数据仓库服务

工具指南

文档版本 02

发布日期 2025-10-11

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

目 录

1 工具简介 1
2 工具下载 3
3 gsql 4
3.1 gsql 概述 4
3.2 下载客户端 18
3.3 使用指导 20
3.3.1 使用 Linux gsql 客户端连接集群 20
3.3.2 使用 Windows gsql 客户端连接集群 22
3.3.3 使用 SSL 进行安全的 TCP/IP 连接 25
3.4 获取帮助 30
3.5 命令参考 31
3.6 元命令参考 35
3.7 常见问题处理 58
4 GDS 63
4.1 安装配置和启动 GDS 63
4.2 停止 GDS 67
4.3 GDS 导入示例 67
4.4 gds 71
4.5 gds_ctl.py 74
4.6 处理错误表 76
5 DSC 79
5.1 前言 79
5.1.1 读者对象 79
5.1.2 文档约定 79
5.1.3 第三方许可 80
5.2 DSC 简介 81
5.2.1 概述 81
5.2.2 运行环境 82
5.3 使用 DSC 85
5.3.1 概述 85
5.3.2 下载并安装 DSC 85
5.3.3 配置 DSC 88

5.3.3.1 DSC 配置 88
5.3.3.2 Teradata SQL 配置 95
5.3.3.3 Teradata Perl 配置 101
5.3.3.4 MySQL 配置 105
5.3.3.5 SQL-Server 配置 108
5.3.3.6 Oracle SQL 配置 108
5.3.3.7 Netezza 配置 116
5.3.4 迁移流程 117
5.3.4.1 前提条件 117
5.3.4.2 准备工作 120
5.3.4.3 执行 DSC 121
5.3.4.4 查看输出文件和日志 123
5.3.4.5 故障处理 123
5.3.5 数据库模式迁移 123
5.3.5.1 迁移参数指导 124
5.3.5.2 Teradata SQL 迁移 126
5.3.5.3 Teradata Perl 迁移 127
5.3.5.4 MySQL SQL 迁移 130
5.3.5.5 Oracle SQL 迁移 131
5.3.5.6 Netezza SQL 迁移 133
5.3.5.7 迁移验证 134
5.3.6 Version 命令迁移 135
5.3.7 Help 命令迁移 135
5.3.8 日志参考 137
5.3.8.1 日志概述 137
5.3.8.2 SQL 迁移日志 138
5.3.8.3 Perl 迁移日志 140
5.4 Teradata 语法迁移 141
5.4.1 支持的关键词和特性 141
5.4.2 约束和限制 145
5.4.3 数据类型 146
5.4.4 函数和操作符 147
5.4.4.1 分析函数 147
5.4.4.2 数学函数 151
5.4.4.3 字符串函数 152
5.4.4.4 日期和时间函数 153
5.4.4.5 比较和列表操作符 158
5.4.4.6 表操作符 160
5.4.4.7 查询优化操作符 160
5.4.4.8 QUALIFY 162
5.4.4.9 ALIAS 164
5.4.4.10 FORMAT 和 CAST 165

5.4.4.11 缩写关键字迁移 167
5.4.4.12 以$开头的对象名称迁移 169
5.4.5 表迁移 172
5.4.5.1 CREATE TABLE 172
5.4.5.2 CHARACTER SET 和 CASESPECIFIC 173
5.4.5.3 VOLATILE 175
5.4.5.4 SET 176
5.4.5.5 MULTISET 176
5.4.5.6 TITLE 176
5.4.5.7 索引 179
5.4.5.8 约束 180
5.4.5.9 COLUMN STORE 181
5.4.5.10 PARTITION 182
5.4.5.11 ANALYZE 188
5.4.5.12 支持指定部分列 188
5.4.6 索引迁移 191
5.4.7 视图迁移 192
5.4.8 COLLECT STATISTICS 197
5.4.9 ACCESS LOCK 198
5.4.10 DBC.COLUMNS 198
5.4.11 DBC.TABLES 201
5.4.12 DBC.INDICES 202
5.4.13 SHOW STATS VALUES SEQUENCED 203
5.4.14 COMMENT 语句 204
5.4.15 数据操作语句（DML） 204
5.4.15.1 INSERT 204
5.4.15.2 SELECT 204
5.4.15.3 UPDATE 212
5.4.15.4 DELETE 213
5.4.15.5 MERGE 215
5.4.15.6 NAMED 215
5.4.15.7 ACTIVITYCOUNT 217
5.4.15.8 TIMESTAMP 218
5.4.16 类型转换和格式化 218
5.4.17 BTEQ 工具命令 224
5.4.18 Teradata 格式 226
5.4.19 系统视图 228
5.5 MySQL 语法迁移 229
5.5.1 支持的关键词和特性 229
5.5.2 数据类型 232
5.5.2.1 数字类型 232
5.5.2.2 日期和时间类型 234

