
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 ) ;
```

输入：LIST PARTITION，使用 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' )
   );
```

输出

```
CREATE TABLE q1_sales_by_region
        ( deptno NUMBER
        , deptname varchar2 (20)
        , quarterly sales NUMBER (10,2)
        , state varchar2 (2)
        );
```

- **PARTITION BY RANGE**

  范围分区是一种分区技术，将不同范围数据分别存储在不同的子表中。当用户需要将不同范围的数据（例如日期字段）存储在一起时，范围分区很有用。DSC 支持 PARTITION BY RANGE，不支持 SUBPARTITION BY RANGE，且会注释掉该语句。

  输入：RANGE PARTITION（使用 STORAGE 参数）

```
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 )*/
```

输出

```
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*/
          );
```

输入：RANGE PARTITION，使用 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));
```

输出

```
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));
```

输入：RANGE PARTITION，使用 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));
```

输出

```
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));
```

分区表的 PRIMARY KEY/UNIQUE 约束

如果 CREATE TABLE 语句包含范围/Hash/列表分区，则添加约束会产生如下错误：

Invalid PRIMARY KEY/UNIQUE constraint for partitioned table

注意：PRIMARY KEY/UNIQUE 约束所在的列必须包含 PARTITION KEY。

脚本： wo_integrate_log_t.SQL, wo_change_log_t.SQL

输入：

```
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);
```

输出：

```
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 ;
```

输入：

```
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);
```

输出：

```
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);
```

数据类型

删除数据类型中的 BYTE 关键字。

Oracle 语法	迁移后语法
CREATE TABLE BL_ORACLE (ID Number, Name VARCHAR2(100 BYTE),	CREATE TABLE TBL_ORACLE (ID NUMBER ,Name VARCHAR2 (100)

Oracle 语法	迁移后语法
ADDRESS VARCHAR2(200 BYTE));	,ADDRESS VARCHAR2 (200)) ;

分区（注释分区）

oracle 配置参数中“#分区表唯一或主键约束”为“comment_partition”。

Oracle 语法	迁移后语法
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 */ ;

分区（注释约束）

oracle 配置参数中“#分区表唯一或主键约束”为“comment_unique”。

Oracle 语法	迁移后语法
CREATE TABLE TBL_ORACLE (ID Number, Name VARCHAR2(100 BYTE),	CREATE UNLOGGED TABLE TBL_ORACLE (ID NUMBER ,Name VARCHAR2 (100)

Oracle 语法	迁移后语法
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);	,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 ;

分区（一）

在非分区表中，为表“ALTER TABLE TRUNCATE PARTITION”添加注释。

Oracle 语法	迁移后语法
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),	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),

Oracle 语法	迁移后语法
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; /	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; /

分区（二）

在非分区表中，删除表“ALTER TABLE TRUNCATE PARTITION”中的数据。

Oracle 语法	迁移后语法
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 ,	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 ,

Oracle 语法	迁移后语法
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; /	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

Oracle 语法	迁移后语法
	Year_Manufacture NOT IN (2011,2012,2013,2014,2015,2016,2017,2018,2019,2020); END IF; NULL; END; /

SEGMENT CREATION

GaussDB 不支持 SEGMENT CREATION { IMMEDIATE | DEFERRED }，因此该语句在迁移后被注释掉，需要设置 commentStorageParameter=true。

输入：TABLE，使用 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 ;
```

输出：

```
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 */;
```

GaussDB 不支持 BUFFER_POOL、MAXEXTENTS 等存储参数。如果 commment_storage_parameter 设置为 true，出现在表或索引中的这些参数在迁移时会被注释掉。

输入：TABLE，使用 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)) ;
```

输出

```
 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 */
 ;
```

说明

如果 commment_storage_parameter 设为 true，存储参数会被注释掉。

STORE

Gauss 不支持 LOB 列的 STORE 关键字，因此该关键字在迁移后会被注释掉。

输入：TABLE，使用 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)) ;
```

输出：

```
 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

所有表均不支持存储参数 PCTINCREASE。此外，分区表不支持所有存储参数（包括 pctfree、minextents 和 maxextents）。

输入：TABLE，使用 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 );*/
```

输出：

```
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 );*/
```

外键

外键在 Oracle 数据库中用于强制保证引用的完整性。外键意味着一个表中的值必须同时存在另一个表中。被引用的表称为父表，而包含外键的表称为子表。子表中的外键

通常会引用父表中的主键。可以在 CREATE TABLE 或 ALTER TABLE 语句中定义外键。

必须通过 REFERENCE 子句建立外键约束。内联约束子句是列定义子句或对象属性子句的一部分。外联约束是关系属性子句或对象属性子句中的一部分。

如果参数 foreignKeyHandler 设置为 true（默认值），工具将这些语句迁移为注释语句。

DSC 支持内联和外联外键约束，如下所示。

输入：CREATE TABLE，使用外键和内联约束

```
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]
);
```

输出：

```
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] */
);
```

输入：CREATE TABLE，使用外键和外联约束

```
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
);
```

输出：

```
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 数据类型

定义为 LONG 的列可存储变长字符数据，最多可包含 2GB 信息。MT 支持表结构和 PL/SQL 中的 LONG 数据类型。

输入：在表结构中使用 LONG 数据类型

```
CREATE TABLE project ( proj_cd INT
                    , proj_name VARCHAR2(32)
                    , dept_no INT
                    , proj_det LONG );
```

输出：

```
CREATE TABLE project ( proj cd INT
                     , proj name VARCHAR2(32)
                     , dept_no INT
                     , proj_det TEXT );
```

输入：在 PL/SQL 中使用 LONG 数据类型

```
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;
/
```

输出：

```
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

将“MDSYS.MBRCOORDLIST”替换为“CLOB”。

Oracle 语法	迁移后语法
create table product_part (partid VARCHAR2(24), mbrcoords MDSYS.MBRCOORDLIST);	CREATE TABLE product_part (partid VARCHAR2(24), mbrcoords CLOB);

将“MDSYS.SDO_GEOMETRY”替换为“CLOB”。

Oracle 语法	迁移后语法
create table product_part (partid VARCHAR2(24), shape MDSYS.SDO GEOMETRY);	CREATE TABLE product_part (partid VARCHAR2(24), shape CLOB);

将“MDSYS.GEOMETRY”为“CLOB”。

Oracle 语法	迁移后语法
create table product_part (partid VARCHAR2(24), shape GEOMETRY);	CREATE TABLE product_part (partid VARCHAR2(24), shape CLOB);

列

xmax、xmin、left、right、maxvalue 为 Gauss 关键字，这些关键字应全字母大写并加英文双引号（""）。

Oracle 语法	迁移后语法
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));

间隔分区

对于间隔分区，应该注释分区。

Oracle 语法	迁移后语法
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 临时表

GaussDB(DWS)不支持 GLOBAL TEMPORARY TABLE。因此，它会迁移为 LOCAL TEMPORARY TABLE。

同样，也不支持 ON COMMIT DELETE ROWS。它会迁移为 ON COMMIT PRESERVE ROWS。

图6-7 **输入：临时表**

```
CREATE
    GLOBAL TEMPORARY TABLE
        schema1.temp_tbl1 (
            col1 VARCHAR2 (400)
            ,col2 DATE NOT NULL
        )
            ON COMMIT DELETE ROWS
;
```

图6-8 **输出：临时表**

```
CREATE
    LOCAL TEMPORARY TABLE
        schema1_temp_tbl1 (
            col1 VARCHAR2 (400)
            ,col2 DATE NOT NULL
        )
            ON COMMIT PRESERVE ROWS
;
```

6.9.2.3 全局临时表

全局临时表迁移为本地临时表。

输入：GLOBAL TEMPORARY TABLE

```
CREATE GLOBAL TEMPORARY TABLE
"Pack1"."GLOBAL_TEMP_TABLE"

( "ID" VARCHAR2(8)

) ON COMMIT DELETE ROWS ;
```

输出

```
CREATE
    LOCAL TEMPORARY TABLE

"Pack1_GLOBAL_TEMP_TABLE" (

"ID" VARCHAR2 (8)
        )
         ON COMMIT PRESERVE ROWS ;
```

6.9.2.4 索引

在 GaussDB(DWS)中创建索引期间，索引名不能与模式名一起指定。该索引将在创建索引表的模式中自动创建。

图6-9 **输入：索引**

```
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
    )
```

图6-10 **输出：索引**

```
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
    )
```

输入：基于 CASE 函数的索引

函数索引是基于列函数或表达式计算结果创建的索引。

输入

```
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)
    ) ;
```

说明

需要将表达式或函数放在()里。

输入：基于 DECODE 函数的索引

```
CREATE UNIQUE index GCC_PLAN_N2
        ON GCC_PLAN.GCC_PLAN_T (
        DECODE (
            ENABLE_FLAG
            ,'Y'
            ,BUSINESS_ID
            ,NULL
        )
    ) ;
```

输出

```
CREATE UNIQUE index GCC_PLAN_N2
        ON GCC_PLAN.GCC_PLAN_T (
        (DECODE (
            ENABLE_FLAG
            ,'Y'
            ,BUSINESS_ID
```

```
            ,NULL
        ))
    ) ;
```

说明

需要将表达式或函数放在()里。

ORA_HASH

ORA_HASH 函数用于计算给定表达式或列的哈希值。如果在 CREATE INDEX 中为列指定了此函数，则此函数将被删除。

输入

```
create index SD_WO.WO_WORK_ORDER_T_N3 on SD_WO.WO_WORK_ORDER_T (PROJECT_NUMBER,
ORA_HASH(WORK_ORDER_NAME));
```

输出

```
CREATE
index WO_WORK_ORDER_T_N3
ON SD_WO.WO_WORK_ORDER_T (
PROJECT_NUMBER
,ORA_HASH( WORK_ORDER_NAME )
) ;
```

DECODE

如果在 CREATE INDEX 语句中给列加上 DECODE 函数，则上报 syntax error at or near 'DECODE' (Script - gcc_plan_t.SQL)错误。

输入

```
create unique index GCC_PLAN.GCC_PLAN_N2 on GCC_PLAN.GCC_PLAN_T
(DECODE(ENABLE_FLAG,'Y',BUSINESS_ID,NULL));
```

输出

```
CREATE
UNIQUE index GCC_PLAN_N2
ON GCC_PLAN.GCC_PLAN_T (
DECODE (
ENABLE_FLAG
,'Y'
,BUSINESS_ID
,NULL
)
) ;
```

CASE 语句

CREATE INDEX 中不支持 CASE 语句。

输入

```
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)
    ) ;
```

输出

```
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 视图

视图是一个基于一个或多个表或视图的逻辑表。视图本身不含数据。

在输入中，如果表名称前没有模式名称修饰，则在输出中，会修改为用视图的模式名去修饰表。

图6-11 输入：视图

```
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'
            )
;
```

图6-12 输出：视图

```
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 序列

SEQUENCE 是 Oracle 对象，用于创建数字序列号。该序列用于创建自动编号字段，可用作主键。

如果参数 MigSupportSequence 设为 true（默认值），则在 PUBLIC 模式中创建序列。

说明

- CACHE 和 ORDER 参数不支持迁移。
- Oracle 中，序列的 MAXVALUE 最大可设为 999999999999999999999999999。GaussDB(DWS) 中，MAXVALUE 最大可设为 9223372036854775807。
- 在迁移序列之前，请复制 custom_scripts.SQL 文件的内容，并在所有目标数据库中执行此脚本。详情请参见 6.7.1 迁移流程。

序列

输入：CREATE SEQUENCE

```
CREATE SEQUENCE GROUP_DEF_SEQUENCE
  minvalue 1
  maxvalue 10000000000000000000
  start with 1152
  increment by 1
  cache 50
  order;
```

输出

```
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 和 NOCACHE

输入：CREATE SEQUENCE，使用 NOCACHE

```
CREATE SEQUENCE customers_seq
 START WITH 1000
 INCREMENT BY   1
 NOCACHE
 NOCYCLE;
```

输出

```
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
    )
;
```

CREATE SEQUENCE 和模式名

输入：CREATE SEQUENCE，使用模式名

```
CREATE SEQUENCE scott.seq_customers
 START WITH 1000 INCREMENT BY 1
 MINVALUE 1000 MAXVALUE 999999999999999
 CACHE 20 CYCLE ORDER;
```

输出

```
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 和默认值

输入：SEQUENCE，使用默认值

```
CREATE SEQUENCE seq_orders;
```

输出

```
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

要迁移 NEXTVAL，用户可使用自定义函数，根据 increment_by、max_value、min_value 和 cycle 生成下一个值。在 MT 安装过程中，需在要执行迁移的所有数据库中创建该函数。

该函数支持 DWS 所有版本。

NEXTVAL 是 Oracle 系统函数，GaussDB(DWS)不隐式支持该函数。为了支持该函数，DSC 会在 PUBLIC 模式中创建一个 NEXTVAL 函数。迁移后的语句会使用该 PUBLIC.NEXTVAL 函数。

说明

将参数 MigSupportSequence 设为 true，可将 NEXTVAL 迁移为 PUBLIC.NEXTVAL('[schema].sequence')。

将参数 MigSupportSequence 设为 false，可将 NEXTVAL 迁移为 NEXTVAL('[schema].sequence')。

在使用此函数之前，请复制 custom_scripts.SQL 文件的内容，并在所有目标数据库中执行此脚本。详情请参见 6.7.1 迁移流程。

输入：NEXTVAL

```
[schema.]sequence.NEXTVAL
```

输出

```
PUBLIC.nextval('[schema.]sequence')
```

输入：NEXTVAL

```
SELECT
        EMP_ID_SEQ.NEXTVAL INTO
            SEQ_NUM
        FROM
            dual
;
```

输出

```
SELECT
        PUBLIC.NEXTVAL ('EMP_ID_SEQ') INTO
            SEQ_NUM
        FROM
            dual
;
```

CURRVAL

要迁移 CURRVAL，用户可使用自定义函数，获取序列当前值。在 MT 安装过程中，需在要执行迁移的所有数据库中创建该函数。

CURRVAL 是 Oracle 系统函数，GaussDB(DWS)不隐式支持该函数。为了支持该函数，DSC 会在 PUBLIC 模式中创建一个 CURRVAL 函数。迁移后的语句会使用该 PUBLIC.CURRVAL 函数。

说明

将参数 MigSupportSequence 设为 true，可将 CURRVAL 迁移为 PUBLIC.CURRVAL('[schema].sequence')。

将参数 MigSupportSequence 设为 false，可将 CURRVAL 迁移为 CURRVAL('[schema].sequence')。

在使用此函数之前，请复制 custom_scripts.SQL 文件的内容，并在所有目标数据库中执行此脚本。详情请参见 6.7.1 迁移流程。

输入：CURRVAL

```
[schema.]sequence.CURRVAL
```

输出

```
currval('[schema.]sequence')
```

输入：CURRVAL

```
INSERT
    INTO
        Line_items_tab (
            Orderno
            ,Partno
            ,Quantity
        )
    VALUES (
        Order_seq.CURRVAL
        ,20321
        ,3
```

```
    )
;
```

输出

```
INSERT
    INTO
        Line_items_tab (
            Orderno
            ,Partno
            ,Quantity
        ) SELECT
                PUBLIC.CURRVAL ('Order_seq')
                ,20321
                ,3
;
```

6.9.2.7 PURGE

在 Oracle 中，DROP TABLE 语句用于将一张表放入回收站，而 PURGE 语句用于将表或索引从回收站彻底删除，并释放所有与该对象相关的空间，或清空整个回收站，或从回收站中彻底删除一个已删除表空间的部分或全部内容。迁移后查询不含 PURGE。

图6-13 输入：PURGE

```
Execute immediate 'Drop table
table1 purge' ;

drop table test.emp purge ;
```

图6-14 输出：PURGE

```
Execute immediate 'Drop table table1' ;


drop table test.emp ;
```

6.9.2.8 数据库关键字

DSC 支持 GaussDB(DWS)关键字，如 NAME、LIMIT、OWNER、KEY 和 CAST。这些关键字必须放在双引号内。

Gauss 关键字（NAME/VERSION/LABEL/POSITION）

NAME, VERSION, LABEL, POSITION 关键字迁移为 AS *关键字*。

输入：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;
```

输出

```
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 和 YEAR

输入：TEXT, YEAR

```
SELECT
  NAME,
  VALUE,
  DESCRIPTION TEXT,
  JOINED YEAR,
  LIMIT
FROM
  EMPLOYEE;

SELECT
  NAME,
  TEXT,
  YEAR,
  VALUE,
  DESCRIPTION,
  LIMIT
FROM
  EMPLOYEE_DETAILS;
```

输出

```
SELECT
  "NAME",
  VALUE,
  DESCRIPTION AS TEXT,
  JOINED AS YEAR,
  "LIMIT"
FROM
  EMPLOYEE;

SELECT
  "NAME",
  "TEXT",
  "YEAR",
  VALUE,
  DESCRIPTION,
  "LIMIT"
FROM
  EMPLOYEE_DETAILS;
```

NAME 和 LIMIT

输入：GaussDB(DWS)关键字 NAME 和 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;
```

输出

```
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 操作

输入：使用 SELECT 查询 GaussDB(DWS)关键字 OWNER

```
SELECT
        owner
    FROM
        Test_Col;
```

输出

```
SELECT
        "OWNER"
    FROM
        Test_Col;
```

输入：DELETE，GaussDB(DWS)关键字 OWNER

```
DELETE FROM emp14
    WHERE
        ename = 'Owner';
```

输入

```
DELETE FROM emp14
    WHERE
        ename = 'Owner'
```

KEY

Blogic 操作

输入：GaussDB(DWS)关键字 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 ;
/
```

输出

```
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 ;
```

范围、账号和语言

当 Gauss 关键字用作 SELECT 列表中任意列的别名且没有使用 AS 时，需要使用“AS *关键字*”的格式来定义别名。

输入

```
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 ;
```

输出

```
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 ;
```

主键和唯一键

如果在建表时声明了主键和唯一键两个约束，仅迁移主键。

```
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);
```

输出

```
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 命令

PROMPT 命令应转换成 Gauss 支持的\ECHO 命令。

Oracle 语法	迁移后语法
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 短语

输入：COMPRESS 短语

```
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)
);
```

输出

```
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 索引

该功能通过 BitmapIndexSupport 设置，迁移过程中默认注释掉 Bitmap 索引。

输入：Bitmap 索引

```
CREATE BITMAP INDEX
emp bitmap idx
ON index_demo (gender);
```

输出

```
/*CREATE BITMAP INDEX emp_bitmap_idx ON index_demo (gender);*/
```

如果 BitmapIndexSupport 设置为 BTREE，迁移结果如下:

输出

```
CREATE
    /*bitmap*/
    INDEX emp_bitmap_idx
        ON index_demo
        USING btree (gender) ;
```

6.9.5 自定义表空间

输入： 自定义表空间

```
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
            )
        )
```

输出

```
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 附加日志数据

可以在重做日志文件中记录附加列。 记录这些附加列的过程称为补充日志记录。Oracle 支持此功能，高斯不支持此功能。

输入

```
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
  ) ;
```

输出

```
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 */
  ) ;
```

说明

高斯不支持的补充日志数据功能，需要注释掉。

CREATE TABLE 不支持“SUPPLEMENTAL LOG DATA”，因此需要注释掉。

输入

```
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
  );
```

输出

```
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

CREATE TABLE 不支持“Data type LONG RAW”，需要用 Bytea 来替换 Long Raw 数据类型。

输入

```
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)
 );
```

输出

```
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 是内嵌函数，返回表中某一行的全域唯一识别元（GUID）。SYS_GUID 不使用参数，返回一个 16 字节的 RAW 值。

输入

```
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*/
 );
```

输出

```
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（Oracle）

本节主要介绍 Oracle DML 的迁移语法。迁移语法决定了关键字/功能的迁移方式。

具体见以下节点内容：

SELECT

INSERT

MERGE

SELECT

概述

Oracle 的 SELECT 语句可以启动查询，使用一个可选的 ORDER BY 子句，该子句用于从数据库的一个或多个表中提取记录。

输入：SELECT

```
SELECT col1, col2
  FROM tab1;
```

输出

```
SELECT col1, col2
  FROM tab1;
```

1. **子句顺序**

   HAVING 子句必须出现在 GROUP BY 子句后面，而 Oracle 允许 HAVING 在 GROUP BY 子句之前或之后。在目标数据库中，HAVING 子句被移至 GROUP BY 子句之后。

图6-15 输入：子句顺序

```
SELECT
        DEPTNO
        ,COUNT( * )
        ,SUM (SAL)
    FROM
        EMP
    WHERE
        JOB = 'CLERK'
    HAVING
        SUM (SAL) >= 500
    GROUP BY
        DEPTNO


    ORDER BY
        DEPTNO
;
```

图6-16 输出：子句顺序

```
1  SELECT
2          DEPTNO
3          ,COUNT( * )
4          ,SUM (SAL)
5      FROM
6          EMP
7      WHERE
8          JOB = 'CLERK'
9      GROUP BY
10         DEPTNO
11     HAVING
12         SUM (SAL) >= 500
13     ORDER BY
14         DEPTNO
15 ;
16
```

2. **扩展 Group By 子句**

   指定 GROUP BY 子句可让数据库将所选行基于 expr(s)的值分组。如果该子句包含 CUBE，ROLLUP，或 GROUPING SETS 扩展项，则数据库除正则分组外还会进行超聚合分组。这些功能在 GaussDB(DWS)中不可用，可通过 UNION ALL 操作符实现。

图6-17 输入：扩展 Group By 子句

```
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
        )
;
```

图6-18 输出：扩展 Group By 子句

```
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 和 ROLLUP

GROUPING_ID 会返回一个数字，该数字与关联到某行的 GROUPING 位向量相对应。GROUPING_ID 仅适用于包含 GROUP BY 扩展项的 SELECT 语句，例如 ROLLUP 操作符和 GROUPING 函数。在包含多个 GROUP BY 表达式的查询中，要确定特定行的 GROUP BY 级别，需要使用多个 GROUPING 函数，这可能导致 SQL 语句变得复杂。在这种情况下，可使用 GROUPING_ID 避免语句复杂化。

3. **括号中的表名**

表名不需要在括号内指定，而 Oracle 允许使用括号。

图6-19 输入：括号中的表名

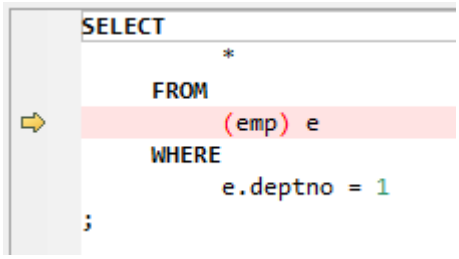
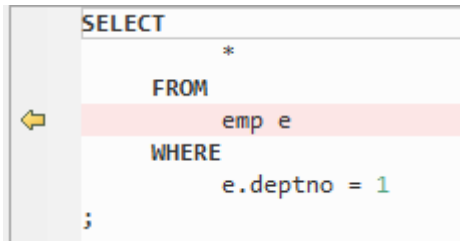


图6-20 输出：括号中的表名



4. **UNIQUE 关键字**

UNIQUE 关键字迁移为 DISTINCT 关键字

输入： 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);
```

输出

```
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**

输入：CLIENT_INFO

返回用户会话信息。

```
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' ;
```

输出

```
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)

包中的函数转换后，不删除结束后的函数标记。4. sad_lookup_contract_pkg.bdy 中的 svproduct_is_for_pa 函数被使用。

USERENV('CLIENT_INFO')

过程中使用的 USERENV。迁移过程因工具而失败。

```
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 切换合同不提供数据  */
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'
```

输入：

```
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;
/
```

输出：

```
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

概述

Oracle INSERT 语句用于将单个记录或多个记录插入到表中。

NOLOGGING

在插入的脚本中对 NOLOGGING 进行注释。

Oracle 语法	迁移后语法
`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**

   Oracle 的 INSERT ALL 语句可通过单个 INSERT 语句向单个或多个表中插入多行。目标查询将转化为公用表表达式（CTE）。

   图6-21 输入：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
```

图6-22 输出：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**

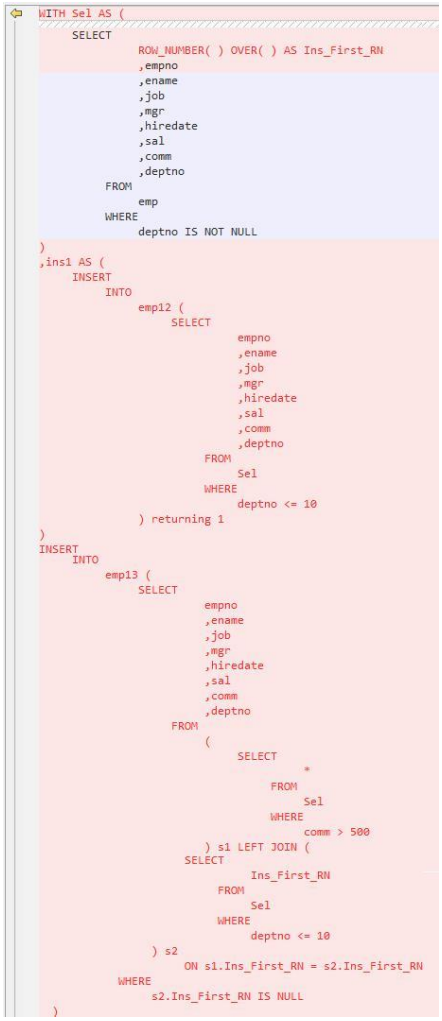
Oracle 的 INSERT FIRST 语句用于在 first 条件为真时执行 INSERT 语句，而其他语句会被忽略。目标查询将转化为公用表表达式。

图6-23 输入：INSERT FIRST

```
INSERT
     FIRST WHEN deptno <= 10
     THEN INTO
          emp12 WHEN comm > 500
          THEN INTO
               emp13 SELECT

                          empno
                          ,ename
                          ,job
                          ,mgr
                          ,hiredate
                          ,sal
                          ,comm
                          ,deptno
                     FROM
                          emp
                     WHERE
                          deptno IS NOT NULL
```

图6-24 输出：INSERT FIRST