5.5.2.3 字符串类型 236
5.5.2.4 空间数据类型 238
5.5.2.5 大对象类型 239
5.5.2.6 集合类型 240
5.5.2.7 布尔类型 241
5.5.2.8 二进制类型 242
5.5.2.9 JSON 类型 243
5.5.3 函数和表达式 243
5.5.4 表（可选参数、操作） 247
5.5.4.1 ALGORITHM 247
5.5.4.2 ALTER TABLE RENAME 247
5.5.4.3 AUTO_INCREMENT 248
5.5.4.4 AVG_ROW_LENGTH 249
5.5.4.5 BLOCK_SIZE 249
5.5.4.6 CHARSET 250
5.5.4.7 CHECKSUM 250
5.5.4.8 CLUSTERED KEY 251
5.5.4.9 COLLATE 251
5.5.4.10 COMMENT 251
5.5.4.11 CONNECTION 252
5.5.4.12 DEFAULT 252
5.5.4.13 DELAY_KEY_WRITE 256
5.5.4.14 DISTRIBUTE BY 257
5.5.4.15 DIRECTORY 257
5.5.4.16 ENGINE 258
5.5.4.17 FOREIGN_KEY_CHECKS 258
5.5.4.18 IF NOT EXISTS 259
5.5.4.19 INDEX_ALL 259
5.5.4.20 INSERT_METHOD 260
5.5.4.21 KEY_BLOCK_SIZE 260
5.5.4.22 LOCK 261
5.5.4.23 MAX_ROWS 261
5.5.4.24 MIN_ROWS 262
5.5.4.25 PACK_KEYS 262
5.5.4.26 PARTITION BY 263
5.5.4.27 PASSWORD 265
5.5.4.28 ROW_FORMAT 265
5.5.4.29 STATS_AUTO_RECALC 266
5.5.4.30 STATS_PERSISTENT 267
5.5.4.31 STATS_SAMPLE_PAGES 267
5.5.4.32 UNION 268
5.5.4.33 WITH AS 269

5.5.4.34 CHANGE 修改列 269
5.5.4.35 CHECK 约束 270
5.5.4.36 DROP 删除表 271
5.5.4.37 LIKE 表克隆 272
5.5.4.38 MODIFY 修改列 272
5.5.4.39 TRUNCATE 删除表 273
5.5.4.40 ROUNDROBIN 表 273
5.5.4.41 RENAME 重命名表名 274
5.5.4.42 设置与清除列默认值 274
5.5.4.43 字段名重命名 275
5.5.4.44 行列存压缩 276
5.5.4.45 添加与删除列 277
5.5.5 索引 278
5.5.5.1 唯一索引 278
5.5.5.2 普通索引和前缀索引 282
5.5.5.3 HASH 索引 283
5.5.5.4 BTREE 索引 284
5.5.5.5 SPATIAL 空间索引 285
5.5.5.6 FULLTEXT 全文索引 287
5.5.5.7 删除索引 289
5.5.5.8 索引重命名 290
5.5.6 注释 291
5.5.7 数据库 291
5.5.8 数据操作语句（DML） 292
5.5.8.1 INSERT 292
5.5.8.2 UPDATE 295
5.5.8.3 REPLACE 296
5.5.8.4 引号 299
5.5.8.5 INTERVAL 300
5.5.8.6 除法表达式 300
5.5.8.7 GROUP BY 转换 300
5.5.8.8 ROLLUP 301
5.5.9 事务管理和数据库管理 301
5.5.9.1 事务管理 301
5.5.9.2 数据库管理 302
5.6 SQL-Server 语法迁移 304
5.6.1 表迁移 304
5.6.2 数据类型迁移 307
5.7 Oracle 语法迁移 308
5.7.1 模式对象 309
5.7.1.1 表（Oracle） 309
5.7.1.2 临时表 330

5.7.1.3 全局临时表 331
5.7.1.4 索引 331
5.7.1.5 视图 333
5.7.1.6 序列 335
5.7.1.7 PURGE 339
5.7.1.8 数据库关键字 339
5.7.2 COMPRESS 短语 345
5.7.3 Bitmap 索引 345
5.7.4 自定义表空间 346
5.7.5 附加日志数据 347
5.7.6 LONG RAW 348
5.7.7 SYS_GUID 349
5.7.8 DML（Oracle） 349
5.7.9 伪列 363
5.7.10 OUTER JOIN 365
5.7.11 OUTER QUERY (+) 366
5.7.12 CONNECT BY 367
5.7.13 系统函数 369
5.7.13.1 日期函数 369
5.7.13.2 LOB 函数 372
5.7.13.3 字符串函数（Oracle） 376
5.7.13.4 分析函数 380
5.7.13.5 正则表达式函数 382
5.7.14 PL/SQL 388
5.7.15 PL/SQL 集合（使用自定义类型） 401
5.7.16 PL/SQL 包 407
5.7.16.1 包 407
5.7.16.2 包变量 418
5.7.16.3 包拆分 436
5.7.16.4 REF CURSOR 438
5.7.16.5 创建包模式 439
5.7.17 VARRAY 440
5.7.18 授予执行权限 441
5.7.19 包名列表 442
5.7.20 数据类型 443
5.7.21 支持中文字符 444
5.8 Netezza 语法迁移 445
5.8.1 表（Netezza） 445
5.8.2 PROCEDURE（使用 RETURNS） 446
5.8.3 Procedure 450
5.8.4 系统函数（Netezza） 461
5.8.5 算子 462

5.8.6 DML（Netezza）.....463
5.8.7 Unique Index.....465
5.9 DSC 常见问题.....466
5.10 故障处理.....466
5.11 术语表.....475
6 DataCheck.....478
6.1 DataCheck 简介.....478
6.1.1 概述.....478
6.1.2 运行环境.....479
6.2 DataCheck 基本功能.....481
6.3 下载并安装 DataCheck.....482
6.4 配置 DataCheck.....483
6.4.1 dbinfo.properties 配置.....483
6.4.2 check_input.xlsx 配置.....486
6.5 使用 DataCheck.....488
7 服务端工具.....493
7.1 gs_dump.....493
7.2 gs_dumpall.....503
7.3 gs_restore.....508
7.4 gds_check.....514
7.5 gds_install.....517
7.6 gds_uninstall.....518
7.7 gds_ctl.....520

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	高斯数据服务工具GDS（Gauss Data Service），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	DSC（Database Schema Convertor），是一款运行在Linux或Windows操作系统上的命令行工具，用于将Teradata或Oracle数据库中的sql脚本迁移为适用于DWS的sql脚本，便于在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
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）数据，同时不影响用户对数据库的正常访问。
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。
gs_restore	gs_restore是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。
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。
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。
gds_uninstall	gds_uninstall是用于批量卸载GDS的脚本工具。
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。

2 工具下载

操作步骤

步骤1 登录DWS控制台。

步骤2 在左侧导航栏中，单击“管理 > 连接客户端”。

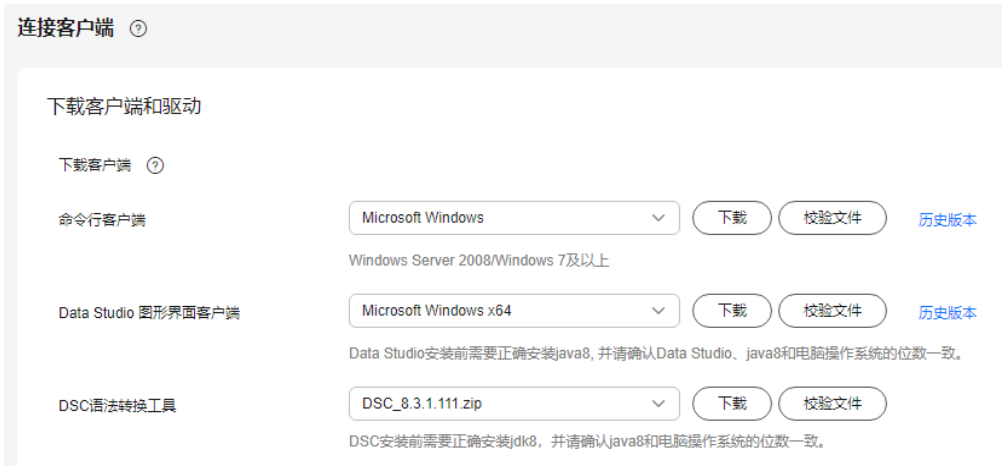
步骤3 在“下载客户端和驱动”区域，请根据计算机的操作系统，选择对应版本的工具进行下载。

此处可以下载以下工具：

- 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。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

3.1 gsql 概述

基本功能

- **连接数据库：** 通过gsql客户端远程连接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**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**。 • 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3-12**），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。
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。

变量	设置方法	变量说明
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。
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。

变量	设置方法	变量说明
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。

变量	设置方法	变量说明
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${}变量引用方式，默认为false。 • true：执行SQL语句生成变量，支持${}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“,”。

变量	设置方法	变量说明
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。 • 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**。
INCOMPLETE_QUERY_ERROR	\set INCOMPLETE_QUERY_ERROR true | false	用于控制在\if \goto \for等流程控制元命令执行前是否检测不完整SQL语句，默认为false。 不完整语句包含“(”、“'”、“"”、“$$”不匹配，以及未以分号结尾，检测到后报错退出。使用示例详见**特殊变量INCOMPLETE_QUERY_ERROR使用示例**。
INCOMPLETE_QUERY_ERROR_CODE	\set INCOMPLETE_QUERY_ERROR_CODE 12	在INCOMPLETE_QUERY_ERROR为true时，检测到不完整语句会报错退出，可通过INCOMPLETE_QUERY_ERROR_CODE设置退出码，默认为-1。 使用示例详见**特殊变量INCOMPLETE_QUERY_ERROR_CODE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)
```

– 特殊变量INCOMPLETE_QUERY_ERROR使用示例：

当INCOMPLETE_QUERY_ERROR为true时，在\if \goto \for等流程控制元命令前检测不完整SQL语句，并报错退出。

说明

主要识别以下类型不完整语句：

- SQL未以分号结尾。
- SQL中括号不匹配。
- SQL中单引号不匹配。
- SQL中双引号不匹配。
- SQL中$$不匹配。

识别方法：

- 未以分号结尾和括号不匹配：在执行\if \goto \for等流程控制元命令前对字符串分析，剔除C语言风格注释（/**/）和任何空白字符串（包括空格、制表符、换页符等，等价于 [\f\n\r\t\v] ）后还有其他字符，均认为有未完整SQL残留，则报错退出。
- 单引号、双引号和$$不匹配：在未匹配时检测到\if、\elif、\else、\endif、\goto、\label、\for、\loop、\exit-for、\end-for元命令，则报错退出。

SQL未以分号结尾的使用示例：