```
WITH Sel AS (
     SELECT
               ROW_NUMBER( ) OVER( ) AS Ins_First_RN
               ,empno
               ,ename
               ,job
               ,mgr
               ,hiredate
               ,sal
               ,comm
               ,deptno
          FROM
               emp
          WHERE
               deptno IS NOT NULL
)
,ins1 AS (
     INSERT
          INTO
               emp12 (
                    SELECT
                              empno
                              ,ename
                              ,job
                              ,mgr
                              ,hiredate
                              ,sal
                              ,comm
                              ,deptno
                         FROM
                              Sel
                         WHERE
                              deptno <= 10
               ) returning 1
)
INSERT
     INTO
          emp13 (
               SELECT
                         empno
                         ,ename
                         ,job
                         ,mgr
                         ,hiredate
                         ,sal
                         ,comm
                         ,deptno
                    FROM
                         (
                              SELECT
                                        *
                                   FROM
                                        Sel
                                   WHERE
                                        comm > 500
                              ) s1 LEFT JOIN (
                                SELECT
                                          Ins_First_RN
                                     FROM
                                          Sel
                                     WHERE
                                          deptno <= 10
                           ) s2
                              ON s1.Ins_First_RN = s2.Ins_First_RN
                     WHERE
                          s2.Ins_First_RN IS NULL
     )
```

3. **INSERT（使用表别名）**

Oracle 表别名通过为查询中的表分配名称或代码，用于声明和提高可读性。INSERT with Alias 可与 INSERT INTO 语句一起使用。DSC 可迁移含有表别名的 INSERT INTO 语句。

a. **Blogic 操作**

输入：INSERT，使用表别名

```
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 ;
/
```

输出

```
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 操作**

输入：INSERT，使用表别名

```
INSERT
    INTO
        Public.emp14 ats (
            ats.empno
            ,ats.ename
        )
    VALUES (
        3
        ,'Categories'
    )
;
```

输出

```
INSERT
    INTO
        Public.emp14 (
            empno
            ,ename
        ) SELECT
                3
                ,'Categories'
;
```

输入：INSERT，使用表别名

```
INSERT
    INTO
        "abc" . "emp18" wmc (
            wmc.empno
            ,wmc.ename
        ) SELECT
                wmc.empno
                ,wm_concat (wmc.ename) AS eName
            FROM
                emp16 wmc
            GROUP BY
                empno
;
```

输出

```
INSERT
    INTO
        "abc" . "emp18" (
            empno
            ,ename
        ) SELECT
                wmc.empno
                ,STRING_AGG (
                    wmc.ename
                    ,','
                ) AS eName
            FROM
                emp16 wmc
            GROUP BY
                empno
;
```

输入：INSERT，使用表别名

```
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
                )
;
```

输出

```
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 是一种 ANSI 标准的 SQL 语法运算符，用于从一个或多个源中选择行来更新或插入表或视图。用户可指定更新或插入目标表或视图的条件。

目前，6.5.0 及之后版本的 GaussDB(DWS)支持此功能。DSC 使用多种方法将 MERGE 迁移到 GaussDB(DWS)兼容的 SQL 中。

配置参数 mergeImplementation：

- 默认设置为 WITH。设为此值时，目标查询将转换成公用表表达式。

图6-25 输入：MERGE（1）

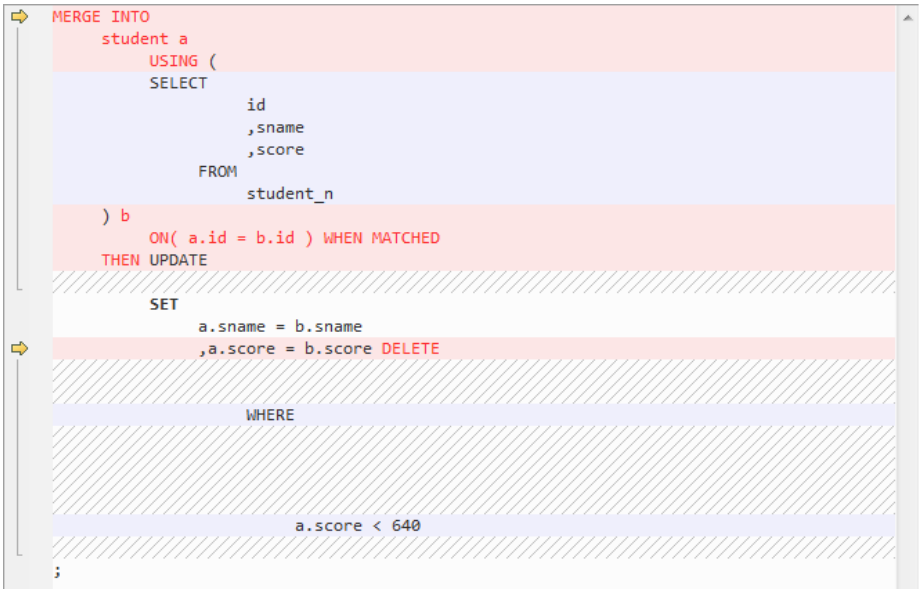


图6-26 输出：MERGE（2）

```
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
;
```

- 也可设置为 SPLIT。设为此值时，MERGE 语句将被分解为多个 INSERT 和 UPDATE 语句。

图6-27 输入：MERGE（3）

```
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);
```

图6-28 输出：MERGE（4）

```
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 伪列

本节主要介绍 Oracle 伪列的迁移语法。迁移语法决定了关键字/功能的迁移方式。

伪列与表的列类似，但不存储在表中。用户可在伪列中进行 SELECT 操作，但无法插入、更新、或删除其中的值。

ROWID

ROWID 伪列返回特定行的具体地址。

图6-29 输入：ROWID

```
SELECT
          empid
          ,ename
          ,ROWID
     FROM
          employees
;
```

图6-30 输出：ROWID

```
SELECT
        empid
        ,ename
        ,CAST( ( xc_node_id || '#' || tableoid || '#' || ctid ) AS TEXT ) AS rowid
    FROM
        employees
;
```

ROWNUM

对于查询返回的每行数据，ROWNUM 伪列段会返回一个数字，表示 Oracle 从一个表或一组连接的行中选择行的顺序。选择的第一行的 ROWNUM 为 1，第二行为 2，以此类推。

图6-31 输入：ROWNUM

```
SELECT
        e.empid
        ,e.ename
    FROM
        employees e
    WHERE
        ROWNUM < 6
;
```

图6-32 输出：ROWNUM

```
SELECT
        e.empid
        ,e.ename
    FROM
        employees e LIMIT 6 - 1
;
```

输入：ROWNUM，使用 UPDATE

执行 UPDATE 时，如果使用了具有某个值（整数）的 ROWNUM，系统将根据 ROWNUM 附近使用的运算符更新记录。

```
UPDATE SCMS_MSGPOOL_LST
          SET MSG_STD = '11'
        WHERE UNISEQNO = IN_OUNISEQNO
          AND MSG_TYP1 IN ('MT103', 'MT199')
          AND ROWNUM = 1;
```

输出

```
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)
```

输入：ROWNUM，使用 DELETE

执行 DELETE 时，如果使用了具有某个值（整数）的 ROWNUM，系统将根据 ROWNUM 附近的运算符依次删除记录。

```
delete from test1
where c1='abc' and rownum = 1;
```

输出

```
delete from test1 where (xc_node_id,ctid) in (select xc_node_id, ctid from test1
where c1='abc' limit 1);
```

输入：UPDATE，使用 ROWNUM

使用 ROWNUM 迁移的 UPDATE 和 DELETE 脚本包含 LIMIT，高斯不支持。

```
UPDATE SCMS_MSGPOOL_LST
 SET MSG_STD = '11'
 WHERE UNISEQNO = IN_OUNISEQNO
 AND MSG_TYP1 IN ('MT103', 'MT199')
 AND ROWNUM = 1;
```

输出

```
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
      );
```

输入：DELETE，使用 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;
```

输出

```
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

本节主要介绍 Oracle OUTER JOIN 的迁移语法。迁移语法决定了关键字/功能的迁移方式。

OUTER JOIN 会返回所有满足关联条件的行。此外，如果无法为一个表中的某些行在另一个表中找到任何满足关联条件的行，则该语句会返回这些行。在 Oracle 中：

- 通过在 WHERE 条件中对表 B 的所有字段使用外连接操作符“+”，表 A 和 B 的左外连接返回表 A 中的所有行和所有满足关联条件的行。
- 通过在 WHERE 条件中对表 A 的所有字段使用外连接操作符“+”，表 A 和 B 的右外连接返回表 B 中的所有行和所有满足关联条件的行。

GaussDB(DWS)不支持“+”操作符。该操作符的功能通过 LEFT OUTER JOIN 和 RIGHT OUTER JOIN 关键词实现。

图6-33 输入：OUTER JOIN

```
SELECT
        empno
        ,ename
        ,job
        ,dname
        ,loc
    FROM
        emp
        ,dept
    WHERE
        emp.deptno = dept.deptno (+)
        AND salary > 50000
;
```

图6-34 输出：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 (+)

GaussDB 18.2.0 支持 JOIN，因此添加 **supportJoinOperator** 配置参数。

设置 **supportJoinOperator=false** 后 OUTER QUERY (+)可迁移。

输入：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 ;
```

输出

```
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

输入：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 ;
```

输出

```
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
;
```

输入：多表 CONNECT BY

说明了每个子行与父行的关系。该语法使用 CONNECT BY *xxx* PRIOR 子句定义当前行（子行）与前一行（父行）的关系。

```
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;
```

输出

```
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 系统函数

本节主要介绍 Oracle 系统函数的迁移语法。迁移语法决定了关键字/特性的迁移方式。

本节包括以下内容：

日期函数、LOB 函数、字符串函数、分析函数以及正则表达式函数，具体内容详见 6.9.14.1 日期函数~6.9.14.5 正则表达式函数章节。

6.9.14.1 日期函数

本节介绍如下日期函数：

- ADD_MONTHS
- DATE_TRUNC
- LAST_DAY
- MONTHS_BETWEEN
- SYSTIMESTAMP

ADD_MONTHS

ADD_MONTHS 是 Oracle 系统函数，GaussDB(DWS)中并不隐式支持该函数。

说明

在使用此函数之前，请执行如下操作：

1. 创建并使用 MIG_ORA_EXT 模式。
2. 复制 custom scripts 文件的内容，并在要执行迁移的所有目标数据库中执行此脚本。详情请参见 6.7.1 迁移流程。

ADD_MONTHS 返回带月份的日期。

- date 参数为 datetime 类型。
- integer 参数为 integer 类型。

返回类型为 date。

输入：ADD_MONTHS

```
SELECT
        TO_CHAR( ADD_MONTHS ( hire_date ,1 ) ,'DD-MON-YYYY' ) "Next month"
    FROM
        employees
    WHERE
        last_name = 'Baer'
;
```

输出

```
SELECT
        TO_CHAR( MIG_ORA_EXT.ADD_MONTHS ( hire_date ,1 ) ,'DD-MON-YYYY' ) "Next
```

```
month"
    FROM
        employees
    WHERE
        last_name = 'Baer'
;
```

TO_DATE（使用第三个参数）

TO_DATE(' 2019-05-02 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')中的第三个参数需要加注释。

输入

```
 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'))
  );
```

输出

```
 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（使用 SYYYY 年份格式）

日期格式不支持 SYYYY，适用 GaussDB T。

输入

```
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'))
  );
```

输出

```
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

DATE_TRUNC 函数返回日期，将日期的时间部分截断为格式模型 fmt 指定的单位。

输入

```
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;
```

输出

```
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

Oracle 的 LAST_DAY 函数根据 date（日期）值返回该月份的最后一天。

```
LAST_DAY(date)
```

不论 date 的数据类型如何，返回类型始终为 DATE。

LAST_DAY 是 Oracle 的系统函数，GaussDB(DWS)不隐式支持该函数。要支持此函数，DSC 会在 MIG_ORA_EXT 模式中创建一个 LAST_DAY 函数。迁移后的语句将使用此新函数 MIG_ORA_EXT.LAST_DAY，如下示例。

说明

在使用此函数之前，请执行如下操作：

1. 创建并使用 MIG_ORA_EXT 模式。
2. 复制 custom scripts 文件的内容，并在要执行迁移的所有目标数据库中执行此脚本。详情请参见 6.7.1 迁移流程。

输入：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;
```

输出

```
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

MONTHS_BETWEEN 函数返回两个日期之间的月份数。

MONTHS_BETWEEN 是 Oracle 系统函数，GaussDB(DWS)并不隐式支持该函数。要支持此函数，DSC 需在 MIG_ORA_EXT 模式中创建一个 MONTHS_BETWEEN 函数。迁移后的语句将使用此新函数 MIG_ORA_EXT.MONTHS_BETWEEN，如下所示。

说明

在使用此函数之前，请执行如下操作：

1. 创建并使用 MIG_ORA_EXT 模式。
2. 拷贝 custom scripts 文件中的内容，并在要执行迁移的所有目标数据库中执行此脚本。详情请参见 6.7.1 迁移流程。

输入：MONTHS_BETWEEN

```
Select Months_Between(to_date('2017-06-20', 'YYYY-MM-DD'), to_date('2011-06-20',
'YYYY-MM-DD')) from dual;
```

输出

```
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

SYSTIMESTAMP 函数返回数据库所在系统的系统日期，包括精确到小数的秒和时区。返回类型为 TIMESTAMP WITH TIME ZONE。

图6-35 输入：SYSTIMESTAMP

```
SELECT
        SYSTIMESTAMP
    FROM
        tab1
;
```

图6-36 输出：SYSTIMESTAMP

```
SELECT
        CURRENT_TIMESTAMP
    FROM
        tab1
;
```

6.9.14.2 LOB 函数

本节介绍如下 LOB 函数：

- DBMS_LOB.APPEND
- DBMS_LOB.COMPARE
- DBMS_LOB.CREATETEMPORARY
- DBMS_LOB.INSTR
- DBMS_LOB.SUBSTR

DBMS_LOB.APPEND

DBMS_LOB.APPEND 函数将源 LOB 的内容追加到指定的 LOB。

输入：DBMS_LOB.APPEND

```
[sys.]dbms_lob.append(o_menuxml, to_clob('DSJKSDAJKSFDA'));
```

输出

```
o_menuxml := CONCAT(o_menuxml, CAST('DSJKSDAJKSFDA' AS CLOB));
```

输入：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 ;
/
```

输出

```
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 函数比较两个 LOB 的所有/部分内容。DBMS_LOB.COMPARE 是 Oracle 系统函数，GaussDB(DWS)并不隐式支持该函数。要支持此函数，DSC 需在 MIG_ORA_EXT 模式中创建一个 COMPARE 函数。迁移后的语句将使用此新函数 MIG_ORA_EXT.MIG_CLOB_COMPARE，SQL 示例如下：

在 SQL 中使用 COMPARE

输入：在 SQL 中使用 DBMS_LOB.COMPARE

```
SELECT a.empno ,dbms_lob.compare ( col1 ,col2 ) FROM emp a ,emp b ;
```

输出

```
SELECT a.empno ,MIG_ORA_EXT.MIG_CLOB_COMPARE ( col1 ,col2 ) FROM emp a ,emp b ;
```

输入：在 SQL 中使用 DBMS_LOB.COMPARE，其中 CREATE TABLE 使用 5 个参数

```
CREATE TABLE abc nologging AS SELECT dbms_lob.compare ( col1 ,col2 ,3 ,5 ,4 ) FROM
emp a ,emp b ;
```

输出

```
CREATE UNLOGGED TABLE abc AS ( SELECT MIG_ORA_EXT.MIG_CLOB_COMPARE
( col1 ,col2 ,3 ,5 ,4 ) FROM emp a ,emp b ) ;
```

输入：在函数（NVL2）的 SQL 中使用 DBMS_LOB.COMPARE

```
SELECT REPLACE( NVL2( DBMS_LOB.COMPARE ( ENAME ,Last_name ) ,'NO NULL' ,'ONE
NULL' ) ,'NULL' ) FROM emp ;
```

输出

```
SELECT REPLACE( DECODE ( MIG_ORA_EXT.MIG_CLOB_COMPARE
( ENAME ,Last_name ) ,NULL ,'ONE NULL' ,'NO NULL' ) ,'NULL' ,'' ) FROM emp ;
```

在 PL/SQL 中使用 COMPARE

输入：在 PL/SQL 中使用 DBMS_LOB.COMPARE

```
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;
/
```

输出

```
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

DBMS_LOB.CREATETEMPORARY 函数在用户默认的临时表空间中创建一个临时 LOB 及其对应索引。DBMS_LOB.FREETEMPORARY 用于删除临时 LOB 及其索引。

输入：DBMS_LOB.CREATETEMPORARY 和 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;
/
```

输出

```
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

DBMS_LOB.FREETEMPORARY 函数释放默认临时表空间中的临时 BLOB 或 CLOB。在调用 FREETEMPORARY 之后，释放的 LOB 定位器标记为无效。

输入：DBMS_LOB.CREATETEMPORARY 和 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;
/
```

输出

```
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 函数从指定的偏移量开始，返回在 LOB 中第 n 次匹配模式的位置。

输入：在 SQL 中使用 DBMS_LOB.INSTR

```
SELECT expr1, …, DBMS_LOB.INSTR(str, septr, 1, 5)
  FROM tab1
 WHERE …;
```

输出

```
SELECT expr1, …, INSTR(str, septr, 1, 5)
  FROM tab1
 WHERE …
```

输入：在 PL/SQL 中使用 DBMS_LOB.INSTR

```
BEGIN
  …
      pos := DBMS_LOB.INSTR(str,septr,1, i);
  ...
END;
/
```

输出

```
BEGIN
  …
      pos := INSTR(str,septr,1, i);
  ...
END;
/
```

DBMS_LOB.SUBSTR

DBMS_LOB.SUBSTR 适用于 V1R8C10。通过配置参数 MigDbmsLob，用户可以指定迁移此函数还是直接保留。

输入：DBMS_LOB.SUBSTR，MigDbmsLob 设为 true

如果参数 MigDbmsLob 设为 true，则迁移。相反，如果参数 MigDbmsLob 设为 false，则不迁移。

输入

```
select dbms_lob.substr('!2d3d4dd!',1,5);
```

输出

```
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);
```

输入

```
select dbms_lob.substr('!2d3d4dd!',5);
```

输出

```
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 字符串函数（Oracle）

本节介绍如下字符串函数：

- LISTAGG
- STRAGG
- WM_CONCAT
- NVL2 和 REPLACE
- QUOTE

LISTAGG

LISTAGG 根据 ORDER BY 子句对每个组中的列值进行排序，并将排序后的结果拼接起来。

图6-37 输入：LISTAGG

```
SELECT
        deptno
        ,ename
        ,LISTAGG (
            ename
            ,': '
        ) OVER( PARTITION BY deptno ,ename ORDER BY ename ) AS rn
    FROM
        emp
    ORDER BY
        deptno
        ,ename
;
```

图6-38 输出：LISTAGG

```
SELECT
          deptno
          ,ename
          ,STRING_AGG (
               ename
               ,': '
          ) OVER( PARTITION BY deptno ,ename ORDER BY ename ) AS rn
     FROM
          emp
     ORDER BY
          deptno
          ,ename
;
```

设置 MigSupportForListAgg=false 后，可迁移 LISTAGG。

输入：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);
```

输出

```
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 是一个字符串聚合函数，用于将多个行的值收集到一个用逗号分隔的字符串中。

输入：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;
```

输出

```
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 用于将多行的数据聚合到一行中，提供与特定值相关联的数据列表。

图6-39 输入：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
;
```

图6-40 输出：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 和 REPLACE

“NVL2(表达式,值 1,值 2)”函数用于根据指定的表达式是否为空来确定查询返回的值。如果表达式不为 Null，则 NVL2 返回“值 1”。如果表达式为 Null，则 NVL2 返回”值 2“。

输入：NVL2

```
NVL2(Expr1, Expr2, Expr3)
```

输出

```
DECODE(Expr1, NULL, Expr3, Expr2)
```

REPLACE 函数用于返回 char，将所有 search_string 替换为 replacement_string。如果将 replacement_string 省略或留空，则会删除所有出现的 search_string。

在 Oracle 中，REPLACE 函数有两个必选参数，一个可选参数。GaussDB(DWS)中的 REPLACE 函数有三个必选参数。

输入：嵌套的 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 ;
/
```

输出

```
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 ;
/
```

输入：多个 REPLACE

```
SELECT
        REPLACE( 'JACK and JUE' ,'J', '' ) "Changes"
        ,REPLACE( 'JACK1 and JUE' ,'J' ) "Changes1"
        ,REPLACE( 'JACK2 and JUE' ,'J' ) "Changes2"
    FROM
        DUAL
;
```

输出

```
SELECT
        REPLACE( 'JACK and JUE' ,'J' ,'' ) "Changes"
        ,REPLACE( 'JACK1 and JUE' ,'J' ,'' ) "Changes1"
        ,REPLACE( 'JACK2 and JUE' ,'J' ,'' ) "Changes2"
    FROM
        DUAL
;
```

输入：REPLACE，使用 3 个参数

```
SELECT
        REPLACE( '123tech123' ,'123', '1')
```

```
    FROM
        dual
;
```

输出

```
SELECT
        REPLACE( '123tech123' ,'123' , '1' )
    FROM
        dual
;
```

QUOTE

QUOTE 允许用户在文字字符串中嵌入单引号而非使用双引号，即可以使用单引号指定一个文字字符串。

示例：

```
SELECT q'[I'm using quote operator in SQL statement]' "Quote (q) Operator" FROM
dual;
```

图6-41 输入：引号

```
SELECT
          q'[It's a string quote operator.]'
     FROM
          dual
;
```

图6-42 输出：引号

```
SELECT
          $q$It's a string quote operator.$q$
     FROM
          dual
;
```

6.9.14.4 分析函数

分析函数根据一组行计算一个聚合值。它与聚集函数的不同之处在于，它为每个组返回多行。分析函数通常用于计算累积值，数据移动值，中间值和报告聚合值。DSC 支持分析函数，包括 RATIO_TO_REPORT 函数。

输入：分析函数

```
SELECT empno, ename,  deptno
  , COUNT(*) OVER() AS cnt
  , AVG(DISTINCT empno) OVER (PARTITION BY deptno) AS cnt_dst
  FROM emp
ORDER BY empno;
```

输出

```
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 是个分析函数，它会返回一个值与一组值的比例。

输入：RATIO_TO_REPORT

```
SELECT last_name, salary
         , RATIO_TO_REPORT(salary) OVER () AS rr
  FROM employees
WHERE job_id = 'PU_CLERK';
```

输出

```
SELECT last_name, salary
    , salary / NULLIF( SUM (salary) OVER( ), 0 ) AS rr
  FROM employees
WHERE job_id = 'PU_CLERK';
```

输入：RATIO_TO_REPORT，在 SELECT 中使用 AGGREGATE 列

```
SELECT
        Ename
        ,Deptno
        ,Empno
        ,SUM (salary)
        ,RATIO_TO_REPORT (
```

```
            COUNT( DISTINCT Salary )
        ) OVER( PARTITION BY Deptno ) RATIO
    FROM
        emp1
    ORDER BY
        Ename
        ,Deptno
        ,Empno
;
```

输出

```
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
;
```

输入：RATIO_TO_REPORT，且 AGGREGATE 列使用扩展分组功能，但 RATIO TO REPORT 列的 COUNT（Salary）不在 SELECT 字段列表中

可以使用 extendedGroupByClause 参数来配置扩展 GROUP BY 子句的迁移。

```
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
;
```

输出

```
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 正则表达式函数

正则表达式使用标准化的语法约定来指定匹配字符串的模式。在 Oracle 中，正则表达式通过一组允许用户搜索和操作字符串数据的 SQL 函数来实现。

DSC 可迁移 REGEXP_INSTR、REGEXP_SUBSTR 和 REGEXP_REPLACE 正则表达式，详情如下：

- 不支持包含 sub_expr 参数的 Regexp（REGEXP_INSTR 和 REGEXP_SUBSTR）。若输入脚本包含 sub_expr，则 DSC 将为其记录为错误。
- Regexp（REGEXP_INSTR、REGEXP_SUBSTR 和 REGXP_REPLACE）使用 match_param 参数来设置默认的匹配行为。DSC 中，该参数仅支持“i”值（匹配不区分大小写）和“c”值（匹配区分大小写），不支持其他值。
- Regexp（REGEXP_INSTR）使用 return_option 参数为 regexp 设置匹配的返回值。DSC 仅支持此参数设为 0，不支持其他值。

REGEXP_INSTR

REGEXP_INSTR 扩展了 INSTR 函数的功能，支持搜索字符串的正则表达式模式。DSC 可迁移含有 2 到 6 个参数的 REGEXP_INSTR。

sub_expr 参数（参数＃7）在 Oracle 中可用，但不支持迁移。如果输入脚本包含 sub_expr，DSC 会将其记录为错误。

支持将 return_option 设为 0，不支持其他值。

支持将 match_param 设为“i”（匹配不区分大小写）和“c”（匹配区分大小写），不支持其他值。

```
REGEXP_INSTR(
    string,
     pattern,
     [start_position,]
     [nth_appearance,]
     [return_option,]
     [match_param,]
     [sub_expr]
)
```

Bulk 操作

- **输入：REGEXP_INSTR**

```
SELECT
        REGEXP_INSTR( 'TechOnTheNet is a great resource' ,'t' )
    FROM
        dual
;
```

输出

```
SELECT
        MIG_ORA_EXT.REGEXP_INSTR (
            'TechOnTheNet is a great resource'
            ,'t'
        )
    FROM
        dual
;
```

- **输入：REGEXP_INSTR，使用 7 个参数（无效）**

```
SELECT
        Empno
        ,ename
```

```
        ,REGEXP_INSTR( ename ,'a|e|i|o|u' ,1 ,1 ,0 ,'i' ,7 ) AS Dname
    FROM
        emp19
;
```

输出

输入表达式含有 7 个参数，但 MT 仅允许 REGEXP_INSTR 包含 2 到 6 个参数，因此会记录错误 “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 操作