```
\set INCOMPLETE_QUERY_ERROR true
select 1 as id
```



```
\if ${ERROR}
   \echo 'find error'
   \q 12
\endif
```

如上用例，\if元命令前存在未以分号结尾的SQL语句，在执行\if时会报错退出。

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

SQL中括号未匹配的使用示例：

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

如上用例，\if元命令前存在括号未匹配的SQL语句，在执行\if时会报错退出。

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

SQL中单引号未匹配的使用示例：

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

如上用例，\if元命令前存在单引号未匹配的SQL语句，在执行\if时会报错退出。

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

SQL中双引号未匹配的使用示例：

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

如上用例，\if元命令前存在双引号未匹配的SQL语句，在执行\if时会报错退出。

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

SQL中$$未匹配的使用示例：

```
\set INCOMPLETE_QUERY_ERROR true
create or replace function gsql_dollar_quote_test()
returns integer
as
$BODY$
declare
   query text;
```



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

如上用例，\if元命令前存在$$未匹配的SQL语句，在执行\if时会报错退出。

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

– 特殊变量INCOMPLETE_QUERY_ERROR_CODE使用示例：

当INCOMPLETE_QUERY_ERROR为true，可通过INCOMPLETE_QUERY_ERROR_CODE设置检测到不完整语句时的退出码。使用示例如下：

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

如上用例，设置INCOMPLETE_QUERY_ERROR_CODE为20，则\if检测到不完整语句后退出，且退出码为INCOMPLETE_QUERY_ERROR_CODE的值。

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

- SQL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， postgres=> \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 下载客户端

DWS提供了与集群版本配套的客户端工具包，用户可以在DWS管理控制台下载客户端工具包。

客户端工具包包含以下内容：

- **数据库连接工具Linux gsql和测试样例数据的脚本**

  Linux gsql是一款运行在Linux环境上的命令行客户端，用于连接DWS集群中的数据库。

  测试样例数据的脚本是执行入门示例时用的。
- **Windows版本gsql**

  Windows gsql是一款运行在Windows环境上的命令行客户端，用于连接DWS集群中的数据库。

  说明

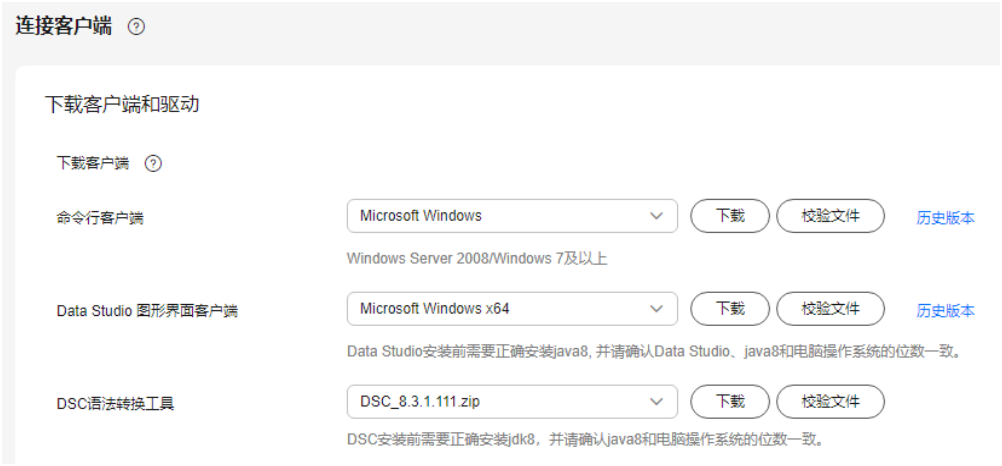
  仅8.1.3.101及以上集群版本支持在console控制台下载。

下载客户端

步骤1 登录DWS控制台，在左侧导航栏中，单击“连接管理”。

步骤2 在“gsql命令行客户端”的下拉列表中，选择对应版本的DWS客户端。

图 3-1 下载客户端



步骤3 单击“下载”可以下载与8.1.x集群版本匹配的gsql。单击“历史版本”可根据集群版本下载相应版本的gsql。

- 推荐下载使用与集群版本匹配的gsql工具，即8.1.0及以上版本集群使用8.1.x版本gsql、8.2.0及以上版本集群使用8.2.x版。
- 表3-6列出了下载的Linux gsql工具包中的文件和文件夹。

表 3-6 Linux gsql 工具包目录及文件说明

文件或文件夹	说明
bin	该文件夹中包含了gsql在Linux中的可执行文件。其中包含了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。
gds	该文件夹中包括了GDS数据服务工具的相关文件，GDS工具用于并行数据加载，可将存储在普通文件系统中的数据文件导入到DWS数据库中。
lib	该文件夹中包括执行gsql所需依赖的lib库。
sample	该文件夹中包含了以下目录或文件： – setup.sh：在使用gsql导入样例数据前所需执行的配置AK/SK访问密钥的脚本文件。 – tpcds_load_data_from_obs.sql：使用gsql客户端导入TPC-DS样例数据的脚本文件。 – query_sql目录：查询TPC-DS样例数据的脚本文件。
gsql_env.sh	在运行gsql前，配置环境变量的脚本文件。

- 表3-7列出了下载的Windows gsql工具包中的文件和文件夹。

  表 3-7 Windows gsql 工具包目录及文件说明

文件或文件夹	说明
x64	该文件夹中包含了64位Windows gsql执行二进制和动态库。
x86	该文件夹中包含了32位Windows gsql执行二进制和动态库。

  说明

  - 在“集群管理”页面的集群列表中，单击指定集群的名称，再选择“集群详情”页签，可查看集群版本。

----结束

3.3 使用指导

3.3.1 使用 Linux gsql 客户端连接集群

用户在创建好数据仓库集群，开始使用集群数据库之前，需要使用数据库SQL客户端连接到数据库。DWS提供了与集群版本配套的Linux gsql命令行客户端工具，您可以使用Linux gsql客户端通过集群的公网地址或者内网地址访问集群。

它的运行环境是Linux操作系统，在使用Linux gsql客户端远程连接DWS集群之前，需要准备一个Linux主机用于安装和运行Linux gsql客户端。如果通过公网地址访问集群，也可以将Linux gsql客户端安装在用户自己的Linux主机上，但是该Linux主机必须具有公网地址。若DWS集群没有配置公网IP，为方便起见，推荐您创建一台Linux弹性云服务器（简称ECS），详情可参见**（可选）准备ECS作为gsql客户端主机**。

（可选）准备 ECS 作为 gsql 客户端主机

创建弹性云服务器的操作步骤，请参见《弹性云服务器用户指南》中的“快速入门 > 创建弹性云服务器”章节。

创建的弹性云服务器需要满足如下要求：

- 弹性云服务器需要与DWS集群具有相同的区域、可用区。
- 如果使用DWS提供的gsql命令行客户端连接DWS集群，弹性云服务器的镜像必须满足如下要求：

  镜像的操作系统必须是gsql客户端所支持的下列Linux操作系统：

  - “Redhat x86_64”客户端工具支持在以下系统中使用：
    - RHEL 6.4~7.6。
    - CentOS 6.4~7.4。
    - EulerOS 2.3。

  - “SUSE x86_64”客户端工具支持在以下系统中使用：
    - SLES 11.1~11.4。
    - SLES 12.0~12.3。
- 如果客户端通过内网地址访问集群，请确保创建的弹性云服务器与DWS集群在同一虚拟私有云里。

  虚拟私有云相关操作请参见《虚拟私有云用户指南》中“虚拟私有云和子网”。
- 如果客户端通过公网地址访问集群，请确保创建的弹性云服务器和DWS集群都要有弹性IP。

  创建弹性云服务器时，参数“弹性IP”需设置为“自动分配”或“使用已有”。
- 弹性云服务器对应的安全组规则需要确保能与DWS集群提供服务的端口网络互通。

  安全组相关操作请参见《虚拟私有云用户指南》中“安全组”章节。

  请确认弹性云服务器的安全组中存在符合如下要求的规则，如果不存在，请在弹性云服务器的安全组中添加相应的规则：
  - 方向：出方向
  - 协议：必须包含TCP，例如TCP、全部。
  - 端口：需要包含DWS集群提供服务的数据库端口，例如，设置为“1-65535”或者具体的DWS数据库端口。
  - 目的地址：设置的IP地址需要包含所要连接的DWS集群的连接地址。其中0.0.0.0/0表示任意地址。
- DWS集群的安全组规则需要确保DWS能接收来自客户端的网络访问。

  请确认DWS集群的安全组中存在符合如下要求的规则，如果不存在，请在DWS集群的安全组中添加相应的规则。
  - 方向：入方向
  - 协议：必须包含TCP，例如TCP、全部。
  - 端口：设置为DWS集群提供服务的数据库端口，例如“8000”。
  - 源地址：设置的IP地址需要包含DWS客户端主机的IP地址，例如“192.168.0.10/32”。

下载 Linux gsql 客户端并连接集群

步骤1 请参见**下载客户端**下载Linux gsql客户端，并使用SSH文件传输工具（例如WinSCP工具），将客户端工具上传到一个待安装Linux gsql的Linux主机上。

推荐下载使用与集群版本匹配的gsql工具，即8.1.0及以上版本集群使用8.1.x版本gsql、8.2.0及以上版本集群使用8.2.x版。若下载8.2.x版本gsql工具，需将dws_client_8.1.x_redhat_x64.zip替换为dws_client_8.2.x_redhat_x64.zip。此处仅以dws_client_8.1.x_redhat_x64.zip作为示例。

执行上传Linux gsql操作的用户需要对客户端主机的目标存放目录有完全控制权限。

步骤2 使用SSH会话工具，远程管理客户端主机。

弹性云服务器的登录方法请参见《弹性云服务器用户指南》中的“实例 > 登录Linux弹性云服务器 > SSH密码方式登录”章节。

步骤3 （可选）如果要使用SSL方式连接集群，请参考**使用SSL进行安全的TCP/IP连接**章节，在客户端主机配置SSL认证相关的参数。

说明

SSL连接方式的安全性高于非SSL方式，建议在客户端使用SSL连接方式。

步骤4 执行以下命令解压客户端工具。