- **输入：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 ;
/
```

输出

```
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 通过支持搜索字符串的正则表达式模式来扩展 SUBSTR 函数的功能。可迁移含有 2 到 5 个参数的 REGEXP_SUBSTR。

sub_expr 参数（参数＃6）在 Oracle 中可用，但不支持迁移。如果输入脚本包含 sub_expr，则 DSC 会将其记录为错误。

支持将 match_param 设为“i”（匹配不区分大小写）和“c”（匹配区分大小写），不支持其他值。

```
REGEXP_SUBSTR(
   string,
    pattern,
    [start_position,]
    [nth_appearance,]
    [match_param,]
    [sub_expr]
 )
```

Bulk 操作

- **输入：REGEXP_SUBSTR**

```
SELECT
        Ename
        ,REGEXP_SUBSTR( 'Programming' ,'(\w).*?\1' ,1 ,1 ,'i' )
    FROM
        emp16
;
```

输出

```
SELECT
        Ename
        ,MIG_ORA_EXT.REGEXP_SUBSTR (
            'Programming'
            ,'(\w).*?\1'
            ,1
            ,1
            ,'i'
        )
    FROM
        emp16
;
```

- **输入：REGEXP_SUBSTR**

```
SELECT
        REGEXP_SUBSTR( '1234567890' ,'(123)(4(56)(78))' ,1 ,1 ,'i' )
"REGEXP_SUBSTR"
    FROM
        DUAL
;
```

输出

```
SELECT
        MIG_ORA_EXT.REGEXP_SUBSTR (
            '1234567890'
            ,'(123)(4(56)(78))'
            ,1
            ,1
            ,'i'
        ) "REGEXP_SUBSTR"
    FROM
```

```
        DUAL
;
```

- **输入：REGEXP_SUBSTR，使用 6 个参数（无效）**

```
SELECT
        REGEXP_SUBSTR( '1234567890' ,'(123)(4(56)(78))' ,1 ,1 ,'i' ,1 )
"REGEXP_SUBSTR"
    FROM
        DUAL
;
```

输出

输入表达式含有 6 个参数，但 MT 仅支持 REGEXP_SUBSTR 含有 2 到 5 个参数，所以会记录错误" 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 操作

- **输入：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 ;
/
```

输出

```
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 通过支持搜索字符串的正则表达式模式来扩展 REPLACE 函数的功能。可迁移含有 2 到 6 个参数的 REGEXP_REPLACE。

支持将 match_param 设为“i”（匹配不区分大小写）和“c”（匹配区分大小写），不支持其他值。

```
REGEXP_REPLACE(
     string,
     pattern,
     [replacement_string,]
     [start_position,]
     [nth_appearance,]
     [match_param]
 )
```

Bulk 操作

- **输入：REGEXP_REPLACE**

```
SELECT
        testcol
        ,regexp_replace( testcol ,'([[:digit:]]{3})\.([[:digit:]]{3})\.([[:digit:]]
{4})' ,'(\1) \2-\3' ) RESULT
    FROM
        test
    WHERE
        LENGTH( testcol ) = 12
;
```

输出

```
SELECT
        testcol
        ,MIG_ORA_EXT.REGEXP_REPLACE (
            testcol
            ,'([[:digit:]]{3})\.([[:digit:]]{3})\. ([[:digit:]]{4})'
            ,'(\1) \2-\3'
        ) RESULT
    FROM
        test
    WHERE
        LENGTH( testcol ) = 12
;
```

- **输入：REGEXP_REPLACE**

```
SELECT
        UPPER( regexp_replace ( 'foobarbequebazilbarfbonk
barbeque' ,'(b[^b]+)(b[^b]+)' ) )
    FROM
        DUAL
;
```

输出

```
SELECT
        UPPER( MIG_ORA_EXT.REGEXP_REPLACE ( 'foobarbequebazilbarfbonk
```

```
barbeque' ,'(b[^b]+)(b[^b]+)' ) )
    FROM
        DUAL
;
```

- **输入：REGEXP_REPLACE，使用 7 个参数（无效）**

```
SELECT
        REGEXP_REPLACE( 'TechOnTheNet' ,'a|e|i|o|u' ,'Z' ,1 ,1 ,'i' ,'(\1) \2-\3' )
AS First_Occurrence
    FROM
        emp
;
```

输出

输入表达式含有 7 个参数，但 MT 仅支持 REGEXP_REPLACE 含有 2 至 6 个参数，因此会记录错误“*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 操作

- **输入：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 ;
/
```

输出

```
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

设置以下参数后，可以迁移 LISTAGG/regexp_replace/regexp_instr：

- MigSupportForListAgg=false
- MigSupportForRegexReplace=false

输入： 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
```

输出

```
"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

本节主要介绍 Oracle PL/SQL 的迁移语法。迁移语法决定了关键字/功能的迁移方式。

PL/SQL 是 SQL 和编程语言过程特性的集合。

SQL 命令

Oracle 语法	迁移后语法
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*/

具体内容详见以下节点：

EDITIONABLE

GaussDB 不支持 EDITIONABLE 关键字，因此需要在目标数据库中删除。

输入：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);
```

输出

```
/*~~PACKAGE_SEND_MESSAGE~~*/
CREATE
    SCHEMA PACKAGE_SEND_MESSAGE
;
```

变量赋值

图6-43 输入：PL/SQL

```
BEGIN
 ...
 v_rowcount : = SQL%ROWCOUNT;
 ..
END;
```

图6-44 输出：PL/SQL

```
BEGIN
 ...
 v_rowcount := SQL%ROWCOUNT;
 ..
END;
```

END

不支持 END 指定标签。因此，迁移期间将删除标签名称。

输入：END，使用过程名

```
CREATE OR REPLACE PROCEDURE sp_ins_emp
  …
  …
  ...
END sp_ins_emp;
```

输出

```
CREATE OR REPLACE PROCEDURE sp_ins_emp
  …
  …
  ...
END;
```

输入：END，使用函数名

```
CREATE FUNCTION fn_get_bal
  …
  …
  ...
END get_bal;
/
```

输出

```
CREATE FUNCTION fn_get_bal
   …
   …
   ...
END;
/
```

EXCEPTION 处理

GaussDB(DWS)不支持 EXCEPTION 处理。要将脚本迁移到 V100R200C60，必须将 exceptionHandler 参数设置为 True。

对于 DSC 18.2.0，此参数必须设置为默认值 False。

图6-45 输入：EXCEPTION 处理

```
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 ;
/
```

图6-46 输出：EXCEPTION 处理

```

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 ;


    /
```

子事务处理

不支持子事务（即 PL/SQL 中的提交和回滚语句）。使用此参数的默认值 True。

图6-47 输入：子事务处理

```
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 ;
/
```

图6-48 输出：子事务处理

```
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

GaussDB T 不支持 Oracle PL/SQL 数据类型 STRING。使用 VARCHAR 来处理该数据类型。

图6-49 输入：STRING

```
--STRING
CREATE
    OR REPLACE FUNCTION text_length ( a CLOB ) RETURN String DETERMINISTIC IS BEGIN
            RETURN DBMS_LOB.GETLENGTH ( a ) ;
    END text_length ;
    /
```

图6-50 输出：STRING

```
/* STRING */
CREATE
    OR REPLACE FUNCTION text_length ( a CLOB ) RETURN VARCHAR DETERMINISTIC IS BEGIN
            RETURN DBMS_LOB.GETLENGTH ( a ) ;
    end ;
    /
```

LONG

数据类型 LONG 迁移为 TEXT。

输入：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;
 /
```

输出

```
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

当调用具有结果缓存的函数时，Oracle 执行该函数，将结果添加到结果缓存中，然后返回该函数。

当重复该函数调用时，Oracle 将从缓存中获取结果，而不必重新执行该函数。

某些场景下，这种缓存行为可带来显著的性能提升。

目标数据库不支持该关键字。该关键字会从目标文件中移除。

图6-51 输入：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;
/
```

图6-52 输出：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;
/
```

包含空格的关系运算符

GaussDB(DWS)不支持含有空格的关系运算符（<=、>=、!=）。DSC 会删除运算符之间的空格。

图6-53 输入：关系运算符

```
--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 ;
    /
```

图6-54 输出：关系运算符

```
/* 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 ;
     /
```

替换变量

替换变量是 Oracle SQL * Plus 工具的一个特性。 当在一个语句中使用一个替换变量时，SQL * Plus 会请求一个输入值并重写该语句以将其包含在内。 重写的语句被传递到 Oracle 数据库。 当输入的 Oracle 脚本包含任何替换变量时，DSC 将显示以下消息。消息记录在控制台和日志文件中。

```
********************************************************************************
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

在 Oracle 中，通过 PARALLEL_ENABLE 启用并发执行，从而实现负载分区。

输入：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;
/
```

输出

```
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 子句

PARALLEL 必须加注释。

输入

```
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;
```

输出

```
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

Oracle 中的 TRUNCATE TABLE 语句用于从表中删除所有记录，与 DELETE 语句功能相同，但不含 WHERE 子句。执行截断操作后，表将成为空表。DSC 仅可迁移含有静态表名称的 TRUNCATE TABLE 语句，不支持迁移含有动态表名称的 TRUNCATE TABLE 语句。

说明

该工具不支持迁移含有动态表名称的 TRUNCATE TABLE 语句。

例如：l_table :='truncate table ' || **itable_name**

在此示例中，**itable_name** 表示动态表名称，不受 DSC 支持。不支持的语句将被原样复制到已迁移的脚本中。

输入：TRUNCATE TABLE，使用 Execute Immediate

```
CREATE OR REPLACE PROCEDURE schema1.proc1
AS
BEGIN
   EXECUTE IMMEDIATE 'TRUNCATE TABLE QUERY_TABLE';
End proc1;
/
```

输出

```
CREATE
    OR REPLACE PROCEDURE schema1.proc1 AS BEGIN
        EXECUTE IMMEDIATE 'TRUNCATE TABLE schema1.QUERY_TABLE' ;
    end ;
/
```

输入：在过程中使用 TRUNCATE TABLE

说明

DSC 不会为动态 PL/SQL 语句添加模式名称。

```
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 ;
/
```

输出

```
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

Oracle 中的 ALTER SESSION 语句用于设置或修改数据库连接的参数和行为。该语句将持续有效，除非数据库连接断开。DSC 可迁移如下形式的 ALTER SESSION 语句：

- 含有 ADVISE、ENABLE、DISABLE、CLOSE 和 FORCE 的 ALTER SESSION 语句将被迁移为注释脚本。

- 含有 SET CLAUSE 参数（例如：NLS_DATE_FORMAT 和 NLS_DATE_LANGUAGE 等）的 ALTER SESSION 语句将被逐字复制。

说明

该工具不支持迁移命令子句含有变量的 ALTER SESSION 语句。

例如：EXECUTE IMMEDIATE ' alter session ' || ***command_val*** || 'parallel ' || type_value.

示例中，command_val 是变量，不受 DSC 支持。不支持的语句将被逐字复制到已迁移的脚本中。

输入：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';
```

输出

```
/*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';
```

输入：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;
/
```

输出

```
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

输入：AUTONOMOUS

```
CREATE OR REPLACE EDITIONABLE PACKAGE BODY "Pack1"."DEMO PROC" is
   PROCEDURE log(proc name IN VARCHAR2, info IN VARCHAR2) IS
   PRAGMA AUTONOMOUS_TRANSACTION;
```

输出

```
CREATE OR REPLACE PROCEDURE DEMO_PROC.log ( proc_name IN VARCHAR2 ,info IN
VARCHAR2 ) IS
/*PRAGMA AUTONOMOUS_TRANSACTION;*/
```

过程调用

调用同一个不包含参数的过程时，需要在过程名称后加上() 。

例如：pkg_etl.clear_temp_tables()

输入

```
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;
/
```

输出

```
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 ;
/
```

调用不包含参数的函数名

EXCEPTION 语句不支持有参数的函数名调用没有参数的函数名称，例如，SAD.SAD_CALC_ITEM_PKG_TEST_OB#error_msg ( )，但此函数 error_msg 没有定义参数，如下所示：

```
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 ;
```

脚本： SAD_CALC_ITEM_PKG_TEST_OB.SQL, SAD_CALC_ITEM_PRI_TEST_OB.SQL

输入：

```
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;
```

输出：

```
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.quotati
on_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 ' ||
SAD.SAD_CALC_ITEM_PKG_TEST_OB#func_name ( ) || ',' || SQLERRM ;
END ;
```

输入：

```
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;
/
```

输出：

```
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 集合（使用自定义类型）

本节主要介绍 Oracle PL/SQL 集合的迁移语法。迁移语法决定了关键字/功能的迁移方式。

自定义类型（UDT）衍生于数据库支持的数据类型。

自定义数据类型基于内置数据类型和其他自定义数据类型，定义应用程序中数据的结构和行为。自定义类型便于用户使用 PL/SQL 集合。

UDT 表

创建该类型的表，以跟踪用户定义类型的结构。表中不存储任何数据。

输入：CREATE TABLE TYPE

```
CREATE <OR REPLACE> TYPE <schema.>inst_no_type IS TABLE OF VARCHAR2 (32767);
```

输出

```
CREATE TABLE<schema.>mig_inst_no_type
      ( typ_col VARCHAR2 (32767) );
```

UDT VArray

输入：CREATE VArray

```
CREATE TYPE phone_list_typ_demo AS VARRAY(n) OF VARCHAR2(25);
```

输出

```
CREATE TABLE mig_pone_list_typ_demo
       ( typ_col VARCHAR2 (25) );
```

声明用户自定义类型

输入：声明用户自定义类型

```
DECLARE
      v_SQL_txt_array
       inst_no_type <:=
       inst_no_type()>;
BEGIN
      …
```

输出

```
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

输入：UDT，在 FOR LOOP 中使用 COUNT

```
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;
  ...
```

输出

```
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 记录

记录类型用于创建记录，并且可以在任何 PL/SQL 块、子程序或包的声明部分中定义。

输入：RECORD 类型

```
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;
/
```

输出

```
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 ;
/
```

增强用户自定义类型

DSC 支持在特定数据类型和任何表字段中增强 Oracle 中使用的 TABLE 的 PL/SQL 类型。

输入：特定数据类型的 TABLE 的 PL/SQL 类型

```
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
     …
```

输出

```
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';
            …
```

输入：任意表字段的 TABLE 的 PL/SQL 类型

```
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
…
```

输出

```
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

Gauss 支持 EXTEND 关键字。

输入： 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;
  /
```

输出

```
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

RECORD 类型在包规范中声明但是实际作用于包体。

设置以下参数后，用户自定义数据类型将迁移为 **VARRY**：

plSQLCollection=varray

输入：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;
/
```

输出

```
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;
/
```

TYPE 命名约定

用户定义的类型允许定义数据类型，以模拟应用程序中数据的结构和行为。

输入

```
CREATE
    TYPE t_line AS ( product_line VARCHAR2 ( 30 )
                           ,product_amount NUMBER ) ;
;
```

输出

```
CREATE
    TYPE sad_dml_product_pkg.t_line AS ( product_line VARCHAR2 ( 30 )
                                                                ,product_amount
NUMBER ) ;
```

输入

```
CREATE
    TYPE t_line AS ( product_line VARCHAR2 ( 30 )
                           ,product_amount NUMBER ) ;
```

输出

```
CREATE
    TYPE SAD.sad_dml_product_pkg#t_line AS ( product_line VARCHAR2 ( 30 )
                                                                  ,product_amount
NUMBER ) ;
```

说明

- 对于第一个输出的 pkg.t，如果配置文件中的“pkgSchemaNaming”设置为“true”，则 PL RECORD 迁移应将包名称作为模式名称以及类型名称。
- 对于第二个输出的 pkg #t，假设 TYPE 属于 sad_dml_product_pkg 包：

若配置文件中的“pkgSchemaNaming”设置为 false，PL RECORD 迁移应将模式名称作为模式名称以及包名称+类型名称，以＃分隔类型名称。

SUBTYPE

SUBTYPE 语句中，PL/SQL 允许您定义自己的子类型或预定义数据类型的别名，有时称为抽象数据类型。

输入

```
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;
 /
```

输出

```
 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 ;
/
```

说明

由于高斯不支持 SUBTYPE，因此使用 SUBTYPE 变量时，需要将其替换成创建 SUBTYPE 时使用的实际类型。

6.9.17 PL/SQL 包

本节主要介绍 Oracle PL/SQL 包（详情请参见 6.9.17.1 包）和 REF CURSOR（详情请参见 6.9.17.4 REF CURSOR）的迁移语法。迁移语法决定了关键字/功能的迁移方式。

本节包括以下内容：

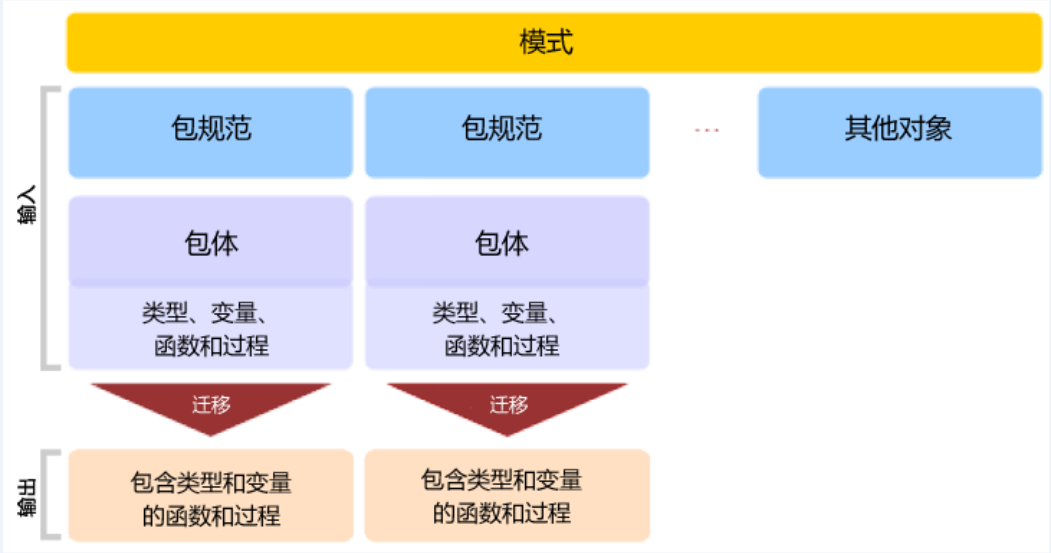
包、包变量、包拆分、REF CURSOR、VARRAY、创建包模式、授予执行权限、包名列表、数据类型，各节点的具体内容详见 6.9.17.1 包~6.9.21 数据类型章节。

6.9.17.1 包

包是对逻辑上相关的 PL/SQL 类型、变量、函数和过程进行分组形成的模式对象。在 Oracle 中，每个包由两部分组成：包规范和包体。包规范可能包含变量，以及在变量中声明的 REF CURSOR。包的 REF CURSOR 会被识别并迁移至引用位置。包体中的函数和过程将迁移到单独的函数和过程中。包体中的类型和变量会迁移到各个函数和过程中。

如果包规范和包体的模式名称不匹配，则 DSC 将在 schematoolError.log 文件中记录模式名称不匹配的错误。

图6-55 PL/SQL 包迁移



输入：PL/SQL 包（包规范和包体）

```
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;
/
```

输出（包含了输入包体中每个函数和过程各自的函数和过程）

```
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 ;
        /
```

输入：PL/SQL 包

```
CREATE OR replace VIEW vw_emp_name AS
        Select pkg_get_empdet.get_sal(emp.empno) as empsal from emp;
```

输出

```
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 ;
```

输入：无参数的函数/过程

以防过程或函数没有任何参数，调用相同的过程或函数时，需要在过程或函数名后添加()。

```
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;
/
```

输出

```
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 ;
/
```

输入：无过程或函数的包体

以防包体没有任何逻辑，如过程和函数，迁移工具需要从同一包中删除所有代码。通常情况下输出为空。

```
CREATE OR REPLACE PACKAGE BODY SAD.bas_subtype_pkg IS
BEGIN
  NULL;
END bas_subtype_pkg;
/
```

输入：SUBTYPE

通过 SUBTYPE 语句，PL/SQL 允许您定义自己的子类型或定义预置数据类型的别名，有时称为抽象数据类型。

```
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;
 /
```

输出

```
"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 ;
 /
```

说明

由于 GaussDB 不支持 SUBTYPE，因此使用 SUBTYPE 变量时，需要替换为 SUBTYPE 创建时使用的实际类型。

输入：sys.dbms_job

DBMS_JOB 调度和管理作业列队中的作业。

```
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;
```

输出

```
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;
```

说明

调用包时需要删除 SYS 模式。

输入：过程/函数变量

由于 GaussDB 的变量声明不支持 NULL 约束，因此需要注释 NULL 关键字。

```
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;
/
```

输出

```
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 ;/
```

输入：配置参数 addPackageNameList 为 true

提示按系统访问特定模式中的对象。

```
CREATE OR REPLACE PACKAGE BODY IC_STAGE.PKG_REVN_ARPU
AS
-----------
-----------
END PKG_REVN_ARPU;
/
```

输出

```
SET  package name list = 'PKG REVN ARPU' ;
--------------
--------------
reset package_name_list ;
```

输入：配置参数 addPackageNameList 为 false

提示按系统访问特定模式中的对象。

```
CREATE OR REPLACE PACKAGE BODY IC_STAGE.PKG_REVN_ARPU
AS
-----------
-----------
END PKG_REVN_ARPU;
/
```

输出

```
SET SEARCH_PATH=PKG_REVN_ARPU,PUBLIC;
```

输入：PACKAGE

提示过程和函数属于包。

```
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;
/
```

输出

```
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 ;
/
```

说明

在 IS/AS 语句之前创建任何过程和函数时需要输入 PACKAGE 关键字。

输入：嵌套过程

在过程中创建过程称为嵌套过程。嵌套过程是私有的，属于父过程。

```
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;
```

输出

```
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;
/
```

说明

当实现嵌套过程/函数时，所有过程/函数中的包变量都需要处理。

子过程/函数迁移后，需要迁移父过程/函数。

pkgSchemaNaming 为 false

如果 pkgSchemaNaming 设置为 false，则 PL RECORD 迁移不应将类型名称中的包名用作其模式。

输入

```
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;
/
```

输出

```
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 包变量

Oracle 支持包变量，允许变量保留包中所有的函数/过程。DSC 通过自定义函数实现 GaussDB(DWS)支持包变量。

说明

前提条件：

- 创建并使用 MIG_ORA_EXT 模式。
- 复制自定义脚本文件的内容，并在要执行迁移的所有目标数据库中执行此脚本。详情请参见 6.7.1 迁移流程。

如果模式和包名称之间存在空格，或包规范或包体（二者之一）含有引号，则输出可能与预期不符。

输入：包变量

```
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 ;
/
```

输出

```
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 ;
/
```

说明

如果 pkgSchemaNaming 设置为 true，

- Oracle 支持多个模式的包变量。如果不同的模式具有相同的包名和变量名，例如：

- schema1.mypackage.myvariable
- schema2.mypackage.myvariable

  则在迁移之后，模式名称将不会用于区分这两个包变量。由于模式名称被忽略，[any_schema] .mypackage.myvariable 的最后一个数据类型声明或操作将覆盖 schema1.mypackage.myvariable 和 schema2.mypackage.myvariable 的类型和值。

输入：使用 CONSTANT 关键字在一个包中声明的默认值的包变量，并在另一个包中使用

在包规范中声明的全局变量可以在相同的包和其他包中被访问。

```
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;
/
```

输出

```
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;
/
```

说明

包变量需要在 CURSOR 声明之前声明。

输入：EXCEPTION 的变量

包变量是一种全局变量，可以通过声明一次在整个包中使用。

```
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
/
```

输出

```
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 ;
/
```

说明

GaussDB 没有软件包功能，因此包变量需要使用它的过程或函数中声明。

输入：若 pkgSchemaNaming 设置为 false

包变量是一种全局变量，可以通过声明一次在整个包中使用。

```
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;
/
```

输出

```
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 ;
/
```

说明

如果配置参数 pkgSchemaNaming 设置为 false，则包变量迁移在某些地方会出错（例如，GET 获取默认值和 SET 分配最终值不会被添加）。但是，内核团队不建议使用此设置。请咨询内核团队。

输入：数据类型声明为包变量的表列%TYPE

如果数据类型被声明为变量的表列%TYPE，在表创建级别定义的数据类型被视为相应的列。

```
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
/
```

输出

```
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 ;
/
```

说明

使用数据类型作为表列%TYPE 迁移包变量时，需要从表中获取实际数据类型，并且在声明变量时使用该数据类型，而不是使用%TYPE。

输入：若配置参数 pkgSchemaNaming 设置为 false

如果一起指定 PACKAGE 名称和 SCHEMA 名称，则需要在 GET()上使用 SCHEMA 名称来获取默认值，并使用 SET()来指定最终值。

```
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;
/
```

输出

```
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 ;
/
```

输入：若配置参数 pkgSchemaNaming 设置为 false

若配置参数 pkgSchemaNaming 设置为 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;
/
```

输出

```
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 ;
/
```

输入： 若 pkgSchemaNaming 设置为 false，使用包变量

全局变量在包转换期间未正确转换，并在编译期间报错。如果配置参数 pkgSchemaNaming 设置为 false，则某些位置不会进行包变量迁移。但是，内核团队不建议使用此设置，详情请咨询内核团队。

```
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 ;
/
```

输出

```
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 ;
/
```

输入：(%type)表中的表字段类型定义

包转换期间，模式定义不会被添加到（%type）表中的表字段类型定义中，并且在编译期间会报错。

```
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 ;
/
```

输出

```
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

包变量可以被添加为 EXCEPTION，GaussDB 不支持此功能。

输入

```
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;
/
```

输出

```
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;

/
```

默认值

function 被指定为包变量的默认值。

输入

```
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;
/
```

输出

```
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

PLS_INTEGER 数据类型未转换为用于包变量的 INTEGER，但对其他本地变量起作用，因此，包变量 PLS_INTEGER 应转换为 INTEGER 数据类型，例如，varaible1 PLS_INTEGER ==> varaible1 INTEGER。

脚本：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

输入

```
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;
```

输出

```
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;
```

输入

```
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;
```

输出

```
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_NUMBER_OF_ENTITIES ) ;

      RETURN po_error_msg ;

END ;
/
```

带有包变量的游标

SAD.sad_calc_product_price_pkg#calc_product_price 中声明的游标包含包变量，并且需要被处理。

输入