```
cd <客户端存放路径>
unzip dws_client_8.1.x_redhat_x64.zip
```

其中：

- <客户端存放路径>：请替换为实际的客户端存放路径。
- dws_client_*8.1.x*_redhat_x64.zip：这是“RedHat x64”对应的客户端工具包名称，请替换为实际下载的包名。

步骤5 执行以下命令配置客户端。

```
source gsql_env.sh
```

提示以下信息表示客户端已配置成功

```
All things done.
```

步骤6 执行以下命令，使用gsql客户端连接DWS集群中的数据库。

```
gsql -d <数据库名称> -h <集群地址> -U <数据库用户> -p <数据库端口> -W <集群密码> -r
```

参数说明如下：

- “数据库名称”：输入所要连接的数据库名称。首次使用客户端连接集群时，请指定为集群的默认数据库“gaussdb”。
- “集群地址”：请参见《数据仓库服务用户指南》中"连接集群>获取集群连接地址"章节进行获取。如果通过公网地址连接，请指定为集群“公网访问地址”，如果通过内网地址连接，请指定为集群“内网访问地址”。
- “数据库用户”：输入集群数据库的用户名。首次使用客户端连接集群时，请指定为创建集群时设置的默认管理员用户，例如“dbadmin”。
- “数据库端口”：输入创建集群时设置的“数据库端口”。

例如，执行以下命令连接DWS集群的默认数据库gaussdb：

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

显示如下信息表示gsql工具已经连接成功：

```
gaussdb=>
```

----结束

gsql 命令参考

有关gsql的命令参考和更多信息，请参见**元命令参考**。

3.3.2 使用 Windows gsql 客户端连接集群

用户在创建好数据仓库集群，开始使用集群数据库之前，需要使用数据库SQL客户端连接到数据库。DWS提供了与集群版本配套的Windows gsql命令行客户端工具，您可以使用Windows gsql客户端通过集群的公网地址或者内网地址访问集群。

操作步骤

步骤1 在计算机本地Windows操作系统服务器（Windows cmd）中安装和运行gsql客户端。Windows操作系统支持Windows Server 2008/Windows 7及以上。

步骤2 请参见**下载客户端**下载Windows gsql客户端，并将压缩包解压到本地文件夹中。

步骤3 在本地主机单击“开始”并搜索“cmd”，用管理员身份运行或单击快捷键“Win+R”打开Windows cmd窗口。

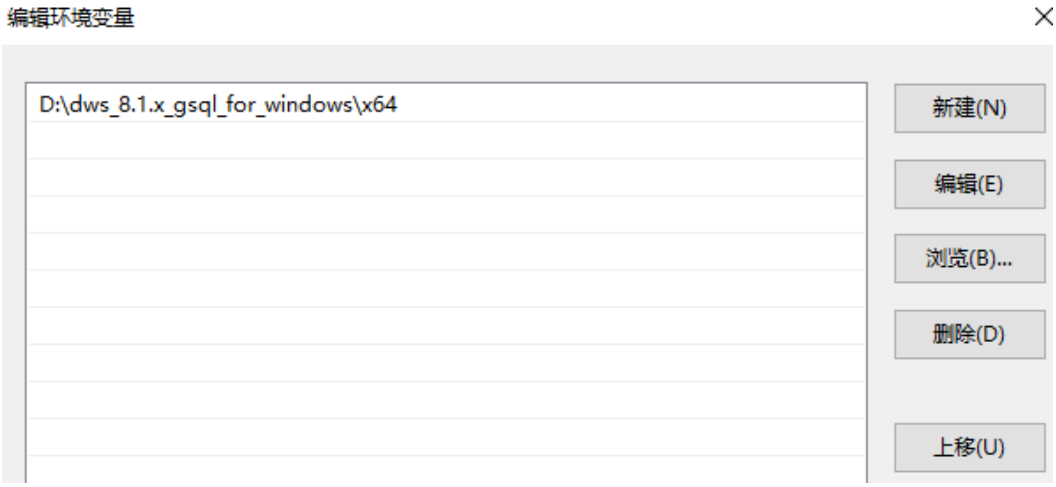
步骤4 设置环境变量，32位选择x86文件夹；64位选择x64文件夹。

方式一：命令行设置环境变量，打开Windows cmd窗口，执行set path=<window gsql>;%path%，其中<window gsql>为上一步骤解压Windows gsql客户端的文件夹路径。例如：

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

方式二：在控制面板中选择“系统->高级系统设置->高级->环境变量”，在系统环境变量Path中增加gsql路径。例如：

图 3-2 设置 Windows 环境变量



步骤5 （可选）如果要使用SSL方式连接集群，请参考**使用SSL进行安全的TCP/IP连接**章节，在客户端主机配置SSL认证相关的参数。

说明

SSL连接方式的安全性高于非SSL方式，建议在客户端使用SSL连接方式。

步骤6 在Windows cmd窗口执行以下命令，使用gsql客户端连接DWS集群中的数据库。