```
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;
/
```

输出

```
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_HEA
DER_WAITING_SPLIT_STATUS ) ;
```

```
END ;
/
```

RETURN 后的 SET VARIABLE 函数

SET VARIABLE 函数应在过程和函数中的 RETURN 语句前被调用。

输入

```
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;
/
```

输出

```
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 ;
/
```

空包

无需迁移空包体。

输入

```
CREATE OR REPLACE PACKAGE BODY SAD.bas_subtype_pkg IS
BEGIN
  NULL;
END bas_subtype_pkg;
/
```

输出文件为空。

6.9.17.3 包拆分

包规范迁移为以包名命名的模式，包体中的存储过程和函数迁移为 *Packagename.procedurename* 和 *Packagename.funtionname*。

设置 **pkgSchemaNaming=true** 后，可以进行迁移。

输入： 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;
/
```

输出

```
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 ;
        /
```

若 pkgSchemaNaming 为 false，可拆分包。

当 bas_lookup_misc_pkg 调用 insert_fnd_data_change_logs 时，不会迁移 insert_fnd_data_change_logs。

输入

```
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;
/
```

输出

```
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 关键字

内核需要将包标签添加到从包转换来的函数和存储过程。

输入

```
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 ;
/
```

输出

```
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 是一种数据类型，它可保存数据库游标值，并可用于返回查询结果。 DSC 支持 REF CURSOR 的迁移。如下示例显示了 DSC 如何迁移 lref_strong_emptyp（本地 REF CURSOR）和 ref_strong_emptyp（包级别 REF CURSOR）。

输入：PL/SQL 程序包中使用 REF CURSOR（包规范和包体）

```
# 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;
/
```

输出

```
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( current_schema
( ) ) ,UPPER( 'pkg_refcur' ) ,'B' ,UPPER( 'var_num' ) ,UPPER( 'NUMBER' ) ,false ,NU
LL ,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 创建包模式

包声明迁移为创建以包名命名的模式。设置 **pkgSchemaNaming=false** 后，可以进行迁移。

输入：为包创建模式名

```
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
   );
```

输出

```
/*~~PACKAGE_GET_NOVA_INFO~~*/
CREATE
    SCHEMA PACKAGE_GET_NOVA_INFO
;
```

6.9.18 VARRAY

REF CURSOR 定义为返回参数。

设置 **plSQLCollection=varray** 后进行迁移。

输入： 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;
/
```

输出

```
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 授予执行权限

此功能授予用户特定包的特定权限。由于 GaussDB 不支持包，特定包中定义的所有过程和函数都将被授予执行权限。

输入

```
GRANT EXECUTE ON SAD.BAS_LOOKUP_MISC_PKG TO EIP_SAD;
```

输出

```
GRANT EXECUTE ON procedure_name TO EIP_SAD;
GRANT EXECUTE ON function1_name TO EIP_SAD;
```

说明

此处，procedure _name 和 function1_name 必须都属于 SAD.BAS_LOOKUP_MISC_PKG。

授予包的执行权限

包的最后一次授权不会被转换。

--GRANT

输入

```
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;
/
```

输出

```
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' )::V
ARCHAR2(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 包名列表

启用&禁用

设置 package_name_list 为 bas_lookup_misc_pkg。

根据配置参数启用和禁用参数。

输入

```
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
```

输出

```
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 数据类型

子类型

包中的自定义类型无法被转换。

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

输入：

```
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;
/
```

输出：

```
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

包的过程/函数包含 IN/OUT 参数中的%ROWTYPE 属性，此功能不被支持。

脚本：BAS_DML_SERVIECE_PKG.SQL, BAS_LOOKUP_MISC_PKG.SQL

输入：

```
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;
```

输出：

```
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;
```

输入

```
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;
/
```

输出

```
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 支持中文字符

输入：中文（

```
create table test11(a  int,b  int)/*create table test11(a  int,b  int)*/;
```

输出

```
create table test11(a int,b int)/*create table test11(a  int,b  int)*/;
```

输入：中文）

```
create table test11(a  int,b  int)/*create table test11(a  int,b  int)*/;
```

输出

```
create table test11(a  int,b  int)/*create table test11(a  int,b  int)*/;
```

输入：中文，

```
create table test11(a  int,b  int)/*create table test11(a  int,b  int)*/;
```

输出

```
create table test11(a int,b int)/*create table test11(a  int,b  int)*/;
```

输入：支持中文 SPACE

```
create table test11(a  int,b  int)/*create table test11(a  int,b  int)*/;
```

输出

```
create table test11(a int,b int)/*create table test11(a  int, b  int)*/;
```

6.10 Netezza 语法迁移

6.10.1 表（Netezza）

分布键

DISTRIBUTE ON (column)迁移为 DISTRIBUTE BY HASH (column)。

Netezza 语法	迁移后语法
CREATE TABLE N_AG_AMT_H	CREATE TABLE N_AG_AMT_H

Netezza 语法	迁移后语法
(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) ;	(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 需加注释。

Netezza 语法	迁移后语法
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)*/ ;

大字段类型

行存储支持 BLOB 和 CLOB。列存储不支持 BLOB，仅支持 CLOB。

Netezza 语法	迁移后语法
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（使用 RETURNS）

使用 RETURNS 的 PROCEDURE 迁移为使用 RETURNS 的 FUNCTION。

Netezza 语法	迁移后语法
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; --写日志，记录过程开始运行 CALL SP_LOG_EXEC(V_PRCNAME,V_PAR_DAY,D_TIME_START,0,0,'过程开始运行！',' '); V_STEP_INFO := '1.初始化'; BEGIN --1.1 SELECT COUNT(*) INTO V_CNT FROM information_schema.columns WHERE TABLE_NAME='TMPO_HXYW_LNSACCTINFO_H1'; if V_CNT>0 then	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; /* 写日志，记录过程开始运行 */ SP_LOG_EXEC(V_PRCNAME,V_PAR_DAY,D_TIME_START,0,0,'过程开始运行！',' '); V_STEP_INFO := '1.初始化'; 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

Netezza 语法	迁移后语法
EXECUTE IMMEDIATE 'DROP TABLE TMPO_HXYW_LNSACCTINFO_H1'; end if; END; RETURN O_RETURN; END; END_PROC;	TMPO_HXYW_LNSACCTINFO_H1'; end if; END; RETURN O_RETURN; END; /

修饰语言

nzplSQL 语言迁移为 plpgSQL 语言，或者直接删除。

Netezza 语法	迁移后语法
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; --写日志，记录过程开始运行 CALL SP_LOG_EXEC(V_PRCNAME,V_PAR_DAY,D_TIME_START,0,0,'过程开始运行！',' '); V_STEP_INFO := '1.初始化'; 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;	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; /* 写日志，记录过程开始运行 */ SP_LOG_EXEC(V_PRCNAME,V_PAR_DAY,D_TIME_START,0,0,'过程开始运行！',' '); V_STEP_INFO := '1.初始化'; 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;

Netezza 语法	迁移后语法
RETURN O_RETURN; END; END_PROC;	 END; /

进程编译规范

如果进程以 Begin_PROC 开始以 END_PROC 结束，则直接删除。

Netezza 语法	迁移后语法
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; --写日志，记录过程开始运行 CALL SP_LOG_EXEC(V_PRCNAME,V_PAR_DAY,D_TIME_START,0,0,'过程开始运行！',' '); V_STEP_INFO := '1.初始化'; 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; /* 写日志，记录过程开始运行 */ SP_LOG_EXEC(V_PRCNAME,V_PAR_DAY,D_TIME_START,0,0,'过程开始运行！',' '); V_STEP_INFO := '1.初始化'; 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; /

声明局部变量的关键字 DECLARE

DECLARE 应该修改为 AS。

Netezza 语法	迁移后语法
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; --写日志，记录过程开始运行 CALL SP_LOG_EXEC(V_PRCNAME,V_PAR_DAY,D_TIME_START,0,0,'过程开始运行！',' '); V_STEP_INFO := '1.初始化'; 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; /* 写日志，记录过程开始运行 */ SP_LOG_EXEC(V_PRCNAME,V_PAR_DAY,D_TIME_START,0,0,'过程开始运行！',' '); V_STEP_INFO := '1.初始化'; 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

变量类型

NVARCHAR 修改为 NCHAR VARING。

Netezza 语法	迁移后语法
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; --写日志，记录过程开始运行 CALL SP_LOG_EXEC(V_PRCNAME,V_PAR_DAY,D_TIME_ START,0,0,'过程开始运行！',' '); V_STEP_INFO := '1.初始化'; 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; /* 写日志，记录过程开始运行 */ SP_LOG_EXEC(V_PRCNAME,V_PAR_DAY,D_TIME_ START,0,0,'过程开始运行！',' '); V_STEP_INFO := '1.初始化'; RETURN O_RETURN; END; /

行计数

支持 row_count 行计数函数。

Netezza 语法	迁移后语法
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;	CREATE OR REPLACE FUNCTION "EDW"."SP_NTZ_ROWCOUNT" (CHARACTER VARYING(8)) RETURN INTEGER AS V_PAR_DAY ALIAS for $1; O_RETURN INTEGER; BEGIN

Netezza 语法	迁移后语法
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;	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; /

说明

ROW_COUNT 表示与前一条 SQL 语句关联的行数。如果前面的 SQL 语句是 DELETE、INSERT 或 UPDATE 语句，ROW_COUNT 表示符合操作条件的行数。

系统表

System tables _V_SYS_COLUMNS 替换为 information_schema.columns。

Netezza 语法	迁移后语法
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;

说明

列映射：

- table_schem => table_schema
- table_name => table_name
- column_name => column_name
- ordinal_position => ordinal_position
- type_name => data_type
- is_nullable => is_nullable

日期减法应返回相应整数

日期减法返回值应为整数。

Netezza 语法	迁移后语法
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)));

支持 TRANSLATE 函数

SQL TRANSLATE()函数用另一个字符序列替换字符串中的一组字符。该函数一次只能替换一个字符。

Netezza 语法	迁移后语法
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))

说明

如果包含一个参数，只需执行 UPPER。

UPPER(param1)

如果包含两个参数，抛出错误。

如果包含三个参数：

TRANSLATE(1st param, 3rd param, RPAD(2nd param, LENGTH(3rd param), ' '))

如果包含四个参数：

TRANSLATE(1st param, 3rd param, RPAD(2nd param, LENGTH(3rd param), 4th param))

数据类型

NATIONAL CHARACTER VARYING (ANY)

Netezza 语法	迁移后语法
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 语法	迁移后语法
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;	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; /

Netezza 语法	迁移后语法
END_PROC;	

Numeric (ANY)

Netezza 语法	迁移后语法
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; /

意外

TRANSACTION_ABORTED

Netezza 语法	迁移后语法
CREATE OR REPLACE PROCEDURE sp_ntz_transaction_aborted (NUMERIC(ANY) , NUMERIC(ANY)) RETURNS NATIONAL CHARACTER VARYING(ANY) LANGUAGE NZPLSQL AS BEGIN_PROC DECLARE	CREATE or replace FUNCTION sp_ntz_transaction_aborted (NUMERIC , NUMERIC) RETURN NATIONAL CHARACTER VARYING AS ERROR_INFO NCHAR VARYING(2000) := '';

Netezza 语法	迁移后语法
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;	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; /

指定 END 语句时不带分号

不带分号指定的 END 语句按如下方案迁移：

END /

Netezza 语法	迁移后语法
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) := 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijkl mnopqrstuvwxyz0123456789+/'; v_I int := 0;	CREATE or replace FUNCTION sp_ntz_end_wo_semicolon (NATIONAL CHARACTER VARYING(10)) RETURN CHARACTER VARYING AS v_B64 Varchar(64) := 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijkl mnopqrstuvwxyz0123456789+/'; v_I int := 0; v_J int := 0; v_K int := 0; v_N int := 0;

Netezza 语法	迁移后语法
v_J int := 0; v_K int := 0; v_N int := 0; v_out Numeric(38,0) := 0; I_LOAD_DT ALIAS FOR $1; BEGIN v_N:=Length(v_B64); FOR v_I In Reverse 1..Length(IN_base64) LOOP v_J:=Instr(v_B64,Substr(IN_base64,v_I,1))-1; If v_J <0 Then RETURN -1; End If; V_Out:=V_Out+v_J*(v_N**v_K); v_K:=v_K+1; END LOOP; RETURN V_Out; END END_PROC;	v_out Numeric(38,0) := 0; I_LOAD_DT ALIAS FOR $1; BEGIN v_N:=Length(v_B64); FOR v_I In Reverse 1..Length(IN_base64) LOOP v_J:=Instr(v_B64,Substr(IN_base64,v_I,1))-1; If v_J <0 Then RETURN -1; End If; V_Out:=V_Out+v_J*(v_N**v_K); v_K:=v_K+1; END LOOP; RETURN V_Out; END; /

LOOP

Netezza 语法	迁移后语法
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; /

高斯关键词

CURSOR

Netezza 语法	迁移后语法
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 语法	迁移后语法
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	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

Netezza 语法	迁移后语法
INSTR(VIEWSQL1,'v_p_etldate') > 0 LOOP VIEWSQL1 := SUBSTR(VIEWSQL1,1,INSTR(VIEWSQL1,'v_p_e tldate') - 1)||''''||I_LOAD_DT||''''||SUBSTR(VIEWS QL1,INSTR(VIEWSQL1,'v_p_etldate') + 11); END LOOP; EXECUTE IMMEDIATE VIEWSQL1; END LOOP; END; RETURN 0; END; END_PROC;	VIEWSQL1 := SUBSTR(VIEWSQL1,1,INSTR(VIEWSQL1,'v_p_e tldate') - 1)||''''||I_LOAD_DT||''''||SUBSTR(VIEWS QL1,INSTR(VIEWSQL1,'v_p_etldate') + 11); END LOOP; EXECUTE IMMEDIATE VIEWSQL1; END LOOP; END; RETURN 0; END; /

EXECUTE AS CALLER

Netezza 语法	迁移后语法
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;	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; /

Netezza 语法	迁移后语法
END; END_PROC;	

表达式

将 SELECT 结果赋值为变量。

Netezza 语法	迁移后语法
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 系统函数（Netezza）

ISNULL()

Netezza 语法	迁移后语法
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	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

Netezza 语法	迁移后语法
FROM tab123;	FROM tab123;

NVL

第二个函数丢失。

Netezza 语法	迁移后语法
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

日期类型转换。

Netezza 语法	迁移后语法
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;

分析函数（analytic_function）

Netezza 语法	迁移后语法
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;

存储过程

Netezza 语法	迁移后语法
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 算子

**

Netezza 语法	迁移后语法
V_Out := V_Out + v_J * (v_N ** v_K) ;	V_Out := V_Out + v_J * (v_N ^ v_K) ;

NOTNULL and ISNULL

Netezza 语法	迁移后语法
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（Netezza）

高斯关键字：SOURCE 指定为无 AS 关键字的列别名

Netezza 语法	迁移后语法
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 语法	迁移后语法
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;

说明

新增配置参数 keywords_addressed_using_doublequote，其值为：

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 (应指定 AS)

Netezza 语法	迁移后语法
SELECT username owner FROM tb_ntz_keyword_owner;	SELECT username AS owner FROM tb_ntz_keyword_owner;

ATTRIBUTE (应指定 AS)

Netezza 语法	迁移后语法
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 语法	迁移后语法
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,	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 */,

Netezza 语法	迁移后语法
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);	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 ADD constraint uq_prod UNIQUE (prod_no); */

说明

仅适用于 COLUMN store。对于 ROW 存储，不应注释“唯一索引”。

6.11 MySQL 语法迁移

本节列出了语法迁移工具支持的 MySQL 语法特性，并针对每一特性提供了 MySQL 语法及相应的 GaussDB(DWS)语法。通过本节所列语法可以了解 MySQL 脚本的内部迁移逻辑

本节还可以作为数据库迁移团队的参考，作为客户现场验证 MySQL 脚本迁移的参考。

6.11.1 基本数据类型

概述

MySQL 支持多种基本数据类型，大致可以分为以下几类：数值、日期/时间、字符串(字符)、大对象、集合、二进制和布尔类型。GaussDB(DWS)或不支持部分 MySQL 基本数据类型，或不支持部分 MySQL 数据类型精度设定，或不支持"UNSIGNED"、"ZEROFILL"等关键字。DSC 工具会根据 GaussDB 的支持情况做相应迁移。

数据类型是数据的一个基本属性，用于区分不同类型的数据。不同的数据类型所占的存储空间不同，能够进行的操作也不相同。数据库中的数据存储在数据表中。数据表中的每一列都定义了数据类型，用户存储数据时，须遵从这些数据类型的属性，否则可能会出错。各数据类型的基本概述详见表 6-20。

表6-20 数据类型基本概述

数据类型	基本描述
数字类型	内容详见数字类型对照。
日期和时间类型	文档中介绍如下日期和时间类型：DATETIME、TIME、TIMESTAMP、YEAR。GaussDB(DWS)不支持以上类型，DSC 工具将会对其转换。内容详见时间和日期类型对照。
字符串类型	MySQL 以字符单位解释字符列定义中的长度规范。这适用于 CHAR、VARCHAR 和 TEXT 类型。内容详见字符串类型对照。
空间数据类型	MySQL 具有对应于 OpenGIS 类的空间数据类型。内容详见空间数据类型对照。
大对象类型	BLOB 是一个二进制大对象，可以容纳可变数量的数据。这四个 BLOB 类型是 TINYBLOB，BLOB， MEDIUMBLOB，和 LONGBLOB。这些不同之处仅在于它们可以容纳的值的最大长度。内容详见大对象类型对照。
集合类型	1. MySQL ENUM 是一个字符串对象，具有从列创建时在列规范中明确枚举的允许值列表中选择的值。 2. SET 是一个字符串对象，可以有零个或多个值，每个值必须从创建

数据类型	基本描述
	表时指定的允许值列表中选择。 内容详见集合类型对照。
布尔类型	MySQL 支持两种布尔写法：BOOL、BOOLEAN。内容详见布尔类型对照。
二进制类型	1. MySQL BIT 数据类型被用于存储比特值。一种类型 允许存储位值。 可以从 1 到 64。 2. MySQL BINARY 和 VARBINARY 类似 CHAR 并且 VARCHAR，只不过它们包含二进制字符串。 内容详见二进制类型对照。

数字类型对照

表6-21 数字类型对照表

MySQL 数字类型	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)]
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]	INTEGER

MySQL 数字类型	MySQL INPUT	GaussDB(DWS) OUTPUT
	[ZEROFILL]	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

输入示例：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
);
```

输出示例

```
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");
```

时间和日期类型对照

表6-22 日期和时间类型对照表

MySQL 日期时间类型	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)

说明

该 *fsp* 值如果给出，则必须在 0 到 6 的范围内。值为 0 表示没有小数部分。如果省略，则默认精度为 0。

输入示例：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'
);
```

输出示例

```
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");
```

输入示例：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)
);
```

输出示例

```
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");
```

输入示例：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'
);
```

输出示例

```
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");
```

输入示例：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
);
```

输出示例

```
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");
```

字符串类型对照

表6-23 字符串类型对照表

MySQL 字符串类型	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)]

输入示例：CHAR

MySQL 一个长度 CHAR 列被固定在创建表声明的长度。长度可以是从 0 到 255 之间的任何值。CHAR 存储值时，它们将空格填充到指定的长度。

```
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
);
```

输出示例

```
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");
```

输入示例：[LONG|MEDIUM|TINY]TEXT

```
CREATE TABLE IF NOT EXISTS `runoob_dataType_test`(
    `dataType_1` LONGTEXT,
    `dataType_2` MEDIUMTEXT,
    `dataType_3` TEXT,
    `dataType_4` TINYTEXT
);
```

输出示例

```
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");
```

输入示例：VARCHAR

MySQL VARCHAR 列中的 值是可变长度的字符串。长度可以指定为 0 到 65,535 之间的值。

```
CREATE TABLE IF NOT EXISTS `runoob_dataType_test`(
    `dataType_1` VARCHAR(0),
    `dataType_2` VARCHAR(1845)
);
```

输出示例

```
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");
```

空间数据类型对照

表6-24 空间数据类型对照表

MySQL 空间数据类型	MySQL INPUT	GaussDB(DWS) OUTPUT
GEOMETRY	GEOMETRY	CIRCLE
POINT	POINT	POINT

MySQL 空间数据类型	MySQL INPUT	GaussDB(DWS) OUTPUT
LINESTRING	LINESTRING	POLYGON
POLYGON	POLYGON	POLYGON
MULTIPOINT	MULTIPOINT	BOX
MULTILINESTRING	MULTILINESTRING	BOX
MULTIPOLYGON	MULTIPOLYGON	POLYGON
GEOMETRYCOLLECTION	GEOMETRYCOLLECTION	CIRCLE

- GEOMETRY 可以存储任何类型的几何值。其他单值类型（POINT，LINESTRING 和 POLYGON）将其值限制为特定的几何类型。
- GEOMETRYCOLLECTION 可以存储任何类型的对象的集合。其他集合类型（MULTIPOINT， MULTILINESTRING， MULTIPOLYGON，和 GEOMETRYCOLLECTION）限制集合成员向那些具有特定的几何形状的类型。

输入示例

```
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;
```

输出示例

```
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");
```

大对象类型对照

表6-25 大对象类型对照表

MySQL 大对象类型	MySQL INPUT	GaussDB(DWS) OUTPUT
TINYBLOB	TINYBLOB	BLOB
BLOB	BLOB	BLOB
MEDIUMBLOB	MEDIUMBLOB	BLOB
LONGBLOB	LONGBLOB	BLOB

输入示例：[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
);
```

输出示例

```
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");
```

集合类型对照

表6-26 集合类型对照表

MySQL 集合类型	MySQL INPUT	GaussDB(DWS) OUTPUT
ENUM	ENUM	VARCHAR(14)
SET	SET	VARCHAR(14)

输入示例：ENUM

```
CREATE TABLE IF NOT EXISTS `runoob_dataType_test`(
    id   int(2) PRIMARY KEY,
```

```
    `dataType_17` ENUM('dws-1', 'dws-2', 'dws-3')
);
```

输出示例

```
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");
```

输入示例：SET

```
CREATE TABLE IF NOT EXISTS `runoob tbl test`(
    `dataType 18` SET('dws-1', 'dws-2', 'dws-3')
);
```

输出示例

```
CREATE TABLE "public"."runoob_tbl_test"
(
  "datatype_18" VARCHAR(14)
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_18");
```

布尔类型对照

输入示例：BOOL/BOOLEAN

```
CREATE TABLE IF NOT EXISTS `runoob_dataType_test`(
    `dataType_1` INT,
    `dataType_2` BOOL,
    `dataType_3` BOOLEAN
);
```

输出示例

```
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");
```

二进制类型对照

表6-27 二进制类型对照表

MySQL 二进制类型	MySQL INPUT	GaussDB(DWS) OUTPUT
BIT[(M)]	BIT[(M)]	BIT[(M)]
BINARY[(M)]	BINARY[(M)]	BYTEA
VARBINARY[(M)]	VARBINARY[(M)]	BYTEA

输入示例：BIT

```
CREATE TABLE IF NOT EXISTS `runoob_dataType_test`(
    `dataType_1` INT,
    `dataType_2` BIT(1),
    `dataType_3` BIT(64)
);
```

输出示例

```
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");
```

输入示例：[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)
);
```

输出示例

```
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 表（可选参数）

本节主要介绍表（可选参数）的迁移语法。迁移语法决定了关键字/功能的迁移方式。GaussDB(DWS)不支持表（可选参数），目前针对表（可选参数）的迁移方法都是临时迁移方法。

AUTO_INCREMENT

在数据库应用中，我们经常需要用到自动递增的唯一编号来标识记录。在 MySQL 中，可通过数据列的 auto_increment 属性来自动生成。可在建表时可用“auto_increment=n”选项来指定一个自增的初始值。可用“alter table table_name auto_increment=n”命令来重设自增的起始值。GaussDB(DWS)不支持该参数，DSC 迁移时会将设置该属性的字段迁移为 SERIAL 类型，并删除该关键字。

输入示例

```
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;
```

输出示例

```
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");
```

此外，GaussDB(DWS)也不支持基于 AUTO_INCREMENT 属性修改表定义信息。DSC 迁移时会将其移除。

输入示例

```
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;
```

输出示例

```
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

输入示例

```
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;
```

输出示例

```
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");
```

此外，GaussDB(DWS)不支持使用 “ **AUTO_INCREMENT**”属性修改表，DSC 在迁移后会直接删除该属性。

输入示例

```
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;
```

输出示例

```
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 指定表的默认字符集。GaussDB(DWS)不支持该属性修改表定义信息，DSC 迁移时会将该关键字删除。

输入示例

```
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;
```

输出示例

```
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

在 MySQL 中，CHECKSUM 表示对所有的行维护实时校验和。GaussDB(DWS)不支持该属性修改表定义信息，DSC 迁移时会将该关键字删除。

输入示例

```
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;
```

输出示例

```
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

在 MySQL 中，COLLATE 表示默认的数据库排序规则。GaussDB(DWS)不支持该属性修改表定义信息，DSC 迁移时会将该关键字删除。

输入示例

```
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;
```

输出示例

```
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

在 MySQL 中，COMMENT 对表进行注释。GaussDB(DWS)不支持该属性修改表定义信息，DSC 在迁移的过程中会将该属性删除。

输入示例

```
CREATE TABLE `public`.`runoob_alter_test`(
    `dataType1` int NOT NULL AUTO_INCREMENT,
    `dataType2` FLOAT(10,2),
    PRIMARY KEY(`dataType1`)
) comment='表的注释';

ALTER TABLE `public`.`runoob_alter_test` COMMENT '修改后的表的注释';
```

输出示例

```
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)不支持该属性修改表定义信息，DSC 迁移时会将该属性删除。

⚠ 注意

CONNECTION 关键字在 MySQL 中用作引用外部数据源。工具暂不支持该特性的完整迁移。基于当前的临时方案，工具仅仅移除该关键字。

输入示例

```
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';
```

输出示例

```
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 只对 MyISAM 引擎表有作用，根据 DELAY_KEY_WRITE 的值来延迟更新直至表关闭。GaussDB(DWS)不支持该属性修改表定义信息，DSC 迁移时会将该属性删除。

输入示例

```
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;
```

输出示例

```
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 表示允许在数据目录和索引目录之外创建表空间。DIRECTORY 包含 DATA DIRECTORY 和 INDEX DIRECTORY。GaussDB(DWS)不支持该属性修改表定义信息，DSC 迁移时会将该属性删除。

输入示例

```
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';
```

输出示例

```
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

在 MySQL 中，ENGINE 指定表的存储引擎。当存储引擎为 ARCHIVE、BLACKHOLE、CSV、FEDERATED、INNODB、MYISAM、MEMORY、MRG_MYISAM、NDB、NDBCLUSTER 和 PERFOMANCE_SCHEMA 时，DSC 支持该属性迁移，迁移过程中会将该属性删除。

输入示例

```
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;
```

输出示例

```
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 指定在表中插入行的位置，使用 FIRST 或 LAST 值将插入转到第一个或最后一个表，或使用值 NO 以防止插入。在迁移的过程中 DSC 会将该属性删除。

输入示例

```
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;
```

输出示例

```
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 的选择与存储引擎有关。对于 MyISAM 表，KEY_BLOCK_SIZE 可选地指定用于索引键块的字节大小。对于 InnoDB 表，KEY_BLOCK_SIZE 指定用于压缩的 InnoDB 表的页面大小（以 KB 为单位）。GaussDB(DWS)不支持该属性，DSC 迁移时会将属性删除。

输入示例

```
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;
```

输出示例

```
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

在 MySQL 中，MAX_ROWS 表示在表中存储的最大行数。DSC 迁移过程时会将该属性删除。

输入示例

```
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;
```

输出示例

```
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 表示在表中存储的最小行数。DSC 迁移过程时会将该属性删除。

输入示例

```
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;
```

输出示例

```
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

在 MySQL 中，PACK_KEYS 表示 MyISAM 存储引擎中的压缩索引。GaussDB(DWS)不支持该属性，DSC 迁移时会将该属性删除。

输入示例

```
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;
```

输出示例

```
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

在 MySQL 中，PASSWORD 表示用户密码。GaussDB(DWS)不支持该参数，DSC 迁移时会将该关键字删除。

输入示例

```
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';
```

输出示例

```
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 定义了行存储的物理形式。ROW_FORMAT 的选择与存储引擎有关，如果在创建表的时候选择了存储引擎不相关的 ROW_FORMAT，则使用默认的 ROW_FORMAT 创建表。当 ROW_FORMAT 取值为 DEFAULT，DSC 迁移为 SET NOCOMPRESS；当 ROW_FORMAT 取值为 COMPRESSED 时，DSC 迁移为 SET COMPRESS。GaussDB(DWS)不支持其他取值，当取其他值时 DSC 迁移时会将该属性删除。

输入示例

```
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;
```

输出示例

```
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 指定是否为 InnoDB 表自动重新计算持久性统计信息。GaussDB(DWS)不支持该属性，DSC 迁移时会将该关键字属性。

输入示例

```
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;
```

输出示例

```
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

在 MySQL 中，STATS_PERSISTENT 指定是否为 InnoDB 表启动持久性统计信息，通过 CREATE TABLE 或 ALTER TABLE 语句启动持久性统计信息。DSC 迁移时会将该属性删除。

输入示例

```
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;
```

输出示例

```
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 指定估计索引列的基数和其他统计信息时要采样的索引页数。DSC 迁移时会将该属性删除。

输入示例

```
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;
```

输出示例

```
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 是 MERGE 引擎的建表参数。通过该关键字建表类似于创建普通视图。新创建的表将在逻辑上合并 UNION 关键字限定的多个表的数据。DSC 迁移时会将该特性转为 GaussDB 视图创建语句。

输入示例

```
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;
```

输出示例

```
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 表（操作）

本节主要介绍表（操作）的迁移语法。迁移语法决定了关键字/功能的迁移方式。

本节包括以下节点内容：

- LIKE 表克隆
- 添加与删除列
- MODIFY 修改列
- CHANGE 修改列
- 设置与清除列默认值
- DROP 删除表
- TRUNCATE 删除表
- LOCK
- RENAME 重命名表名

LIKE 表克隆

MySQL 数据库中，可以使用 CREATE TABLE .. LIKE ..方式克隆旧表结构创建新表。GaussDB(DWS)也支持这种建表方式。DSC 工具迁移时会添加额外的表属性信息。

输入示例

```
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`);
```

输出示例

```
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);
```

添加与删除列

MySQL 添加、删除列语句与 GaussDB(DWS)存在差异。DSC 工具迁移时会根据 GaussDB 的特性进行相应适配。

⚠ 注意

GaussDB 不支持表定义中列序数的变更，工具暂不支持 FRIST，AFTER 特性的完整迁移。基于当前的临时方案，工具仅仅移除该关键字。

输入示例

```
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;
```

输出示例

```
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 修改列

MySQL 使用 MODIFY 关键字修改列数据类型、设置非空约束。DSC 工具迁移时会根据 GaussDB 的特性进行相应适配。

输入示例

```
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;
```

输出示例

```
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 修改列

MySQL 使用 CHANGE 关键字同时修改列名、列数据类型、设置非空约束。DSC 工具迁移时会根据 GaussDB 的特性进行相应适配。

输入示例

```
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;
```

输出示例

```
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);
```

设置与清除列默认值

MySQL 使用 ALTER 语句设置列默认值时可省略"COLUMN"关键字。DSC 工具迁移时会根据 GaussDB 的特性进行相应适配。

输入示例

```
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;
```

输出示例

```
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 删除表

GaussDB(DWS)与 MySQL 都支持使用 DROP 语句删除表，但 GaussDB(DWS)不支持在 DROP 语句中使用 RESTRICT | CASCADE 关键字。DSC 工具迁移时会将上述关键字移除。

输入示例

```
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;
```

输出示例

```
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 删除表

MySQL 在使用 TRUNCATE 语句删除表数据时可以省略 "TABLE"关键字，GaussDB 不支持这种用法。此外，DSC 工具在做迁移 TRUNCATE 语句时会添加 "CONTINUE IDENTITY RESTRICT"关键字。

输入示例

```
TRUNCATE TABLE `public`.`test_create_table01`;
TRUNCATE TEST_CREATE_TABLE01;
```

输出示例

```
TRUNCATE TABLE "public"."test_create_table01" CONTINUE IDENTITY RESTRICT;
TRUNCATE TABLE "public"."test_create_table01" CONTINUE IDENTITY RESTRICT;
```

LOCK

GaussDB(DWS)不支持 MySQL 中的"ALTER TABLE tbName LOCK"语句，DSC 工具迁移时会将其删除。

输入示例

```
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;
```

输出示例

```
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 重命名表名

MySQL 重命名表名的语句与 GaussDB(DWS)有一些差异。DSC 工具迁移时会根据 GaussDB 的特性进行相应适配。

⚠ 注意

工具暂不支持原表名附有 DATABASE(SCHEMA)的场景。

1. MySQL 通过 RENAME TABLE 语句修改表名

   输入示例

   ```
   # 单表重命名
   RENAME TABLE DEPARTMENT TO NEWDEPT;

   # 多表重命名
   RENAME TABLE NEWDEPT TO NEWDEPT_02,PEOPLE TO PEOPLE_02;
   ```

   输出示例

   ```
   --单表重命名
   ALTER TABLE "public"."department" RENAME TO "newdept";

   --多表重命名
   ALTER TABLE "public"."newdept" RENAME TO "newdept_02";
   ALTER TABLE "public"."people" RENAME TO "people_02";
   ```

2. MySQL 通过 ALTER TABLE RENAME 语句修改表名，DSC 工具迁移该语句时会将 "AS" 关键字迁移为 "TO"。

   输入示例

   ```
   ## A.
   ALTER TABLE runoob_alter_test RENAME TO runoob_alter_testnew;

   ## B.
   ALTER TABLE runoob_alter_testnew RENAME AS runoob_alter_testnewnew;
   ```

   输出示例

   ```
   -- 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 唯一索引

> **⚠ 注意**
>
> MySQL 唯一索引(约束)与主键约束联合使用的场景在工具迁移时会与 OLAP 场景下的分布键构成复杂的关系。工具暂不支持唯一索引(约束)与主键约束联合使用的场景。

1. 内联唯一索引，DSC 工具迁移时会将其移除

   输入示例

```
CREATE TABLE IF NOT EXISTS `public`.`runoob_dataType_test`
(
  `id` INT PRIMARY KEY AUTO_INCREMENT,
  `name` VARCHAR(128) NOT NULL,
  UNIQUE (id ASC)
);
```

   输出示例

```
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. ALTER TABLE 创建唯一索引，DSC 工具迁移时会根据 GaussDB 的特性创建普通索引

   输入示例

```
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);
```

输出示例

```
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. CREATE INDEX 创建唯一索引，DSC 工具迁移时会根据 GaussDB 的特性创建普通索引

输入示例

```
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);
```

输出示例

```
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 普通索引和前缀索引

GaussDB(DWS) 不支持前缀索引，也不支持内联普通索引。DSC 工具迁移时会根据 GaussDB 的特性将其迁移为普通索引。

1. 内联普通(前缀)索引

输入示例

```
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))
);
```

输出示例

```
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. ALTER TABLE 创建普通(前缀)索引

输入示例

```
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));
```

输出示例

```
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. CREATE INDEX 创建普通(前缀)索引

输入示例

```
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));
```

输出示例

```
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 索引

GaussDB(DWS) 不支持 HASH 索引。DSC 工具迁移时会根据 GaussDB 的特性将其迁移为普通索引。

1. 内联 HASH 索引

   输入示例

```
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)
);
```

   输出示例

```
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. ALTER TABLE 创建 HASH 索引

   输入示例

```
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;
```

   输出示例

```
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. CREATE INDEX 创建 HASH 索引

输入示例

```
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;
```

输出示例

```
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 BTREE 索引

GaussDB(DWS) 支持 BTREE 索引，但 USING BTREE 关键字在语句中的位置与 MySQL 存在差异。DSC 工具迁移时会根据 GaussDB 的特性进行相应适配。

1. 内联 BTREE 索引

输入示例

```
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))
);
```

输出示例

```
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. ALTER TABLE 创建 BTREE 索引

输入示例

```
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;
```

输出示例

```
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. CREATE INDEX 创建 BTREE 索引

输入示例

```
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));
```

输出示例

```
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 空间索引

GaussDB(DWS) 不支持 SPATIAL 空间索引。DSC 工具迁移时会根据 GaussDB 的特性进行相应适配。

1. 内联 SPATIAL 空间索引

   输入示例

```
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)
);
```

输出示例

```
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. ALTER TABLE 创建 SPATIAL 空间索引

输入示例

```
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;
```

输出示例

```
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. CREATE INDEX 创建 SPATIAL 空间索引

输入示例

```
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);
```

输出示例

```
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 删除索引

MySQL 支持 DROP INDEX 和 ALTER TABLE DROP INDEX 两种删除索引的语句。DSC 工具迁移时会根据 GaussDB 的特性进行相应适配。

1. DROP INDEX

输入示例

```
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;
```

输出示例

```
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

输入示例

```
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;
```

输出示例

```
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 注释

MySQL 支持由 '#' 或 '--' 字符引起的单行注释，而 GaussDB(DWS)仅支持由双破折号 '--' 字符引起的单行注释。DSC 工具迁移时会将 '#' 转化为 '--' 注释。

输入示例

```
## 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;
```

输出示例

```
-- 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 数据库

在 MySQL 中，DATABASE 是一种模式对象，等同于 Oracle、GaussDB(DWS)数据库的 SCHEMA 概念。DSC 工具迁移时考虑了以下两个场景。

1. 创建数据库

   输入示例

```
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;
```

输出示例

```
CREATE SCHEMA "dbname1";
CREATE SCHEMA "dbname2";

DROP SCHEMA IF EXISTS "dbname1";
DROP SCHEMA IF EXISTS "dbname2";
```

2. 使用数据库

输入示例

```
drop database if exists test;
create database if not exists test;
use test;
```

输出示例

```
DROP SCHEMA IF EXISTS "test";
CREATE SCHEMA "test";
SET CURRENT_SCHEMA = "test";
```

6.11.12 数据操作语句(DML)

本节主要介绍 MySQL DML 的迁移语法。迁移语法决定了关键字/功能的迁移方式。

本节包括以下内容：

- INSERT
- UPDATE
- REPLACE

INSERT

INSERT 插入形式包括：HIGH_PRIORITY、LOW_PRIORITY、PARTITION、DELAYED、IGNORE、多值插入以及 ON DUPLICATE KEY UPDATE。GaussDB(DWS)不支持以上类型插入，DSC 工具将会对其转换。

1. **HIGH_PRIORITY**

   MySQL 中如果指定 HIGH_PRIORITY，则会覆盖 LOW_PRIORITY 选项的效果。

   输入示例

```
# HIGH_PRIORITY 高优先级
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);
```

输出示例

```
-- HIGH_PRIORITY 高优先级
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**

   MySQL INSERT 插入语句使用 LOW_PRIORITY 修饰符时，则执行该 INSERT 延迟。

   输入示例

```
# LOW_PRIORITY 低优先级
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);
```

   输出示例

```
-- LOW_PRIORITY 低优先级
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**

   当插入到分区表中时，可以控制哪些分区和子分区接受新行。

   输入示例

```
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);
```

   输出示例

```
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**

须知

在 MySQL 5.7 中，DELAYED 关键字被接受，但被服务器忽略。

输入示例

```
# DELAYED 延迟
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);
```

输出示例

```
-- DELAYED 延迟
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**

MySQL INSERT 语句如果使用 IGNORE 修饰符，则执行 INSERT 语句时发生的错误将被忽略。

输入示例

```
# 如果表中已经存在相同的记录,则忽略当前新数据
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);
```

输出示例

```
-- 如果表中已经存在相同的记录,则忽略当前新数据
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（单语句多行值插入）**

   INSERT 使用 VALUES 语法的语句可以插入多行，以逗号分隔。

   输入示例

```
INSERT INTO  exmp_tb1 (tb1_name,tb1_sex,tb1_address,tb1_number)
VALUES('David','male','NewYork','01015827875'),('Rachel','female','NewYork','01
015827749'),('Monica','female','NewYork','010158996743');
```

   输出示例

```
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 使用 ON DUPLICATE KEY UPDATE 子句可以使现有行更新。

   输入示例

```
#ON DUPLICATE KEY UPDATE 若该数据的主键值/ UNIQUE KEY 已经在表中存在,则执行更新操作, 即
UPDATE;否则执行插入操作
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;
```

   输出示例

```
--ON DUPLICATE KEY UPDATE 若该数据的主键值/ UNIQUE KEY 已经在表中存在,则执行更新操作,
即 UPDATE;否则执行插入操作
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 语句的形式插入基于明确指定的值的行。

   输入示例

```
# INSERT INTO  SET 可以针对性的执行插入操作，但是一次只能插入一行数据，不能批量添加数据
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';
```

   输出示例

```
-- INSERT INTO  SET 可以针对性的执行插入操作，但是一次只能插入一行数据，不能批量添加数据
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

MySQL 的 UPDATE 操作形式包括：LOW_PRIORITY、ORDER BY、LIMIT 、IGNORE。GaussDB(DWS)不支持以上类型更新，DSC 工具将会对其转换。

1. **LOW_PRIORITY**

   MySQL UPDATE 语句如果使用 LOW_PRIORITY 修饰符，则执行 UPDATE 延迟。

   输入示例

```
#测试 LOW_PRIORITY 语法点
UPDATE LOW_PRIORITY employees SET department_id=2;
```

   输出示例

```
--测试 LOW_PRIORITY 语法点
UPDATE "public"."employees" SET "department_id" = 2;
```

2. **ORDER_BY**

   如果一个 MySQL UPDATE 语句包含一个 ORDER BY 子句，则这些行将按照该子句指定的顺序更新。

   输入示例

```
# 测试 ORDER BY 语法点
UPDATE  employees SET department_id=department_id+1  ORDER BY id;
```

   输出示例

```
-- 测试 ORDER BY 语法点
UPDATE "public"."employees" SET "department_id" = department_id+1;
```

3. **LIMIT**

UPDATE LIMIT 语法可以用来限制的范围。一个子句是一个行匹配的限制。只要发现满足该子句的行， 语句就会停下来 ，不管它们是否真的发生了变化。

输入示例

```
#单独测试 LIMIT 语法点
UPDATE  employees SET department_id=department_id+1   LIMIT 3 ;
UPDATE  employees SET department_id=department_id+1   LIMIT 3 , 10 ;

#测试 LIMIT + OFFSET 语法点
UPDATE  employees SET department_id=department_id+1   LIMIT 3   OFFSET 2;

#测试 LIMIT + ORDER BY 语法点搭配使用
UPDATE  employees SET department_id=department_id+1 ORDER BY fname  LIMIT 3 ;

#测试 LIMIT + WHERE + ORDER BY 语法点搭配使用
UPDATE  employees SET department_id=department_id+1 WHERE id<5 ORDER BY  fname
LIMIT 3 ;

#测试 LIMIT + WHERE + ORDER BY + OFFSET 语法点搭配使用
UPDATE  employees SET department_id=department_id+1 WHERE id<5 ORDER BY  fname
LIMIT 3 OFFSET 2 ;
```

输出示例

```
--单独测试 LIMIT 语法点
UPDATE "public"."employees" SET "department_id" = department_id+1;
UPDATE "public"."employees" SET "department_id" = department_id+1;

--测试 LIMIT + OFFSET 语法点
UPDATE "public"."employees" SET "department_id" = department_id+1;

--测试 LIMIT + ORDER BY 语法点搭配使用
UPDATE "public"."employees" SET "department_id" = department_id+1;

--测试 LIMIT + WHERE + ORDER BY 语法点搭配使用
UPDATE "public"."employees" SET "department_id" = department_id+1 WHERE id<5;

--测试 LIMIT + WHERE + ORDER BY + OFFSET 语法点搭配使用
UPDATE "public"."employees" SET "department_id" = department_id+1 WHERE id<5;
```

4. **IGNORE**

MySQL UPDATE 语句如果使用 IGNORE 修饰符，即使更新期间发生错误，update 语句也不会中止。

输入示例

```
#测试 IGNORE 语法点
UPDATE IGNORE employees SET department_id=3;
```

输出示例

```
--测试 IGNORE 语法点
UPDATE "public"."employees" SET "department_id" = 3;
```

REPLACE

MySQL 的 REPLACE 操作形式包括：LOW_PRIORITY、PARTITION 、DELAYED 、VALUES、SET；**（下述迁移示例为临时迁移方案）**

说明

REPLACE 的工作方式与 INSERT 完全相同，不同之处在于，如果表中的旧行与主键或唯一索引的新行具有相同的值，则在插入新行之前删除该旧行。

1. **LOW_PRIORITY**

   MySQL REPLACE 支持使用 LOW_PRIORITY ，DSC 工具将对其进行转换。

   输入

```
# LOW_PRIORITY 低优先级
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);
```

   输出

```
-- LOW_PRIORITY 低优先级
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 支持使用 PARTITION 关键字和分区，子分区或两者的逗号分隔名称列表显式分区选择。

   输入

```
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);
```

   输出

```
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**

⚠ 警告

DELAYED 插入和替换在 MySQL 5.6 中被弃用。在 MySQL 5.7 中，DELAYED 不支持。服务器识别但忽略 DELAYED 关键字，将替换处理为非延迟替换，并生成 ER_WARN_LEGACY_SYNTAX_CONVERTED 警告。（“ REPLACE DELAYED 不再被支持，语句被转换为 REPLACE。 ”）DELAYED 关键字将在未来版本中被删除。

输入

```
#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);
```

输出

```
--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 支持一条语句插入或删除多值，以逗号分隔。

输入

```
#有数据的话则替换 replace，没有的话则插入新的数据同 insert
Replace INTO  exmp_tb1 (tb1_id,tb1_name,tb1_sex,tb1_address,tb1_number)
VALUES(17,'David','male','NewYork11','01015827875'),(18,'Rachel','female','NewY
ork22','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','New
York22','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','fem
ale',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');
```

输出

```
--有数据的话则替换 replace，没有的话则插入新的数据同 insert
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 支持使用 SET 设置值，DSC 工具将对其转换。

   输入

```
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 ;
```

输出

```
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 事务管理与数据库管理

本节介绍如何迁移 MySQL 事务及数据库管理方面的关键字和功能。

事务管理

1. **TRANSACTION**

   DSC 工具在迁移 MySQL 事务处理语句时会根据 GaussDB 特性进行相应适配。

   输入示例

```
##该声明仅适用于会话中执行的下一个单个事务
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;
##使用 SESSION 关键字,适用于当前会话中执行的所有后续事务
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 ;
```

   输出示例

```
--该声明仅适用于会话中执行的下一个单个事务
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;
--使用 SESSION 关键字,适用于当前会话中执行的所有后续事务
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 工具在迁移 MySQL 事务处理锁表语句时会根据 GaussDB 特性进行相应适配。

   输入示例

```
## 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;
```

   输出示例

```
-- 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;
```

数据库管理

1. **SET CHARACTER**

   DSC 工具迁移时会将 MySQL "SET CHARACTER SET"语句迁移为"SET SESSION NAMES"。字符集对照如下表。

表6-28

MySQL CHARACTER SET	GaussDB SESSION NAMES
ASCII	SQL_ASCII
BIG5	BIG5

MySQL CHARACTER SET	GaussDB SESSION NAMES
CP1250	WIN1250
CP1251	WIN1251
CP1256	WIN1256
CP1257	WIN1257
CP932	SJIS
EUCJPMS	EUC_JP
EUCKR	EUC_KR
GB2312	GB18030
GBK	GBK
GREEK	ISO_8859_7
HEBREW	ISO_8859_8
KOI8R	KOI8R
KOI8U	KOI8U
LATIN1	LATIN1
LATIN2	LATIN2
LATIN5	LATIN5
LATIN7	LATIN7
SJIS	SJIS
SWE7	UTF8
TIS620	WIN874
UTF8	UTF8
UTF8MB4	UTF8

输入示例

```
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 中不支持 SET CHARACTER SET 'UCS2';
##mysql 中不支持 SET CHARACTER SET 'UTF16';
##mysql 中不支持 SET CHARACTER SET 'UTF16LE';
##mysql 中不支持 SET CHARACTER SET 'UTF32';
```

输出示例

```
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 中不支持 SET CHARACTER SET 'UCS2';
--mysql 中不支持 SET CHARACTER SET 'UTF16';
--mysql 中不支持 SET CHARACTER SET 'UTF16LE';
--mysql 中不支持 SET CHARACTER SET 'UTF32';
```

6.12 DB2 语法迁移

6.12.1 表（DB2）

Gauss 关键词

关键字作为列名时，必须添加英文引号，例如："order"。

DB2 语法	迁移后语法
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);

数据类型 （一）

“LONG VARCHAR”应修改为“CLOB”。

DB2 语法	迁移后语法
CREATE TABLE tbl_db2 (ID VARCHAR(36), NAME LONG VARCHAR);	CREATE TABLE tbl_db2 (ID VARCHAR(36), NAME CLOB);

LONG VARGRAPHIC。

DB2 语法	迁移后语法
CREATE TABLE tbl_db2 (ID VARCHAR(36), NAME LONG VARGRAPHIC);	CREATE TABLE tbl_db2 (ID VARCHAR(36), NAME CLOB);

外键

对以下外键约束属性进行注释：

- ON UPDATE RESTRICT
- ENFORCED
- ENABLE QUERY OPTIMIZATION

DB2 语法	迁移后语法
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*/;

序列

内置自动增量函数。

DB2 语法	迁移后语法
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));

表空间

即将放置表的表空间由 IN 子句指定。

DB2 语法	迁移后语法
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 ;

缺省值

给“WITH DEFAULT”指定给一个具体的缺省值。

DB2 语法	迁移后语法
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');

没有指定值的缺省值。

DB2 语法	迁移后语法
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));

数据类型 （二）

CLOB(1048576)

DB2 语法	迁移后语法
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 语法	迁移后语法
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 选项

LOGGED/UNLOGGED

DB2 语法	迁移后语法
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 语法	迁移后语法
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 子句

Organize By 子句

DB2 语法	迁移后语法
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（DB2）

SELECT 语句

FETCH 子句

DB2 语法	迁移后语法

DB2 语法	迁移后语法
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;

说明

fetch-first 子句设置了可检索的最大行数。

WITH AS

WITH AS 带列列表

DB2 语法	迁移后语法
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 子句用于显示指定 AS 的属性值。

DB2 语法	迁移后语法
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;

表空间

TABLE 函数由子查询指定。

DB2 语法	迁移后语法
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 索引（DB2）

反向扫描

反向扫描

DB2 语法	迁移后语法
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*/;

模式

索引模式（schema）不同于表模式。

DB2 语法	迁移后语法
CREATE INDEX IDXSC.IDX_1 ON EMP (ID ASC);	CREATE INDEX IDX_1 ON EMP (ID ASC);

6.12.4 NICKNAME

NICKNAME

在 DB2 中，同义词称为“Nickname”。

DB2 语法	迁移后语法
CREATE NICKNAME sc2.emp FOR sc.emp;	CREATE SYNONYM sc2.emp FOR sc.emp;

6.12.5 语句

CURRENT SCHEMA 语句

SET CURRENT SCHEMA 语句

DB2 语法	迁移后语法
CREATE NICKNAME sc2.emp FOR sc.emp;	CREATE SYNONYM sc2.emp FOR sc.emp;

含 ROW_COUNT 的 GET DIAGNOSTICS 语句

DB2 语法	转换后语法
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 系统功能

DAYS

系统功能：DAYS。

DB2 语法	迁移后语法
`SELECT DAYS(doj) FROM emp;`	`SELECT (TRUNC(doj - TO_DATE('0001/01/01', 'YYYY/MM/DD'))+1) FROM emp;`

MONTH

Month

DB2 语法	迁移后语法
`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 语法	迁移后语法
`SELECT YEAR(ORDER_DATE) as TEMP_HY;`	`SELECT EXTRACT (YEAR FROM ORDER DATE) as TEMP_HY;`