```
gsql -d <数据库名称> -h <集群地址> -U <数据库用户> -p <数据库端口> -W <集群密码> -r
```

参数说明如下：

- “数据库名称”：输入所要连接的数据库名称。首次使用客户端连接集群时，请指定为集群的默认数据库“gaussdb”。
- “集群地址”：请参见《数据仓库服务用户指南》中"连接集群>获取集群连接地址"章节进行获取。如果通过公网地址连接，请指定为集群“公网访问域名”，如果通过内网地址连接，请指定为集群“内网访问域名”。
- “数据库用户”：输入集群数据库的用户名。首次使用客户端连接集群时，请指定为创建集群时设置的默认管理员用户，例如“dbadmin”。

- “数据库端口”：输入创建集群时设置的“数据库端口”。

例如，执行以下命令连接DWS集群的默认数据库gaussdb：

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

显示如下信息表示gsql工具已经连接成功：

```
gaussdb=>
```

----结束

注意事项

1. Windows cmd默认的字符集是GBK，所以Windows gsql默认的client_encoding为GBK，部分UTF-8编码的字符无法在Windows gsql中显示。

   建议：-f执行的文件使用UTF-8编码，并设置默认的编码格式为UTF-8（set client_encoding=’utf-8’;）

2. Windows gsql中的路径需要使用‘/’作为分隔符，否则会报错。因为在元命令中‘\’是作为元命令开始的标志，在一般的单引号中，‘\’起转义作用。

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

3. Windows gsql使用\!元命令执行系统命令时，需要使用系统命令要求的路径分隔符，一般是‘\’。

   ```
   gaussdb=> \! type D:/test.sql
   命令语法不正确。
   gaussdb=> \! type D:\test.sql
   select 1 as id;
   ```

4. Windows gsql不支持元命令\parallel。

   ```
   gaussdb=> \parallel
   ERROR: "\parallel" is not supported in Windows.
   ```

5. Linux shell中可以使用单引号和双引号作为字符串边界，但在Windows必须使用双引号作为字符串边界。

   ```
   gsql -h 192.168.233.189 -p 8109 -d postgres -U odbcuser -W password -c "select 1 as id"
    id
   ----
    1
   (1 row)
   ```

   使用单引号时报错，并忽略输入。

   ```
   gsql -h 192.168.233.189 -p 8109 -d postgres -U odbcuser -W password -c 'select 1 as id'
   gsql: warning: extra command-line argument "1" ignored
   gsql: warning: extra command-line argument "as" ignored
   gsql: warning: extra command-line argument "id'" ignored
   ERROR:  unterminated quoted string at or near "'select"
   LINE 1: 'select
   ```

6. Windows gsql在建立连接之后长时间未使用，连接session超时，会出现SSL报错，需要重新登录。报错如下：

   ```
   SSL SYSCALL error: Software caused connection abort (0x00002745/10053), remote datanode <NULL>, error: Result too large
   ```

7. Windows下Ctrl+C退出gsql。在当前行输入SQL语句时，若捕获到Ctrl+C信号后，无法将状态调整到重新输入的状态，会按照当前没有输入处理，将直接退出gsql。

   在输入as后执行Ctrl+C，输出\q后退出gsql。

   ```
   gaussdb=> select 1
   gaussdb=> as \q
   ```


8. Windows gsql不支持连接字符集为LATIN1的数据库，报错信息为：
   ```
   gsql: FATAL: conversion between GBK and LATIN1 is not supported
   ```
9. gsqlrc.conf文件的位置。

   默认的gsqlrc路径为%APPDATA%/postgresql/gsqlrc.conf，也可通过PSQLRC变量设置。
   ```
   set PSQLRC=C:\Users\xx\Desktop\dws_8.1.x_gsql_for_windows\x64\gsqlrc.conf
   ```

gsql 命令参考

有关gsql的命令参考和更多信息，请参见**元命令参考**。

3.3.3 使用 SSL 进行安全的 TCP/IP 连接

DWS支持SSL标准协议，SSL协议是安全性更高的协议标准，它们加入了数字签名和数字证书来实现客户端和服务器的双向身份验证，保证了通信双方更加安全的数据传输。为支持SSL连接方式，DWS已经从CA认证中心申请到正式的服务器、客户端的证书和密钥（假设服务器的私钥为server.key，证书为server.crt，客户端的私钥为client.key，证书为client.crt，CA根证书名称为cacert.pem）。