当前日期

当前日期

DB2 语法	迁移后语法
`SELECT CURRENT DATE FROM DUAL;`	`SELECT CURRENT_DATE FROM DUAL;`

当前时间戳

当前时间戳

DB2 语法	迁移后语法
`SELECT CURRENT TIMESTAMP - 7 DAYS;`	`SELECT CURRENT_TIMESTAMP - 7 DAYS;`

POSSTR 函数

POSSTR 函数

DB2 语法	迁移后语法
SELECT POSSTR('THIS IS TEST','TEST') FROM DUAL;	SELECT INSTR('THIS IS TEST','TEST') FROM DUAL;

VALUE 函数

Value 函数

DB2 语法	迁移后语法
Select VALUE('abc','') from dual;	Select Coalesce('abc','') from dual;

date 函数

date 函数通过值返回日期。

DB2 语法	迁移后语法
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 命令行参考

6.13.1 数据库模式迁移

功能

runDSC.sh 和 runDSC.bat 分别用于将 Teradata、Oracle、Netezza、MySQL 和 DB2 的 schema 和 query 迁移到 GaussDB(DWS)上。

命令格式

Linux 操作系统：

```
./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 操作系统：

```
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>
```

参数说明

表6-29 参数列表

全称	缩写	数据类型	说明	范围	默认值	示例
--source-db	-S	字符串	源数据库。	• Oracle • Teradata • Netezza • MySQL • DB2	N/A	--source-db *Oracle*(or) -S *Oracle*

全称	缩写	数据类型	说明	范围	默认值	示例
--input-folder	-I	字符串	包含 Teradata/Oracle 脚本的输入文件夹。	不适用	不适用	--input-folder */home/testmigration/Documentation/input* (or) -I */home/testmigration/Documentation/input*
--output-folder	-O	字符串	保持迁移后脚本的输出文件夹。	不适用	不适用	--output-folder */home/testmigration/Documentation/output*(or)-O */home/testmigration/Documentation/output*
--application-lang	-A	字符串	用于迁移的应用程序语言解析器。 **SQL**：迁移 SQL 文件中的 SQL 模式/脚本。 **Perl**：迁移 Perl 文件中的 BTEQ/SQL_LANG 脚本。	• SQL • Perl	SQL	--application-lang Perl 或 -A *Perl*
--conversion-type	-M	字符串	迁移类型。用户需根据输入脚本指定该参数： **Bulk**：迁移 DML 和 DDL 脚本。 **BLogic**：迁移业务逻辑，如过程和函数。 BLogic 仅适用于 Oracle PL/SQL。	• Bulk • BLogic	Bulk	--conversion-type *bulk* 或 -M *bulk*

全称	缩写	数据类型	说明	范围	默认值	示例
--log-folder	-L	字符串	日志文件路径。	不适用	不适用	--log-folder */home/testmigration/Documentation*(or) -L */home/testmigration/Documentation*
--version-number	-VN	字符串	Oracle 必选参数	Oracle	不适用	--version-number 或 -V1R8_330
--target-db	-T	字符串	目标数据库	• gaussdbT • gaussdbA	gaussdbA	--target-db gaussdbT （或） *-T gaussdbT*

使用指南

必须指定源数据库、输入和输出文件夹路径。迁移类型和日志路径可选。

说明

如果未指定日志路径，DSC 会在 TOOL_HOME 路径下创建 log 文件夹，用于存储所有日志。

命令示例

```
./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
```

系统回显

```
********************* 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 ************************
```

说明

如果输入文件夹中没有 SQL 文件，则在控制台上会显示如下消息：

```
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.
```

环境搭建及恢复（数据库及数据库用户）

GaussDB(DWS) -数据库创建和 schema 建立

步骤 1 登录 Postgres 系统。

```
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
```

步骤 2 运行 Setup 目录下的所有文件。

----结束

命令：

```
sh runDSC.sh -S oracle -M blogic -I <input path>
sh runDSC.sh -S oracle -M bulk -I <input path>
```

配置详情

步骤 1 设置 GaussDBSQLExec=True。并更新 gaussdb.properties 文件。

步骤 2 创建新用户（T），新数据库(A)。添加所有 schema 模式。

----结束

迁移后验证

Database Schema Convertor 转换完含有 SQL 语句的源文件后，在目标 GaussDB 上执行转换后的文件，并生成文件执行成功和失败的明细报告。

Database Schema Convertor 完成迁移后，会调用迁移后验证脚本（通过配置项控制）。此验证脚本（配置详情见配置文件）会连接到目标 GaussDB 数据库并执行。

迁移后验证脚本会连接到目标 GaussDB 数据库（具体信息在配置文件中配置），并执行该脚本。

1. 配置 config 文件夹下的 application.properties

   在 GaussDB 中执行迁移脚本的取值范围：true/false，默认值：false。

将 executesqlingauss 设置为 true。

true: executesqlingauss 将在 GaussDB 上执行迁移脚本。

2. 配置 config 文件夹下的 gaussdb.properties

#目标数据库配置

```
#gauss database user with all privileges
 gaussdb-user=
 gaussdb-port=
 #Database name for GaussDBA
 gaussdb-name=
 #gaussdb ip
 gaussdb-ip=
```

gsql 和 zsql 客户端的依赖关系：

a. 由于在 GaussDB 上执行脚本时需依赖 gsql（GaussDB(DWS)），为保证 Database Schema Convertor 正常运行，需在安装了 GaussDB 实例或客户端（gsql）的节点上运行 Database Schema Convertor，且进行验证的用户具有执行 gsql 或 zsql 命令的权限。

b. 由于 Gauss 数据库实例/客户端只能安装在 Linux 操作系统中，因此只能用于 Linux 环境下的功能验证。

c. 在远程 GaussDB 实例上执行 gsql 命令，建议在 GaussDB 实例的如下配置文件中增加客户端系统 IP 或主机名。

```
/home/gsmig/database/coordinator
---pg_hba.conf
```

回显

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 命令

功能

Version 命令用于显示 DSC 版本号。

命令格式

Linux:

```
./runDSC.sh --version
```

Windows:

```
runDSC.bat --version
```

参数说明

无

使用指南

Linux:

```
./runDSC.sh --version
```

Windows:

```
runDSC.bat --version
```

系统回显

```
Version: DSC (Gauss Tools 8.0.1)
```

6.13.3 Help 命令

功能

help 命令用于提供 DSC 支持的命令相关的帮助信息。

命令格式

Linux 操作系统:

```
./runDSC.sh --help
```

Windows 操作系统:

```
runDSC.bat --help
```

参数说明

无

使用指南

无

命令示例

Linux 操作系统:

```
./runDSC.sh --help
```

Windows 操作系统:

```
runDSC.bat --help
```

系统回显

Linux 操作系统:

```
./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 操作系统:

```
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 日志参考

6.14.1 日志概述

日志文件是 DSC 所有操作和状态的存储库。支持以下日志文件：

- **SQL 迁移日志**
  a. *DSC.log*：SQL 迁移的所有活动。
  b. *DSCError.log*：SQL 迁移错误。
  c. *successRead.log*：SQL 迁移中对输入文件的成功读次数。
  d. *successWrite.log*：SQL 迁移中对输入文件的成功写次数。
- **Perl 迁移日志**
  a. *perlDSC.log*：Perl 迁移中所有的活动、预警和错误。

Apache Log4j 用于指定 DSC 记录日志的框架。用户可使用以下 Log4j 配置文件，也可以根据需要进行自定义：

- Teradata/Oracle/Netezza/DB2 : config/log4j2.xml
- MySQL : config/log4j2_mysql.xml

本章具体介绍工具支持的日志。

6.14.2 SQL 迁移日志

SQL DSC（DSC.jar）支持以下类型的日志记录：

- 活动日志
- 错误日志
- 成功读
- 成功写

说明

- 如果用户指定了日志路径，所有日志都会保存在该路径下。
- 如果未指定日志路径，DSC 会在 TOOL_HOME 路径下创建 log 文件夹，用于存储所有日志。
- 为控制磁盘空间用量，日志文件的大小上限为 10 MB。用户最多可拥有 10 个日志文件。
- 工具日志不记录敏感数据，如查询。

活动日志

DSC 将所有日志和错误信息保存到 DSC.log 文件中。该文件位于 log 文件夹中。DSC.log 文件包含执行迁移的用户、迁移的文件、时间戳等详细信息。活动日志的记录级别为 INFO。

DSC.log 的文件结构如下：

```
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.ja
r , 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.ja
r , 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
```

错误日志

DSC 仅将迁移过程中发生的错误记录到 DSCError.log 文件中。该文件位于 log 文件夹中。DSCError.log 文件包含这些错误的日期、时间，文件详细信息（如文件名），以及查询位置等信息。错误日志的记录级别为 ERROR。

DSCError.log 的文件结构如下：

```
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
```

成功读

在 DSC 读取文件之后，该文件将被记录日志以进行跟踪。在某些情况下，用户可通过这些日志获取文件执行状态的信息。该文件位于 log 文件夹中。日志文件包括日期、时间、文件名等详细信息。此日志文件的日志记录级别为 INFO。

successRead.log 的文件结构如下：

```
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.
```

成功写

DSC 读取、处理文件并将输出写入磁盘。这个过程被记录到成功写日志文件中。在某些情况下，用户可通过此文件了解哪些文件已处理成功。在重新运行的情况下，用户

可以跳过这些文件运行剩余的文件。该文件位于 log 文件夹中。日志文件包括日期、时间、文件名等详细信息。此日志文件的日志记录级别为 INFO。

successWrite.log 的文件结构如下：

```
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 迁移日志

Perl 迁移时，DSC 将所有日志信息写入 perlDSC.log 文件。

说明

DSC 通过调用 SQL 来迁移 Perl 文件中的 SQL 脚本，因此支持以下 6.14.2 SQL 迁移日志：

- 活动日志
- 错误日志
- 成功读
- 成功写

日志级别

可以使用 logging-level 参数来配置 perl 迁移日志的记录级别。

日志记录

DSC 将所有日志、告警和错误信息保存到 log 文件夹下的 perlDSC.log 文件中。日志文件包含执行迁移的用户、迁移的文件、时间戳等详细信息。

perlDSC.log 的文件结构如下：

```
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 DSC 故障处理

本章介绍使用 DSC 时可能遇到的问题，并提供故障处理步骤。

下表列举了常见故障的问题现象、原因、解决方案。

表6-30 错误消息参考

问题现象	原因及解决方案
Error occurred while formatting! Returning unformatted SQL: select count(* from table_temp;	**原因：**可能原因为输出文件中左右括号数量不一致。 **解决方案：**确保文件中所有左右括号匹配。
ERROR QueryConversionUtility:249 Query is not converted as it contains unsupported keyword: LAST	**原因：**输入的查询文件包含一个不支持的关键词。 **解决方案：**确保要迁移的脚本中不含有不支持的关键词。 有关详情，请参见 6.2 支持的关键词和特性。
Disk is almost full. Please clear the space	**原因：**磁盘空间不足。

问题现象	原因及解决方案
and re-run the tool.	**解决方案：**从磁盘中释放空间然后重试。
Please enter valid input parameters, Kindly refer the user manual to execute.	**原因：**可能原因为： 1. 未输入有效参数。 2. 缩写关键字为小写。 **解决方案：** 1. 迁移时提供全部必选参数。 2. 确保所有缩写关键字为大写。 有关详情，请参见 6.13.1 数据库模式迁移。
No SQL files found in input folder. Hence stopping migration.	**原因：**迁移过程中输入的文件夹中不存在有效 SQL 文件。 **解决方案：**确保要迁移的 SQL 文件存在输入的文件夹中。 有关详情，请参见 6.7.1 迁移流程。
Migration Application failed to start : Currently we are not supporting this Database : <*database-name*>	**原因：**源数据库参数中提到的数据库名称不正确。 **解决方案：**DSC 仅支持 Teradata 或 Oracle 作为源数据库参数的值。 有关详情，请参见 6.13.1 数据库模式迁移。
Output folder is not set. Please enter an output folder and refer the user manual for syntax.	**原因：**未指定输出文件夹路径。 **解决方案：**指定输出文件夹参数的有效路径。 有关详情，请参见 6.13.1 数据库模式迁移。
ascii “****” does not map to charset	**原因：**DSC 无法检测输入文件的编码格式，且系统区域设置的字符集与输入文件的字符集不匹配。于是，系统上报告警。 **解决方案：**将 encodingFormat 参数设为实际编码值，并再次执行。 示例： `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`

问题现象	原因及解决方案
	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)

表6-31 错误消息参考

问题现象	原因及解决方案
无法创建评估被强制到远程节点进行的索引。	**原因：**GaussDB 存在限制，分布键列必须是唯一索引列的超集。 **解决方案：**GaussDB 目前不支持。
urowid 类型不存在。	**原因：**创建表时使用了用户自定义类型。 **解决方案：**GaussDB 目前不支持。
在或接近“LOCAL”的位置有语法错误。	**原因：**GaussDB 不支持索引中的 LOCAL 关键字。

问题现象	原因及解决方案
	解决方案：需要创建本地索引。
在或接近“1”的位置有语法错误。	**原因：**GaussDB(DWS)不支持 index 中的附加参数。 **解决方案：**需要注释掉。
在或接近“=”的位置有语法错误。	**原因：**GaussDB(DWS)不支持约束中的“=”。 **解决方案：**GaussDB 目前不支持。

6.16 DSC 常见问题

本章介绍常见问题。

问题 1：在安装过程中，提示“ Root privileged users are not allowed to install the DSC for Linux. ”应如何处理？

答：拥有 root 权限的用户不得在 Linux 中安装和执行 DSC。建议使用没有 root 权限的用户来安装和操作 DSC。

问题 2：如何配置 DSC，以便 Teradata 支持 GaussDB 200 V100R002C60 版本？

答：执行以下步骤设置表变量值，以支持当前 GaussDB 200 V100R002C60 版本：

1. 打开 TOOL_HOME 路径下 config 文件夹中的 Teradata features-teradata.properties 文件。
2. 根据需要修改下列变量：
   - VOLATILE
   - PRIMARY INDEX

   例如：

```
VOLATILE=UNLOGGED | LOCAL TEMPORARY
PRIMARY INDEX=ONE | MANY
```

说明

VOLATILE 变量的默认值为 LOCAL TEMPORARY，PRIMARY INDEX 变量的默认值为 MANY。

6.17 安全管理

须知

请务必使用最新的补丁更新操作系统和相关软件（详情请参见 6.4 系统要求 (DSC)），以防漏洞和其他安全问题。

为确保安全性，DSC 会对其创建的文件和文件夹进行访问控制。要访问这些文件和文件夹，用户必须拥有所需权限。例如，用户需要权限 600/400 访问目标文件和日志文件，需要权限 700 访问目标文件夹和日志文件夹。此外，该工具不在日志中保存敏感数据，以确保数据安全。

--input-folder 中指定的文件或文件夹不得具有 GROUP 和/或 OTHERS 的写权限。 出于安全考虑，如果输入文件/文件夹具有写入权限，则该工具不会执行。

不得使用拥有 root 权限的用户在 Linux 中安装和执行 DSC。

DSC.jar 文件中提供的 umask 值是系统设置值，与文件权限相关。建议用户不要修改此值。修改此值将影响文件权限。

说明

DSC 是一个单机应用程序，无需与任何网络或数据库连接即可运行。它可以在与任何网络隔离的任何机器上运行。

7 服务端工具

7.1 gs_dump

背景信息

gs_dump 是 GaussDB(DWS)用于导出数据库相关信息的工具，用户可以自定义导出一个数据库或其中的对象（模式、表、视图等）。支持导出的数据库可以是默认数据库 postgres，也可以是自定义数据库。

gs_dump 工具在进行数据导出时，其他用户可以访问集群数据库（读或写）。

gs_dump 工具支持导出完整一致的数据。例如，T1 时刻启动 gs_dump 导出 A 数据库，那么导出数据结果将会是 T1 时刻 A 数据库的数据状态，T1 时刻之后对 A 数据库的修改不会被导出。

gs_dump 支持将数据库信息导出至纯文本格式的 SQL 脚本文件或其他归档文件中。

- 纯文本格式的 SQL 脚本文件：包含将数据库恢复为其保存时的状态所需的 SQL 语句。通过 gsql 运行该 SQL 脚本文件，可以恢复数据库。即使在其他主机和其他数据库产品上，只要对 SQL 脚本文件稍作修改，也可以用来重建数据库。
- 归档格式文件：包含将数据库恢复为其保存时的状态所需的数据，可以是 tar 格式、目录归档格式或自定义归档格式，详见表 7-1。该导出结果必须与 gs_restore 配合使用来恢复数据库，gs_restore 工具在导入时，系统允许用户选择需要导入的内容，甚至可以在导入之前对等待导入的内容进行排序。

主要功能

gs_dump 可以创建四种不同的导出文件格式，通过**[**-F 或者--format=**]**选项指定，具体如表 7-1 所示。

表7-1 导出文件格式

格式名称	-F 的参数值	说明	建议	对应导入工具
纯文本	p	纯文本脚本文件包含	小型数据库，	使用 gsql 工具恢复

格式名称	-F 的参数值	说明	建议	对应导入工具
格式		SQL 语句和命令。命令可以由 gsql 命令行终端程序执行，用于重新创建数据库对象并加载表数据。	一般推荐纯文本格式。	数据库对象前，可根据需要使用文本编辑器编辑纯文本导出文件。
自定义归档格式	c	一种二进制文件。支持从导出文件中恢复所有或所选数据库对象。	中型或大型数据库，推荐自定义归档格式。	使用 gs_restore 可以选择要从自定义归档导出文件中导入相应的数据库对象。
目录归档格式	d	该格式会创建一个目录，该目录包含两类文件，一类是目录文件，另一类是每个表和 blob 对象对应的数据文件。	-	
tar 归档格式	t	tar 归档文件支持从导出文件中恢复所有或所选数据库对象。tar 归档格式不支持压缩且对于单独表大小应小于 8GB。	-	

说明

可以使用 gs_dump 程序将文件压缩为纯文本或自定义归档导出文件，减少导出文件的大小。生成纯文本导出文件时，默认不压缩。生成自定义归档导出文件时，默认进行中等级别的压缩。gs_dump 程序无法压缩已归档导出文件。

注意事项

禁止修改导出的文件和内容，否则可能无法恢复成功。

为了保证数据一致性和完整性，gs_dump 会对需要转储的表设置共享锁。如果表在别的事务中设置了共享锁，gs_dump 会等待锁释放后锁定表。如果无法在指定时间内锁定某个表，转储会失败。用户可以通过指定--lock-wait-timeout 选项，自定义等待锁超时时间。

语法

```
gs_dump [OPTION]... [DBNAME]
```

说明

“dbname”前面不需要加短或长选项。“dbname”指定要连接的数据库。

例如:

不需要-d，直接指定“dbname”。

```
gs_dump -p port_number  postgres -f dump1.sql
```

或者

```
export PGDATABASE=postgres
 gs_dump -p port_number -f dump1.sql
```

环境变量： PGDATABASE

参数说明

通用参数：

- -f, --file=FILENAME

  将输出发送至指定文件或目录。如果省略该参数，则使用标准输出。如果输出格式为(-F c/-F d/-F t)时，必须指定-f 参数。如果-f 的参数值含有目录，要求目录对当前用户具有读写权限。

- -F, --format=c|d|t|p

  选择输出格式。格式如下：

  - p|plain：输出一个文本 SQL 脚本文件（默认）。
  - c|custom：输出一个自定义格式的归档，并且以目录形式输出，作为 gs_restore 输入信息。该格式是最灵活的输出格式，因为能手动选择，而且能在恢复过程中将归档项重新排序。该格式默认状态下会被压缩。
  - d|directory：该格式会创建一个目录，该目录包含两类文件，一类是目录文件，另一类是每个表和 blob 对象对应的数据文件。
  - t|tar：输出一个 tar 格式的归档形式，作为 gs_restore 输入信息。tar 格式与目录格式兼容；tar 格式归档形式在提取过程中会生成一个有效的目录格式归档形式。但是，tar 格式不支持压缩且对于单独表有 8GB 的大小限制。此外，表数据项的相应排序在恢复过程中不能更改。

    输出一个 tar 格式的归档形式，也可以作为 gsql 输入信息。

- -v, --verbose

  指定 verbose 模式。该选项将导致 gs_dump 向转储文件输出详细的对象注解和启动/停止次数，向标准错误流输出处理信息。

- -V, --version

  打印 gs_dump 版本，然后退出。

- -Z, --compress=0-9

  指定使用的压缩比级别。

  取值范围：0~9

  - 0 表示无压缩。
  - 1 表示压缩比最小，处理速度最快。
  - 9 表示压缩比最大，处理速度最慢。

  针对自定义归档格式，该选项指定单个表数据片段的压缩，默认方式是以中等级别进行压缩。对于文本输出，设置非零压缩级别将会导致整个输出文件被压缩（类似通过 gzip 进行压缩），默认不压缩。tar 归档格式目前不支持压缩。

- --lock-wait-timeout=TIMEOUT

请勿在转储刚开始时一直等待以获取共享表锁。如果无法在指定时间内锁定某个表，就选择失败。可以以任何符合 SET statement_timeout 的格式指定超时时间。

- -?, --help

  显示 gs_dump 命令行参数帮助，然后退出。

转储参数：

- -a, --data-only

  只输出数据，不输出模式(数据定义)。转储表数据、大对象和序列值。

- -b, --blobs

  该参数为扩展预留接口，不建议使用。

- -c, --clean

  在将创建数据库对象的指令输出到备份文件之前，先将清理（删除）数据库对象的指令输出到备份文件中。（如果目标数据库中没有任何对象，gs_restore 工具可能会输出一些提示性的错误信息）

  该选项只对文本格式有意义。针对归档格式，可以在调用 gs_restore 时指定选项。

- -C, --create

  备份文件以创建数据库和连接到创建的数据库的命令开始。（如果命令脚本是这种方式执行，无所谓在运行脚本之前连接的是哪个数据库。）

  该选项只对文本格式有意义。针对归档格式，可以在调用 gs_restore 时指定选项。

- -E, --encoding=ENCODING

  以指定的字符集编码创建转储。默认情况下，以数据库编码创建转储。（得到相同结果的另一个办法是将环境变量“PGCLIENTENCODING”设置为所需的转储编码。）

- -n, --schema=SCHEMA

  只转储与模式名称匹配的模式，此选项包括模式本身和所有它包含的对象。如果该选项没有指定，所有在目标数据库中的非系统模式将会被转储。写入多个-n 选项来选择多个模式。此外，根据 gsql 的\d 命令所使用的相同规则，模式参数可被理解成一个 pattern，所以多个模式也可以通过在该 pattern 中写入通配符来选择。使用通配符时，注意给 pattern 打引号，防止 shell 扩展通配符。

  说明

  - 当-n 已指定时，gs_dump 不会转储已选模式所附着的任何其他数据库对象。因此，无法保证某个指定模式的转储结果能够自行成功地储存到一个空数据库中。
  - 当-n 指定时，非模式对象不会被转储。

  转储支持多个模式的转储。多次输入-n schemaname 转储多个模式。

  例如：

  ```
  gs_dump -h host_name -p port_number postgres -f backup/bkp_sh12.sql -n sch1 -n sch2
  ```

  在上面这个例子中，sch1 和 sch2 会被转储。

- -N, --exclude-schema=SCHEMA

  不转储任何与模式 pattern 匹配的模式。Pattern 将参照针对-n 的相同规则来理解。可以通过输入多次-N，不转储与任何 pattern 匹配的模式。

当同时输入-n 和-N 时，会转储与至少一个-n 选项匹配、与-N 选项不匹配的模式。如果有-N 没有-n，则不转储常规转储中与-N 匹配的模式。

转储过程支持排除多个模式。

在转储过程中，输入-N exclude schema name 排除多个模式。

例如：

```
gs_dump -h host_name -p port_number postgres -f backup/bkp_sh12.sql -N sch1 -N sch2
```

在上面这个例子中，sch1 和 sch2 在转储过程中会被排除。

- -o, --oids

  转储每个表的对象标识符（OIDs），作为表的一部分数据。该选项用于应用以某种方式（例如：外键约束方式）参照了 OID 列的情况。如果不是以上这种情况，请勿使用该选项。

- -O, --no-owner

  不输出设置对象的归属这样的命令，以匹配原始数据库。默认情况下，gs_dump 会发出 ALTER OWNER 或 SET SESSION AUTHORIZATION 语句设置所创建的数据库对象的归属。如果脚本正在运行，该语句不会执行成功，除非是由系统管理员触发（或是拥有脚本中所有对象的同一个用户）。通过指定-O，编写一个任何用户都能存储的脚本，且该脚本会授予该用户拥有所有对象的权限。

  该选项只对文本格式有意义。针对归档格式，可以在调用 gs_restore 时指定选项。

- -s, --schema-only

  只转储对象定义（模式），而非数据。

- -S, --sysadmin=NAME

  该参数为扩展预留接口，不建议使用。

- -t, --table=TABLE

  指定转储的表（或视图、或序列、或外表）对象列表，可以使用多个-t 选项来选择多个表，也可以使用通配符指定多个表对象。

  当使用通配符指定多个表对象时，注意给 pattern 打引号，防止 shell 扩展通配符。

  当使用-t 时，-n 和-N 没有任何效应，这是因为由-t 选择的表的转储不受那些选项的影响。

说明

-t 参数选项个数必须小于等于 100。

如果-t 参数选项个数大于 100，建议使用参数--include-table-file 来替换。

当-t 已指定时，gs_dump 不会转储已选表所附着的任何其他数据库对象。因此，无法保证某个指定表的转储结果能够自行成功地储存到一个空数据库中。

-t tablename 只转储在默认搜索路径中可见的表。-t '*.tablename'转储数据库下所有模式下的 tablename 表。-t schema.table 转储特定模式中的表。

-t tablename 不会导出表上的触发器信息。

例如：

```
gs_dump -h host_name -p port_number postgres -f backup/bkp_sh12.sql -t schema1.table1 -t schema2.table2
```

在上面这个例子中，schema1.table1 和 schema2.table2 会被转储。

- --include-table-file=FILENAME

  指定需要 dump 的表文件。

- -T, --exclude-table=TABLE

  不转储的表（或视图、或序列、或外表）对象列表，可以使用多个-t 选项来选择多个表，也可以使用通配符指定多个表对象。

  当同时输入-t 和-T 时，会转储在-t 列表中，而不在-T 列表中的表对象。

  例如：

  ```
  gs_dump -h host_name -p port_number postgres -f backup/bkp_sh12.sql -T table1 -T table2
  ```

  在上面这个例子中，table1 和 table2 在转储过程中会被排除。

- --exclude-table-file=FILENAME

  指定不需要 dump 的表文件。

  说明

  同--include-table-file，其内容格式如下：

  schema1.table1

  schema2.table2

  .....

- -x, --no-privileges|--no-acl

  防止转储访问权限（授权/撤销命令）。

- --column-inserts|--attribute-inserts

  以 INSERT 命令带列名（INSERT INTO 表（列、…）值…）方式导出数据。这会导致恢复缓慢。但是由于该选项会针对每行生成一个独立分开的命令，所以在重新加载某行时出现的错误只会导致那行丢失，而非整个表内容。

- --disable-dollar-quoting

  该选项将禁止在函数体前使用美元符号$，并强制使用 SQL 标准字符串语法对其进行引用。

- --disable-triggers

  该参数为扩展预留接口，不建议使用。

- --exclude-table-data=TABLE

  指定不转储任何匹配表 pattern 的表这方面的数据。依照针对-t 的相同规则理解该 pattern。

  可多次输入--exclude-table-data 来排除匹配任何 pattern 的表。当用户需要特定表的定义但不需要其中的数据时，这个选项很有帮助。

  排除数据库中所有表的数据，参见--schema-only。

- --inserts

  发出 INSERT 命令（而非 COPY 命令）时转储数据。这会导致恢复缓慢。

  但是由于该选项会针对每行生成一个独立分开的命令，所以在重新加载某行时出现的错误只会导致那行丢失，而非整个表内容。注意如果重排列顺序，可能会导致恢复整个失败。列顺序改变时，--column-inserts 选项不受影响，虽然会更慢。

- --no-security-labels

  该参数为扩展预留接口，不建议使用。

- --no-tablespaces

  不输出选择表空间的命令。使用该选项，无论默认表空间是哪个，在恢复过程中所有对象都会被创建。

  该选项只对文本格式有意义。针对归档格式，可以在调用 gs_restore 时指定选项。

- --no-unlogged-table-data

  该参数为扩展预留接口，不建议使用。

- --non-lock-table

  该参数为扩展预留接口，不建议使用。

- --quote-all-identifiers

  强制对所有标识符加引号。为了向后续版本迁移，且其中可能涉及引入额外关键词，在转储相应数据库时该选项会有帮助。

- --section=SECTION

  指定已转储的名称区段（pre-data、data、和 post-data）。

- --serializable-deferrable

  转储过程中使用可串行化事务，以确保所使用的快照与之后的数据库状态一致；要实现该操作需要在无异常状况的事务流中等待某个点，因为这样才能保证转储成功，避免引起其他事务出现 serialization_failure 要重新再做。

  但是该选项对于灾难恢复没有益处。对于在原始数据库进行升级的时候，加载一个数据库的拷贝作为报告或其他只读加载共享的转储是有帮助的。没有这个选项，转储会反映一个与任何事务最终提交的序列化执行不一致的状态。

  如果当 gs_dump 启动时，读写事务仍处于非活动状态，即便使用该选项也不会对其产生影响。如果读写事务处于活动状态，转储的开始时间可能会延迟一段不确定的时间。

- --use-set-session-authorization

  输出符合 SQL 标准的 SET SESSION AUTHORIZATION 命令而不是 ALTER OWNER 命令来确定对象所有权。这样令转储更加符合标准，但是如果转储文件中的对象的历史有些问题，那么可能不能正确恢复。并且，使用 SET SESSION AUTHORIZATION 的转储需要数据库系统管理员的权限才能转储成功，而 ALTER OWNER 需要的权限则低得多。

- --with-encryption=AES128

  指定转储数据需用 AES128 进行加密。

- --with-key=KEY

  AES128 密钥长度必须是 16 字节。

- --include-nodes

  将 TO NODE/TO GROUP 语句包含在已转储的 CREATE TABLE/CREATE FOREIGN TABLE 语句中。该参数只对 HDFS 表和外表生效。

- --include-extensions

  在转储中包含扩展。

- --include-depend-objs

  备份结果包含依赖于指定对象的对象信息。该参数需要同-t/--include-table-file 参数关联使用才会生效。

- --exclude-self

备份结果不包含指定对象自身的信息。该参数需要同-t/--include-table-file 参数关联使用才会生效。

- --dont-overwrite-file

  文本、tar、以及自定义格式情况下会重写现有文件。这对目录格式不适用。

  例如：

  设想这样一种情景，即当前目录下 backup.sql 已存在。如果在输入命令中输入-f backup.sql 选项时，当前目录恰好也生成 backup.sql，文件就会被重写。

  如果备份文件已存在，且输入--dont-overwrite-file 选项，则会报告附带‘转储文件已经存在’信息的错误。

```
gs_dump -p port_number postgres -f backup.sql -F plain --dont-overwrite-file
```

说明

- -s/--schema-only 和-a/--data-only 不能同时使用。
- -c/--clean 和-a/--data-only 不能同时使用。
- --inserts/--column-inserts 和-o/--oids 不能同时使用，因为 INSERT 命令不能设置 OIDS。
- --role 和--rolepassword 必须一起使用。
- --binary-upgrade-usermap 和--binary-upgrade 必须一起使用。
- --include-depend-objs/--exclude-self 需要同-t/--include-table-file 参数关联使用才会生效
- --exclude-self 必须同--include-depend-objs 一起使用。

连接参数：

- -h, --host=HOSTNAME

  指定主机名称。如果数值以斜杠开头，则被用作到 Unix 域套接字的路径。缺省从 PGHOST 环境变量中获取（如果已设置），否则，尝试一个 Unix 域套接字连接。

  该参数只针对集群外，对集群内本机只能用 127.0.0.1。

  例如：主机名

  环境变量：PGHOST

- -p, --port=PORT

  指定主机端口。

  环境变量：PGPORT

- -U, --username=NAME

  指定所连接主机的用户名。

  环境变量：PGUSER

- -w, --no-password

  不出现输入密码提示。如果主机要求密码认证并且密码没有通过其它形式给出，则连接尝试将会失败。 该选项在批量工作和不存在用户输入密码的脚本中很有帮助。

- -W, --password=PASSWORD

  指定用户连接的密码。如果主机的认证策略是 trust，则不会对系统管理员进行密码验证，即无需输入-W 选项；如果没有-W 选项，并且不是系统管理员，“Dump Restore 工具”会提示用户输入密码。

- --role=ROLENAME

  指定创建转储使用的角色名。选择该选项，会使 gs_dump 连接数据库后，发起一个 SET ROLE 角色名命令。当所授权用户（由-U 指定）没有 gs_dump 要求的权限时，该选项会起到作用，即切换到具备相应权限的角色。某些安装操作规定不允许直接以超系统管理员身份登录，而使用该选项能够在不违反该规定的情况下完成转储。

- --rolepassword=ROLEPASSWORD

  指定角色名的密码。

说明

场景 1

如果某数据库集群有任何本地数据要添加到 template1 数据库，请谨慎将 gs_dump 的输出恢复到一个真正的空数据库中，否则可能会因为被添加对象的定义被复制，出现错误。要创建一个无本地添加的空数据库，需从 template0 而非 template1 复制，例如：

```
CREATE DATABASE foo WITH TEMPLATE template0;
```

tar 归档形式的文件大小不得超过 8GB（tar 文件格式的固有限制）。tar 文档整体大小和任何其他输出格式没有限制，操作系统可能对此有要求。

由 gs_dump 生成的转储文件不包含优化程序用来做执行计划决定的统计数据。因此，最好从某转储文件恢复之后运行 ANALYZE 以确保最佳效果。转储文件不包含任何 ALTER DATABASE…SET 命令，这些设置由 gs_dumpall 转储，还有数据库用户和其他完成安装设置。

场景 2

当 SEQUENCE 已经到达最大或最小值时，通过 gs_dump 来备份 SEQUENCE 值会因执行报错退出。可参考如下说明处理：

1. SEQUENCE 已经到达最大值，但最大值小于 2^63-2

报错示例：

sequence 对象定义

```
CREATE SEQUENCE seq INCREMENT 1 MINVALUE 1 MAXVALUE 3 START WITH 1;
```

执行 gs_dump 备份

```
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
```

处理方法：

通过 SQL 语句连接 postgres 数据库，执行如下语句，修改 sequence seq1 的最大值。

```
gsql -p 37300 postgres -r -c "ALTER SEQUENCE PUBLIC.seq MAXVALUE 10;"
```

执行 dump 工具进行备份。

```
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. SEQUENCE 已经到达最小值或最大值 2^63-2

gs_dump 不支持该场景下的 SEQUENCE 数值备份。

说明

SQL 端不支持 SEQUENCE 到达最大值 2^63-2 后的 MAXVALUE 修改，不支持 SEQUENCE 到达最小值后的 MINVALUE 修改。

场景 3

gs_dump 主要用于全库元数据导出场景，对导出单表做过性能优化，但是导出多表性能较差。对于导出多表场景，建议逐个表导出。例如：

```
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
```

如果业务停止情况下，或者业务空闲期，可以增加--non-lock-table 参数提升 gs_dump 的性能。例如：

```
gs_dump -U dbadmin -W {password} -p 37300 postgres -t public.table03 -s --non-lock-
table -f backup/table03.sql
```

示例

使用 gs_dump 转储数据库为 SQL 文本文件或其它格式的操作，如下所示。

示例中“password”表示数据库用户密码，由用户自己设置；“backup/MPPDB_backup.sql”表示导出的文件，其中 backup 表示相对于当前目录的相对目录；“37300”表示数据库服务器端口；“postgres”表示要访问的数据库名。

说明

导出操作时，请确保该目录存在并且当前的操作系统用户对其具有读写权限。

示例 1：执行 gs_dump，导出 postgres 数据库全量信息，导出的 MPPDB_backup.sql 文件格式为纯文本格式。

```
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
```

使用 gsql 程序从纯文本导出文件中导入数据。

示例 2：执行 gs_dump，导出 postgres 数据库全量信息，导出的 MPPDB_backup.tar 文件格式为 tar 格式。

```
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
```

示例 3：执行 gs_dump，导出 postgres 数据库全量信息，导出的 MPPDB_backup.dmp 文件格式为自定义归档格式。

```
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
```

示例 4：执行 gs_dump，导出 postgres 数据库全量信息，导出的 MPPDB_backup 文件格式为目录格式。

```
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
```

示例 5：执行 gs_dump，导出 postgres 数据库信息，但不导出/home/MPPDB_temp.sql 中指定的表信息。导出的 MPPDB_backup.sql 文件格式为纯文本格式。

```
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
```

示例 6：执行 gs_dump，仅导出依赖于指定表 testtable 的视图信息。然后创建新的 testtable 表，再恢复依赖其上的视图。

备份仅依赖于 testtable 的视图

```
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
```

修改 testtable 名称

```
gsql -p 37300 postgres -r -c "ALTER TABLE PUBLIC.testtable RENAME TO
testtable_bak;"
```

创建新的 testtable 表

```
CREATE TABLE PUBLIC.testtable(a int, b int, c int);
```

还原依赖于 testtable 的视图

```
gsql -p 37300 postgres -r -f backup/MPPDB_backup.sql
```

相关命令

gs_dumpall，gs_restore

7.2 gs_dumpall

背景信息

gs_dumpall 是 GaussDB(DWS)用于导出所有数据库相关信息工具，它可以导出集群数据库的所有数据，包括默认数据库 postgres 的数据、自定义数据库的数据、以及集群所有数据库公共的全局对象。

gs_dumpall 工具在进行数据导出时，其他用户可以访问集群数据库（读或写）。

gs_dumpall 工具支持导出完整一致的数据。例如，T1 时刻启动 gs_dumpall 导出整个集群数据库，那么导出数据结果将会是 T1 时刻该集群数据库的数据状态，T1 时刻之后对集群数据库的修改不会被导出。

gs_dumpall 在导出整个集群所有数据库时分为两部分：

- gs_dumpall 自身对所有数据库公共的全局对象进行导出，包括有关数据库用户和组，表空间以及属性（例如，适用于数据库整体的访问权限）信息。

- gs_dumpall 通过调用 gs_dump 来完成集群中各数据库的 SQL 脚本文件导出，该脚本文件包含将数据库恢复为其保存时的状态所需要的全部 SQL 语句。

以上两部分导出的结果为纯文本格式的 SQL 脚本文件，使用 gsql 运行该脚本文件可以恢复集群数据库。

注意事项

- 禁止修改导出的文件和内容，否则可能无法恢复成功。
- 为了保证数据一致性和完整性，gs_dumpall 会对需要转储的表设置共享锁。如果某张表在别的事务中设置了共享锁，gs_dumpall 会等待此表的锁释放后锁定此表。如果无法在指定时间内锁定某张表，转储会失败。用户可以通过指定--lock-wait-timeout 选项，自定义等待锁超时时间。
- 由于 gs_dumpall 读取所有数据库中的表，因此必须以数据库集群管理员身份进行连接，才能导出完整文件。在使用 gsql 执行脚本文件导入时，同样需要管理员权限，以便添加用户和组，以及创建数据库。

语法

```
gs_dumpall [OPTION]...
```

参数说明

通用参数：

- -f, --filename=FILENAME

  将输出发送至指定文件。如果这里省略，则使用标准输出。
- -v, --verbose

  指定 verbose 模式。该选项将导致 gs_dumpall 向转储文件输出详细的对象注解和启动/停止次数，向标准错误流输出处理信息。
- -V, --version

  打印 gs_dumpall 版本，然后退出。
- --lock-wait-timeout=TIMEOUT

  请勿在转储刚开始时一直等待以获取共享表锁。如果无法在指定时间内锁定某个表，就选择失败。可以以任何符合 SET statement_timeout 的格式指定超时时间。
- -?, --help

  显示 gs_dumpall 命令行参数帮助，然后退出。

转储参数：

- -a, --data-only

  只转储数据，不转储模式（数据定义）。
- -c, --clean

  在重新创建数据库之前，执行 SQL 语句清理（删除）这些数据库。针对角色和表空间的转储命令已添加。
- -g, --globals-only

  只转储全局对象（角色和表空间），无数据库。

- -o, --oids

  转储每个表的对象标识符（OIDs），作为表的一部分数据。该选项用于应用以某种方式（例如：外键约束方式）参照了 OID 列的情况。如果不是以上这种情况，请勿使用该选项。

- -O, --no-owner

  不输出设置对象的归属这样的命令，以匹配原始数据库。默认情况下，gs_dumpall 会发出 ALTER OWNER 或 SET SESSION AUTHORIZATION 语句设置所创建的模式元素的所属。如果脚本正在运行，该语句不会执行成功，除非是由系统管理员触发（或是拥有脚本中所有对象的同一个用户）。通过指定-O，编写一个任何用户都能存储的脚本，且该脚本会授予该用户拥有所有对象的权限。

- -r, --roles-only

  只转储角色，不转储数据库或表空间。

- -s, --schema-only

  只转储对象定义（模式），而非数据。

- -S, --sysadmin=NAME

  在转储过程中使用的系统管理员名称。

- -t, --tablespaces-only

  只转储表空间，不转储数据库或角色。

- -x, --no-privileges

  防止转储访问权限（授权/撤销命令）。

- --column-inserts|--attribute-inserts

  以 INSERT 命令带列名（INSERT INTO 表（列、…）值…）方式导出数据。这会导致恢复缓慢。但是由于该选项会针对每行生成一个独立分开的命令，所以在重新加载某行时出现的错误只会导致那行丢失，而非整个表内容。

- --disable-dollar-quoting

  该选项将禁止在函数体前使用美元符号$，并强制使用 SQL 标准字符串语法对其进行引用。

- --disable-triggers

  该参数为扩展预留接口，不建议使用。

- --inserts

  发出 INSERT 命令（而非 COPY 命令）时转储数据。这会导致恢复缓慢。注意如果重排列顺序，可能会导致恢复整个失败。--column-inserts 选项更加安全，虽然可能更慢些。

- --no-security-labels

  该参数为扩展预留接口，不建议使用。

- --no-tablespaces

  请勿输出创建表空间的命令，也请勿针对对象选择表空间。使用该选项，无论默认表空间是哪个，在恢复过程中所有对象都会被创建。

- --no-unlogged-table-data

  该参数为扩展预留接口，不建议使用。

- --quote-all-identifiers

强制对所有标识符加引号。为了向后续版本迁移，且其中可能涉及引入额外关键词，在转储相应数据库时该选项会有帮助。

- --dont-overwrite-file

  不重写当前文件。

- --use-set-session-authorization

  输出符合 SQL 标准的 SET SESSION AUTHORIZATION 命令而不是 ALTER OWNER 命令来确定对象所有权。这样令转储更加符合标准，但是如果转储文件中的对象的历史有些问题，那么可能不能正确恢复。并且，使用 SET SESSION AUTHORIZATION 的转储需要数据库系统管理员的权限才能转储成功，而 ALTER OWNER 需要的权限则低得多。

- --with-encryption=AES128

  指定转储数据需用 AES128 进行加密。

- --with-key=KEY

  AES128 密钥长度必须是 16 字节。

- --include-extensions

  如果 include-extensions 参数被设置，将备份所有的 CREATE EXTENSION 语句。

- --include-templatedb

  转储过程中包含模板库。

- --dump-nodes

  转储过程中包含节点和 Node Group。

- --include-nodes

  将 TO NODE 语句包含在已转储的 CREATE TABLE 命令中。

- --include-buckets

  该参数为扩展预留接口，不建议使用。

- --dump-wrm

  存储过程中包含负载资源管理器，具体包括资源池、负载组以及负载组映射。

- --binary-upgrade

  该参数为扩展预留接口，不建议使用。

- --binary-upgrade-usermap="USER1=USER2"

  该参数为扩展预留接口，不建议使用。

- --tablespaces-postfix

  该参数为扩展预留接口，不建议使用。

- --parallel-jobs

  指定备份进程并发数，取值范围为 1~1000。

说明

- -g/--globals-only 和-r/--roles-only 不能同时使用。
- -g/--globals-only 和-t/--tablespaces-only 不能同时使用。
- -r/--roles-only 和-t/--tablespaces-only 不能同时使用。
- -s/--schema-only 和-a/--data-only 不能同时使用。
- -r/--roles-only 和-a/--data-only 不能同时使用。

- -t/--tablespaces-only 和-a/--data-only 不能同时使用。
- -g/--globals-only 和-a/--data-only 不能同时使用。
- --tablespaces-postfix 和--binary-upgrade 必须一起使用。
- --parallel-jobs 和-f/--file 必须一起使用。

连接参数：

- -h, --host

  指定主机的名称。如果取值是以斜线开头，它将用作 Unix 域套接字的目录。默认值取自 PGHOST 环境变量；如果没有设置，将启动某个 Unix 域套接字建立连接。

  该参数只针对集群外，对集群内本机只能用 127.0.0.1。

  环境变量：PGHOST

- -l, --database

  指定所连接的转储全局对象的数据库名称，并去寻找还有其他哪些数据库需要被转储。如果没有指定，会使用 postgres 数据库，如果 postgres 数据库不存在，会使用 template1。

- -p, --port

  指定服务器所监听的 TCP 端口或本地 Unix 域套接字后缀，以确保连接。默认值设置为 PGPORT 环境变量。

  环境变量：PGPORT

- -U, --username

  所连接的用户名。

  环境变量：PGUSER

- -w, --no-password

  不出现输入密码提示。如果服务器要求密码认证并且密码没有通过其它形式给出，则连接尝试将会失败。 该选项在批量工作和不存在用户输入密码的脚本中很有帮助。

- -W, --password

  指定用户连接的密码。如果主机的认证策略是 trust，则不会对系统管理员进行密码验证，即无需输入-W 选项；如果没有-W 选项，并且不是系统管理员，“Dump Restore 工具”会提示用户输入密码。

- --role

  指定创建转储使用的角色名。选择该选项，会使 gs_dumpall 连接数据库后，发起一个 SET ROLE 角色名命令。当所授权用户（由-U 指定）没有 gs_dumpall 要求的权限时，该选项会起到作用，即切换到具备相应权限的角色。某些安装操作规定不允许直接以系统管理员身份登录，而使用该选项能够在不违反该规定的情况下完成转储。

- --rolepassword

  指定具体角色用户的角色密码。

说明

由于 gs_dumpall 内部调用 gs_dump，所以一些诊断信息参见 gs_dump。

一旦恢复，最好在每个数据库上运行 ANALYZE，优化程序提供有用的统计数据。

gs_dumpall 恢复前需要所有必要的表空间目录才能退出；否则，对于处在非默认位置的数据库，数据库创建会失败。

示例

使用 gs_dumpall 一次导出集群的所有数据库。

说明

gs_dumpall 仅支持纯文本格式导出。所以只能使用 gsql 恢复 gs_dumpall 导出的转储内容。

```
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
```

相关命令

gs_dump，gs_restore

7.3 gs_restore

背景信息

gs_restore 是 GaussDB(DWS)提供的针对 gs_dump 导出数据的导入工具。通过此工具可由 gs_dump 生成的导出文件进行导入。

主要功能包含：

- 导入到数据库

  如果连接参数中指定了数据库，则数据将被导入到指定的数据库中。其中，并行导入必须指定连接的密码。
- 导入到脚本文件

  如果未指定导入数据库，则创建包含重建数据库所必须的 SQL 语句脚本并写入到文件或者标准输出。等效于直接使用 gs_dump 导出为纯文本格式。

命令格式

```
gs_restore [OPTION]... FILE
```

说明

- FILE 没有短选项或长选项。用来指定归档文件所处的位置。

- 作为前提条件，需输入 dbname 或-l 选项。不允许用户同时输入 dbname 和-l 选项。
- gs_restore 默认是以追加的方式进行数据导入。为避免多次导入造成数据异常，在进行导入时，建议使用"-e"和"-c" 参数，即导入前删除已存在于待导入数据库中的数据库对象，同时当出现导入错误时，忽略当前错误，继续执行导入任务，并在导入后会显示相应的错误信息。

参数说明

通用参数：

- -d, --dbname=NAME
  连接数据库 dbname 并直接导入到该数据库中。
- -f, --file=FILENAME
  指定生成脚本的输出文件，或使用-l 时列表的输出文件。
  默认是标准输出。

  说明

  -f 不能同-d 一起使用。
- -F, --format=c|d|t
  指定归档格式。由于 gs_restore 会自动决定格式，因此不需要指定格式。
  取值范围：
  - c/custom：该归档形式为 4.21-gs_dump 的自定义格式。
  - d/directory：该归档形式是一个目录归档形式。
  - t/tar：该归档形式是一个 tar 归档形式。
- -l, --list
  列出归档形式内容。这一操作的输出可用作-L 选项的输入。注意如果像-n 或-t 的过滤选项与-l 使用，过滤选项将会限制列举的项目（即归档形式内容）。
- -v, --verbose
  指定 verbose 模式。
- -V, --version
  打印 gs_restore 版本，然后退出。
- -?, --help
  显示 gs_restore 命令行参数帮助，然后退出。

导入参数：

- -a, -data-only
  只导入数据，不导入模式（数据定义）。gs_restore 的导入是以追加方式进行的。
- -c, --clean
  在重新创建数据库对象前，清理（删除）已存在于将要还原的数据库中的数据库对象
- -C, --create

导入到数据库之前请创建数据库。（选择该选项后，以-d 打头的数据库将被用作发布首个 CREATE DATABASE 命令。所有数据将被导入到出现在归档文件的数据库中。）

- -e, --exit-on-error

  当发送 SQL 语句到数据库时如果出现错误，请退出。默认状态下会继续，且在导入后会显示一系列错误信息。

- -I, --index=NAME

  只导入已列举的 index 的定义。允许导入多个 index。如果多次输入-I index 导入多个 index。

  例如：

  ```
  gs_restore -h host_name -p port_number -d gaussdb -I Index1 -I Index2
  backup/MPPDB_backup.tar
  ```

  在上面这个例子中，Index1 和 Index2 会被导入。

- -j, --jobs=NUM

  运行 gs_restore 最耗时的部分（如加载数据、创建 index、或创建约束）使用并发任务。该选项能大幅缩短导入时间，即将一个大型数据库导入到某一多处理器的服务器上。

  每个任务可能是一个进程或一个线程，这由操作系统决定；每个任务与服务器进行单独连接。

  该选项的最优值取决于服务器的硬件设置、客户端、以及网络。还包括这些因素，如 CPU 核数量、硬盘设置。建议是从增加服务器上的 CPU 核数量入手，更大的值（服务器上 CPU 核数量）在很多情况下也能导致数据文件更快的被导入。当然，过高的值会由于超负荷反而导致性能降低。

  该选项只支持自定义归档格式。输入文件必须是常规文件（不能是像 pipe 的文件）。如果是通过脚本文件，而非直接连接数据库服务器，该选项可忽略。而且，多任务不能与--single-transaction 选项一起使用。

- -L, --use-list=FILENAME

  只导入列举在 list-file 中的那些归档形式元素，导入顺序以它们在文件中的顺序为准。注意如果像-n 或-t 的过滤选项与-L 使用，它们将会进一步限制导入的项目。

  一般情况下，list-file 是通过编辑前面提到的某个-l 参数的输出创建的。文件行的位置可更改或直接删除行，也可使用分号（;）在行的开始注出。见下文的举例。

- -n, --schema=NAME

  只导入已列举的模式中的对象。

  该选项可与-t 选项一起用以导入某个指定的表。

  多次输入-n *schemaname* 可以导入多个模式。

  例如：

  ```
  gs_restore -h host_name -p port_number -d gaussdb -n sch1 -n sch2
  backup/MPPDB_backup.tar
  ```

  在上面这个例子中，sch1 和 sch2 会被导入。

- -O, --no-owner

  不输出设置对象的归属这样的命令，以匹配原始数据库。默认情况下，gs_restore 会发出 ALTER OWNER 或 SET SESSION AUTHORIZATION 语句设置所创建的模式元素的所属。除非是由系统管理员（或是拥有脚本中所有对象的同一个用户）

进行数据库首次连接的操作，否则语句会失败。使用-O 选项，任何用户名都可用于首次连接，且该用户拥有所有已创建的对象。

- -P, --function=NAME(args)

  只导入已列举的函数。请按照函数所在转储文件中的目录，准确拼写函数名称和参数。

  当-P 单独使用时，表示导入文件中所有'function-name(args)'函数；当-P 同-n 一起使用时，表示导入指定模式下的'function-name(args)'函数；多次输入-P，而仅指定一次-n，表示所有导入的函数默认都是位于-n 模式下的。

  可以多次输入-n schema-name -P 'function-name(args)'同时导入多个指定模式下的函数。

  例如：

  ```
  ./gs_restore -h host_name -p port_number -d gaussdb -n test1 -P
  'Func1(integer)' -n test2 -P 'Func2(integer)' backup/MPPDB_backup.tar
  ```

  在上面这个例子中，test1 模式下的函数 Func1(i integer)和 test2 模式下的函数 Func2(j integer)会被一起导入。

- -s, --schema-only

  只导入模式（数据定义），不导入数据（表内容）。当前的序列值也不会导入。

- -S, --sysadmin=NAME

  该参数为扩展预留接口，不建议使用。

- -t, --table=NAME

  只导入已列举的表定义、数据或定义和数据。该选项与-n 选项同时使用时，用来指定某个模式下的表对象。-n 参数不输入时，默认为 PUBLIC 模式。多次输入-n <schemaname> -t <tablename>可以导入指定模式下的多个表。

  例如：

  导入 PUBLIC 模式下的 table1

  ```
  gs_restore -h host_name -p port_number -d gaussdb -t table1
  backup/MPPDB_backup.tar
  ```

  导入 test1 模式下的 test1 和 test2 模式下 test2

  ```
  gs_restore -h host_name -p port_number -d gaussdb -n test1 -t test1 -n test2 -t
  test2 backup/MPPDB_backup.tar
  ```

  导入 PUBLIC 模式下的 table1 和 test1 模式下 test1

  ```
  gs_restore -h host_name -p port_number -d gaussdb -n PUBLIC -t table1 -n test1
  -t table1 backup/MPPDB_backup.tar
  ```

须知

-t 不支持 schema_name.table_name 的输入格式。

- -T, --trigger=NAME

  该参数为扩展预留接口。

- -x, --no-privileges/--no-acl

  防止导入访问权限（grant/revoke 命令）。

- -1, --single-transaction

  执行导入作为一个单独事务（即把命令包围在 BEGIN/COMMIT 中）。

  该选项确保要么所有命令成功完成，要么没有改变应用。该选项意为--exit-on-error。

- --disable-triggers

  该参数为扩展预留接口，不建议使用。

- --no-data-for-failed-tables

  默认状态下，即使创建表的命令失败（如表已经存在），表数据仍会被导入。使用该选项，像这种表的数据会被跳过。如果目标数据库已包含想要的表内容，这种行为会有帮助。

  该选项只有在直接导入到某数据库中时有效，不针对生成 SQL 脚本文件输出。

- --no-security-labels

  该参数为扩展预留接口，不建议使用。

- --no-tablespaces

  不输出选择表空间的命令。使用该选项，无论默认表空间是哪个，在导入过程中所有对象都会被创建。

- --section=SECTION

  导入已列举的区段（如 pre-data、data、或 post-data）。

- --use-set-session-authorization

  该选项用来进行文本格式的备份。

  输出 SET SESSION AUTHORIZATION 命令，而非 ALTER OWNER 命令，用以决定对象归属。该选项使转储更加兼容标准，但通过参考转储中对象的记录，导入过程可能会有问题。使用 SET SESSION AUTHORIZATION 的转储要求必须是系统管理员，同时在导入前还需参考"SET SESSION AUTHORIZATION"，手工对导出文件的密码进行修改验证，只有这样才能进行正确的导入操作，相比之下，ALTER OWNER 对权限要求较低。

- --with-key=KEY

  AES128 密钥长度必须是 16 字节。

说明

如果转储被加密，则必须在 gs_restore 命令中输入--with-key <keyname>选项。如果未输入，用户会收到错误信息。

应该输入转储时所输入的相同的 key。

须知

- 如果安装过程中有任何本地数据要添加到 template1 数据库，请谨慎将 gs_restore 的输出载入到一个真正的空数据库中；否则可能会因为被添加对象的定义被复制，而出现错误。要创建一个无本地添加的空数据库，需从 template0 而非 template1 复制，例如：

```
CREATE DATABASE foo WITH TEMPLATE template0;
```

- gs_restore 不能选择性地导入大对象；例如只能导入那些指定表的对象。如果某个归档形式包含大对象，那所有大对象都会被导入，或一个都不会被导入，如果它们通过-L、-t 或其他选项被排除。

说明

1.-d/--dbname 和 -f/--file 不能同时使用；

2. -s/--schema-only 和 -a/--data-only 不能同时使用；

3. -c/--clean 和 -a/--data-only 不能同时使用；

4. 使用--single-transaction 时，-j/--jobs 必须为单任务；

5. --role 和 --rolepassword 必须一起使用。

连接参数：

- -h, --host=HOSTNAME

  指定的主机名称。如果取值是以斜线开头，他将用作 Unix 域套接字的目录。默认值取自 PGHOST 环境变量；如果没有设置，将启动某个 Unix 域套接字建立连接。

  该参数只针对集群外，对集群内本机只能用 127.0.0.1。

- -p, --port=PORT

  指定服务器所监听的 TCP 端口或本地 Unix 域套接字后缀，以确保连接。默认值设置为 PGPORT 环境变量。

- -U, --username=NAME

  所连接的用户名。

- -w, --no-password

  不出现输入密码提示。如果服务器要求密码认证并且密码没有通过其它形式给出，则连接尝试将会失败。 该选项在批量工作和不存在用户输入密码的脚本中很有帮助。

- -W, --password=PASSWORD

  指定用户连接的密码。如果主机的认证策略是 trust，则不会对系统管理员进行密码验证，即无需输入-W 参数；如果没有-W 参数，并且不是系统管理员，“gs_restore”会提示用户输入密码。

- --role=ROLENAME

  指定导入操作使用的角色名。选择该参数，会使 gs_restore 连接数据库后，发起一个 SET ROLE 角色名命令。当所授权用户（由-U 指定）没有 gs_restore 要求的权限时，该参数会起到作用，即切换到具备相应权限的角色。某些安装操作规定不允许直接以初始用户身份登录，而使用该参数能够在不违反该规定的情况下完成导入。

- --rolepassword=ROLEPASSWORD

  指定具体角色用户的角色密码。

示例

特例：执行 gsql 程序，使用如下选项导入由 gs_dump/gs_dumpall 生成导出文件夹（纯文本格式）的 MPPDB_backup.sql 文件到 gaussdb 数据库。

```
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 用来导入由 gs_dump 生成的导出文件。

示例 1：执行 gs_restore，将导出的 MPPDB_backup.dmp 文件（自定义归档格式）导入到 gaussdb 数据库。

```
gs_restore -W {password} backup/MPPDB_backup.dmp -p 8000 -d gaussdb
gs_restore: restore operation successful
gs_restore: total time: 13053  ms
```

示例 2：执行 gs_restore，将导出的 MPPDB_backup.tar 文件（tar 格式）导入到 gaussdb 数据库。

```
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
```

示例 3：执行 gs_restore，将导出的 MPPDB_backup 文件（目录格式）导入到 gaussdb 数据库。

```
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
```

示例 4：执行 gs_restore，使用自定义归档格式的 MPPDB_backup.dmp 文件来进行如下导入操作。 导入 PUBLIC 模式下所有对象的定义和数据。在导入时会先删除已经存在的对象，如果原对象存在跨模式的依赖则需手工强制干预。

```
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;
```

手工删除依赖，导入完成后再重新创建。

```
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
```

示例 5：执行 gs_restore，使用自定义归档格式的 MPPDB_backup.dmp 文件来进行如下导入操作。只导入 PUBLIC 模式下表 table1 的定义。

```
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
```

示例 6：执行 gs_restore，使用自定义归档格式的 MPPDB_backup.dmp 文件来进行如下导入操作。只导入 PUBLIC 模式下表 table1 的数据。

```
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
```

说明

创建集群的时候会启动调度器，启动调度器时会创建调度器的一些资源，包括调度器的表所在的schema scheduler，调度器运行时创建的几张表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,pg_task 在执行 gs_restore 的时候，会将调度器的表、schema 和索引等对象也一起恢复，由于调度器是一个常驻进程，新建的集群也会自动的创建这些对象，所以在执行 gs_restore 时会发生调度器的对象存在的错误信息，该报错对集群正常操作没有影响，可忽略。

相关命令

gs_dump，gs_dumpall

7.4 gds_check

背景信息

gds_check 用于对 GDS 部署环境进行检查，包括操作系统参数、网络环境、磁盘占用情况等，也支持对可修复系统参数的修复校正，有助于在部署运行 GDS 时提前发现潜在问题，提高执行成功率。

注意事项

- 执行脚本前需设置环境变量，可参考《开发指南》中“导入数据>通过外表并行导入>安装配置和启动 GDS”章节。
- 脚本需要在 python 3 环境下运行。
- 必须在 root 用户下执行脚本。
- 必须指定-t、--host 参数。
- 当--host 指定网络地址 0.0.0.0 或 127.0.0.1 时，不会检查 MTU 和网卡多队列。
- 网卡多队列的检查、修复要求网卡至少是万兆。
- --host 参数指定的所有节点的密码必须保持一致，才能保证脚本成功进行远程检查。
- 执行修复时，对配置劣于推荐值的参数，建议设置为 OS 中配置项的推荐值，具体见下表：

表7-2 OS 配置项

参数	推荐值
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	26000~65535
MTU	1500
net.core.wmem_max	21299200
net.core.rmem_max	21299200
net.core.wmem_default	21299200
net.core.rmem_default	21299200
max handler	1000000
vm.swappiness	10

表7-3 磁盘检查

检查项	警告
磁盘空间使用率	大于等于 70%且小于 90%
inode 使用率	大于等于 70%且小于 90%

表7-4 网络检查

检查项	报错
检查网络连通性	包 100%丢失
检查网卡多队列	开启网卡多队列且绑定不同 CPU，支持 fix 修改

语法

- 检查命令

```
gds_check -t check --host [/path/to/hostfile | ipaddr1,ipaddr2...] --ping-host
[/path/to/pinghostfile | ipaddr1,ipaddr2...] [--detail]
```

- 修复命令

```
gds_check -t fix --host [/path/to/hostfile | ipaddr1,ipaddr2...] [--detail]
```

参数说明

- -t

  操作类型，表示检查/修复。

  取值：check，fix。

- --host

  需要检查/修复的节点 IP 列表。

  取值：IP 列表，支持文件和字符串两种形式。

  - 文件形式：每一行一个 IP 地址，如：

    192.168.1.200

    192.168.1.201

  - 字符串形式：半角逗号分隔，如：

    192.168.1.200,192.168.1.201

- --ping-host

  在各检查节点上进行网络 ping 检查的目标地址。

  取值：IP 列表，支持文件和字符串两种形式，一般是 DN、CN、网关的 IP 地址。

  - 文件形式：每一行一个 IP 地址，如：

    192.168.2.200

    192.168.2.201

  - 字符串形式：半角逗号分隔，如：

    192.168.2.200,192.168.2.201

- --detail

  显示检查/修复项详细信息，并存入日志。

- -V

  显示版本信息。

- -h, --help

显示帮助信息。

示例

执行检查，--host、--ping-host 均为 IP 字符串形式：

```
gds_check -t check --host 192.168.1.100,192.168.1.101 --ping-host 192.168.2.100
```

执行检查，--host 为字符串，--ping-host 为文件形式：

```
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
```

执行检查，--host 为文件形式，--ping-host 为字符串：

```
gds_check -t check --host  /home/gds/iplist --ping-host 192.168.1.100,192.168.1.101
```

执行修复，--host 为字符串：

```
gds_check -t fix --host 192.168.1.100,192.168.1.101
```

执行检查，打印详细信息，并存入日志：

```
gds_check -t check --host 192.168.1.100 --detail
```

执行修复，打印详细信息，并存入日志：

```
gds_check -t fix --host 192.168.1.100 --detail
```

7.5 gds_install

背景信息

gds_install 是用于批量安装 gds 的脚本工具，可大大提高 GDS 部署效率。

注意事项

- 执行脚本前需设置环境变量，可参考《开发指南》中“导入数据>通过外表并行导入>安装配置和启动 GDS”章节。
- 脚本需要在 python 3 环境下运行。
- 必须在 root 用户下执行脚本 gds_install。
- 用户需要检查上层目录权限，保证 GDS 用户对安装操作目录、安装目录及安装包有读写执行的权限。
- 目前不支持跨平台的安装部署。
- 执行命令节点也必须是安装部署机器之一。
- --host 参数指定的所有节点的密码必须保持一致，才能保证脚本成功进行远程部署。

语法

```
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...]]
```

参数说明

- -I

  安装目录。

  默认值：/opt/${gds_user}/packages/，其中${gds_user}表示 GDS 业务的操作系统用户。

- -U

  GDS 用户。

- -G

  GDS 用户所属组。

- --pkg

  GDS 安装包路径，形如/path/to/GaussDB-8.1.3-REDHAT-x86_64bit-Gds.tar.gz。

- --host

  待安装部署节点的 IP 列表，支持文件和字符串两种形式：

  - 文件形式：每一行一个 IP 地址，如：

    192.168.2.200

    192.168.2.201

  - 字符串形式：半角逗号分隔，如：

    192.168.2.200,192.168.2.201。

  说明

  执行命令节点必须是待部署节点之一，其 IP 须在列表中。

- --ping-host

  调用 gds_check 时，在各检查节点上进行网络 ping 检查的目标地址。

  取值：IP 列表，支持文件和字符串两种形式，一般是 DN、CN、网关的 IP 地址。

  - 文件形式：每一行一个 IP 地址，如：

    192.168.2.200

    192.168.2.201

  - 字符串形式：半角逗号分隔，如：

    192.168.2.200,192.168.2.201

- -V

  显示版本信息。

- -h, --help

  显示帮助信息。

示例

将 GDS 安装部署在节点 192.168.1.100、192.168.1.101 上，并指定安装目录为 /opt/gdspackages/install_dir，GDS 用户是 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
```

7.6 gds_uninstall

背景信息

gds_uninstall 是用于批量卸载 GDS 的脚本工具。

注意事项

- 执行脚本前需设置环境变量，可参考《开发指南》中“导入数据>通过外表并行导入>安装配置和启动 GDS”章节。
- 脚本需要在 python 3 环境下运行。
- 必须在 root 用户下执行脚本 gds_uninstall。
- 必须包含--host、-U 参数。
- 目前不支持跨平台的卸载操作。
- --host 参数指定的所有节点的密码必须保持一致，才能保证脚本成功进行远程卸载。

语法

```
gds uninstall --host [/path/to/hostfile | ipaddr1,ipaddr2...] -U gds user [--
delete-user | --delete-user-and-group]
```

参数说明

- --host

  待卸载节点的 IP 列表，支持文件和字符串两种形式：
  - 文件形式：每一行一个 IP 地址，如：

    192.168.2.200

    192.168.2.201
  - 字符串形式：半角逗号分隔，如：

    192.168.2.200,192.168.2.201。
- -U

  GDS 用户。
- --delete-user

  卸载的同时，删除用户。被删除的用户不可以是 root 用户。
- --delete-user-and-group

卸载的同时，删除用户和其所在用户组。仅当用户组只包含该待删除用户一个用户时可以删除用户组。该用户组不能是 root 用户组。

- -V

  显示版本信息。

- -h, --help

  显示帮助信息。

示例

卸载安装部署在节点 192.168.1.100、192.168.1.101 上，安装用户为 gds_test 的，GDS 文件夹及环境变量。

```
gds_uninstall -U gds_test --host 192.168.1.100,192.168.1.101
```

卸载时，同时删除用户。

```
gds_uninstall -U gds_test --host 192.168.1.100,192.168.1.101 --delete-user
```

卸载时，同时删除用户和用户组。

```
gds_uninstall -U gds_test --host 192.168.1.100,192.168.1.101 --delete-user-and-group
```

7.7 gds_ctl

背景信息

gds_ctl 是一个批量控制 GDS 启停的脚本工具，一次执行可以在多个节点上启动/停止相同端口的 GDS 服务进程，并在启动时为每一个进程设置看护程序，用于看护 GDS 进程。

注意事项

- 执行脚本前需切换到 GDS 用户，必须在普通用户下执行脚本 gds_ctl。
- 脚本需要在 python 3 环境下运行。
- gds_ctl 继承了 GDS 主要命令行参数，除-p 以及-h 外，其他参数意义不变。在 gds_ctl 中，-p 只需指定端口即可。
- 使用 gds_ctl 批量操作的节点必须是此前使用 gds_install 安装部署的节点。

语法

- 启动命令

  ```
  gds_ctl start --host [/path/to/hostfile | ipaddr1,ipaddr2...] -p PORT -d DATADIR -H ALLOW_IPs [gds other original options]
  ```

- 停止命令

  ```
  gds_ctl stop --host [/path/to/hostfile | ipaddr1,ipaddr2...] -p PORT
  ```

- 重启命令

  ```
  gds_ctl restart --host [/path/to/hostfile | ipaddr1,ipaddr2...] -p PORT
  ```

参数说明

- --host

  待运行 GDS 节点的 IP 列表，支持文件和字符串两种形式：

  - 文件形式：每一行一个 IP 地址，如：

    192.168.2.200

    192.168.2.201

  - 字符串形式：半角逗号分隔，如：

    192.168.2.200,192.168.2.201

- -p

  监听端口。

  取值范围：1024~65535，正整数。

  默认值：8098

- --help

  显示帮助信息。

- -V

  显示版本信息。

兼容 GDS 原参数

- -d dir

  设置待导入数据文件的目录。在 GDS 进程权限允许的条件下，-d 指定的目录会自动被创建。

- -l log_file

  设置日志文件。

  与-R 参数一起使用，可支持日志自动切分。当设置-R 参数后，GDS 会根据设置的值重新生成新的文件，以此来避免单个日志文件过大。

  生成规则：GDS 默认只识别后缀是 log 的文件重新生成日志文件。

  例如，当-l 参数指定为 gds.log，-R 指定为 20MB 时，当 gds.log 达到 20MB 后就会新创建一个 "gds-2020-01-17_115425.log"文件。

  当-l 指定的日志文件没有以 log 为后缀，例如：" gds.log.txt"， 则新创建的日志文件名为" gds.log-2020-01-19_122739.txt"。

  GDS 启动时会检测-l 参数设置的日志文件是否存在，如果存在则根据当前日期时间新生成一个日志文件，不会覆盖之前的日志文件。

- -H address_string

  设置允许哪些主机连接到 GDS，参数需为 CIDR 格式，仅支持 linux 系统。需要配置多个不同网段时，使用“,”分隔。例如：-H 10.10.0.0/24,10.10.5.0/24。

- -e dir

  设置导入时产生的错误日志存放路径。

  默认值：数据文件目录。

- -E size

  设置导入产生的错误日志的上限值。

取值范围：0<size<1TB，请使用正整数+单位的形式进行取值设置，单位支持 KB、MB 和 GB。

- -S size

  设置导出单个文件大小上限。

  取值范围：1MB<size<100TB，请使用正整数+单位的形式进行取值设置，单位支持 KB、MB 和 GB。如果使用 KB，取值需要大于 1024KB。

- -R size

  设置-l 指定的 GDS 单个日志文件大小上限。

  取值范围：1MB<size<1TB，请使用正整数+单位的形式进行取值设置，单位支持 KB、MB 和 GB。如果使用 KB，取值需要大于 1024KB。

  默认值：16MB。

- -t worker_num

  设置导入导出工作并发线程数目。

  取值范围：0<worker_num≤200，正整数。

  默认值：8。

  推荐值：普通文件导入导出场景取值：CPU 核数*2；管道文件导入导出场景取值：64。

说明

当管道文件导入导出场景并发较大时，该值应不低于业务并发数。

- -s status_file

  设置状态文件，仅支持 linux 系统。

- -D

  后台运行 GDS，仅支持 linux 系统。

- -r

  递归遍历目录（外表目录下的子目录）下文件，仅支持 linux 系统。

- --enable-ssl

  使用 SSL 认证的方式与集群通信。

- --ssl-dir cert_file

  在使用 SSL 认证方式时，指定认证证书的所在路径。

- --debug-level

  设置 GDS 端的 debug 日志级别，以控制 GDS debug 相关的日志输出。

  取值范围：0、1、2。

  - 0：仅打印导入导出相关的文件列表，日志量小，推荐在系统处于正常状态时使用设置。
  - 1：打印日志的完整信息，增加各节点的连接信息、session 转换信息和一些数据统计。推荐仅在故障定位时开启。
  - 2：打印详细的交互日志以及所属状态，输出较大量的 debug 日志信息，以帮助故障定位分析。推荐仅在故障定位时开启。

- --pipe-timeout

  设置 GDS 操作管道文件的等待超时时间。

取值范围：大于 1s。请使用正整数+单位的形式进行取值设置，单位支持 s、m 和 h。如：1 小时可以设置为 3600s、60m 或者 1h。

默认值：1h/60m/3600s

说明

- 该参数的设置是为了避免人为或程序自身问题造成管道文件的一端长时间不读取或者不写入，导致管道另一端的读取或写入操作 hang 住。
- 该参数表示的超时时间不是指 GDS 一个导入导出任务的最长时间，而是 GDS 对管道文件的每一次 read/open/write 的最大超时时间，当超过--pipe-timeout 参数设置时间会向前端报错。

示例

启动一个 GDS 进程，其数据文件存放在“/data” 目录，IP 为 192.168.0.90，监听端口为 5000。

```
gds_ctl start --host 192.168.0.90 -d /data/ -p 5000 -H 10.10.0.1/24 -D
```

启动一批 GDS 进程，其数据文件存放在“/data” 目录，IP 为 192.168.0.90、192.168.0.91、192.168.0.92，监听端口为 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
```

批量关闭位于 192.168.0.90、192.168.0.91、192.168.0.92 节点上，端口是 5000 的 GDS 进程：

```
gds_ctl stop --host 192.168.0.90,192.168.0.91,192.168.0.92 -p 5000
```

批量重启位于 192.168.0.90、192.168.0.91、192.168.0.92 节点上，端口是 5000 的 GDS 进程：

```
gds_ctl restart  --host 192.168.0.90,192.168.0.91,192.168.0.92 -p 5000
```

7.8 gs_sshexkey

背景信息

集群在安装过程中，需要在集群中的节点间执行命令，传送文件等操作。因此，安装前需要确保互信是连通的。GaussDB(DWS)提供了 gs_sshexkey 工具来帮助用户建立互信。

须知

root 用户互信可能会存在安全隐患，因此建议用户在使用完成后，立即删除各主机上 root 用户的互信。

前提条件

- 确保 ssh 服务打开。
- 确保 ssh 端口不会被防火墙关闭。
- 确保 xml 文件中各主机名称和 IP 配置正确。
- 确保所有机器节点间网络畅通。
- 如果为普通用户建立互信，需要提前在各主机创建相同用户并设置密码。
- 如果各主机安装并启动了 SELinux 服务，需要确保/root 和/home 目录安全上下文为默认值（root 目录：system_u:object_r:home_root_t:s0，home 目录：system_u:object_r:admin_home_t:s0）或者关闭掉 SELinux 服务。

  检查系统 SELinux 状态的方法：执行命令 getenforce，如果返回结果是 Enforcing ，说明 SELinux 安装并启用。

  检查目录安全上下文的命令：

  ```
  ls -ldZ  /root | awk '{print $4}'
  ls -ldZ  /home | awk '{print $4}'
  ```

  恢复目录安全上下文命令：

  ```
  restorecon -r -vv /home/
  restorecon -r -vv /root/
  ```

语法

- 建立互信

  ```
  gs_sshexkey -f HOSTFILE [-W PASSWORD] [...] [--skip-hostname-set] [-l LOGFILE]
  [--uuid=uvalue] [--logAction=action] [--logStep=num]
  ```

- 显示帮助信息

  ```
  gs_sshexkey -? | --help
  ```

- 显示版本号信息

  ```
  gs_sshexkey -V | --version
  ```

参数说明

- -f

  主机列表，列出所有需要建立互信主机的 IP。

  说明

  确保 hostfile 文件中只配置正确的主机 IP，不包含其它信息。

- -W, --password=PASSWORD

  待建互信用户的密码。如果不指定该参数则在建立互信过程中需要交互式输入用户密码。如果各个主机的用户密码不一样时则使用多个-W 参数，密码顺序和 IP 地址需要一一对应，交互式情况下则依次输入对应主机的密码。

  说明

  密码不能包含";","'","$"3 个特殊字符。

- -l

  指定日志文件的保存路径。

取值范围：任意存在的可访问的绝对路径。

- --skip-hostname-set

  是否将“-f”参数文件中 IP 与其 hostname 的映射关系写入“/etc/hosts”文件中。默认写入，如果指定该参数则不写入。

- -?, --help

  显示帮助信息。

- -V, --version

  显示版本号信息。

- --uuid=uvalue

  设置日志文件中的追踪标志。

  取值范围：字符串，且由大写或者小写字母、数字、中划线组成，长度为 36 个字符。

- --logAction=action

  设置日志文件中的工具行为标签。

  取值范围：字符串。

- --logStep=num

  设置日志文件中的工具步骤标签。

  取值范围：正整数。

示例

如下是为 root 用户建立互信的示例。

- 用户密码相同情况下，非交互式模式使用以下命令建立互信。

  *{password}需替换*为 root 用户的密码。

```
./gs_sshexkey -f /opt/software/hostfile -W {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.
```

- 用户密码不同情况下，非交互式模式使用以下命令建立互信。

  {password}1 为主机列表中第一台主机的 root 密码，*{password2}*为主机列表中第二台主机的 root 密码。

```
./gs_sshexkey -f /opt/software/hostfile -W {password} -W {password1} -W
{password2}
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.
```

- 用户密码相同情况下，交互式模式使用以下命令建立互信。

```
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.
```

- 用户密码不同情况下，交互式模式使用以下命令建立互信。

```
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.
```

8 修订记录

发布日期	修改说明
2022-11-25	第三次正式发布，适配 DWS 8.1.3.110。
2022-09-10	第二次正式发布。
2022-06-20	第一次正式发布，适配 DWS 8.1.1.100。