SSL连接方式的安全性高于普通模式，集群默认开启SSL功能允许来自客户端的SSL连接或非SSL连接，从安全性考虑，建议用户在客户端使用SSL连接方式。并且DWS服务器端的证书、私钥以及根证书已经默认配置完成。如果要强制使用SSL连接，需要在集群“安全设置”页面开启“服务器端是否强制使用SSL连接”，操作详情可参见**设置SSL连接**，客户端和服务器端SSL连接参数组合情况可请见**客户端和服务器端SSL连接参数组合情况**。

客户端或JDBC/ODBC应用程序使用SSL连接方式，用户必须在客户端或应用程序代码中配置相关的SSL连接参数。DWS管理控制台提供了客户端所需的SSL证书，该SSL证书包含了客户端所需的默认证书、私钥、根证书以及私钥密码加密文件。请将该SSL证书下载到客户端所在的主机上，然后在客户端中指定证书所在的路径，操作详情请参见**在gsql客户端配置SSL认证相关的数字证书参数**，SSL认证及客户端参数介绍可参见**SSL认证方式及客户端参数介绍**。

说明

使用默认的证书可能存在安全风险，为了提高系统安全性，强烈建议用户定期更换证书以避免被破解的风险。如果需要更换证书，请联系。

设置 SSL 连接

前提条件

- 修改安全配置参数并保存后，生效可能需要重启集群，否则将导致集群暂时不可用。
- 修改集群安全配置必须同时满足以下两个条件：
  - 集群状态为“可用”、“待重启”或“非均衡”。
  - 任务信息不能处于“创建快照中”、“节点扩容”、“配置中”或“重启中”。

操作步骤

步骤1 登录DWS控制台。

步骤2 在左侧导航树中，单击“专属集群 > 集群列表”。

步骤3 在集群列表中，单击指定集群的名称，然后单击“安全设置”。

默认显示“配置状态”为“已同步”，表示页面显示的是数据库当前最新结果。

步骤4 在“SSL连接”区域中，单击“服务器端是否强制使用SSL连接”的设置开关进行设置，建议开启。

：开启，设置参数**require_ssl**=1，表示服务器端强制要求SSL连接。

 ：关闭，设置参数**require_ssl**=0，表示服务器端对是否通过SSL连接不作强制要求，默认为关闭。设置require_ssl参数详情请参见**require_ssl（服务器）**。

说明

- 如果使用DWS提供的gsql客户端或ODBC驱动，DWS支持的SSL协议为TLSv1.2。
- 如果使用DWS提供的JDBC驱动，支持的SSL协议有SSLv3、TLSv1、TLSv1.1、TLSv1.2。客户端与数据库之间实际使用何种SSL协议，依赖客户端使用的JDK（Java Development Kit）版本，一般JDK支持多个SSL协议。

步骤5 单击“应用”。

系统将自动应用保存SSL连接设置，在“安全设置”页面，“配置状态”显示“应用中”。当“配置状态”显示为“已同步”，表示配置已保存生效。

----结束

在 gsql 客户端配置 SSL 认证相关的数字证书参数

DWS在集群部署完成后，默认已开启SSL认证模式。服务器端证书，私钥以及根证书已经默认配置完成。用户需要配置客户端的相关参数。

步骤1 登录DWS控制台，在左侧导航栏中，进入“连接客户端”页面。

步骤2 在“下载驱动程序”区域，单击“下载SSL证书”进行下载。

步骤3 使用文件传输工具（例如WinSCP工具）将SSL证书上传到客户端主机。

例如，将下载的证书“dws_ssl_cert.zip”存放到“/home/dbadmin/dws_ssl/”目录下。

步骤4 使用SSH远程连接工具（例如PuTTY）登录gsql客户端主机，然后执行以下命令进入SSL证书的存放目录，并解压SSL证书：

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

步骤5 在gsql客户端主机上，执行export命令，配置SSL认证相关的数字证书参数。

SSL认证有两种认证方式：双向认证和单向认证。认证方式不同用户所需配置的客户端环境变量也不同，详细介绍请参见**SSL认证方式及客户端参数介绍**。

双向认证需配置如下参数：

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

单向认证需要配置如下参数：

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


须知

- 从安全性考虑，建议使用双向认证方式。
- 配置客户端环境变量，必须包含文件的绝对路径。

步骤6 修改客户端密钥的权限。

客户端根证书、密钥、证书以及密钥密码加密文件需保证权限为600。如果权限不满足要求，则客户端无法以SSL方式连接到集群。

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

----结束

SSL 认证方式及客户端参数介绍

SSL认证有两种认证方式，如表3-8所示。从安全性考虑，建议使用双向认证方式。

表 3-8 认证方式

认证方式	含义	配置客户端环境变量	维护建议
双向认证（推荐）	客户端验证服务器证书的有效性，同时服务器端也要验证客户端证书的有效性，只有认证成功，连接才能建立。	设置如下环境变量： • PGSSLCERT • PGSSLKEY • PGSSLROOTCERT • PGSSLMODE	该方式应用于安全性要求较高的场景。使用此方式时，建议设置客户端的PGSSLMODE变量为verify-ca。确保了网络数据的安全性。
单向认证	客户端只验证服务器证书的有效性，而服务器端不验证客户端证书的有效性。服务器加载证书信息并发送给客户端，客户端使用根证书来验证服务器端证书的有效性。	设置如下环境变量： • PGSSLROOTCERT • PGSSLMODE	为防止基于TCP链接的安全攻击，建议使用SSL证书认证功能。除配置客户端根证书外，建议客户端使用PGSSLMODE变量为verify-ca方式连接。

在客户端配置SSL认证相关的环境变量，详细信息请参见表3-9。

说明

客户端环境变量的路径以“*/home/dbadmin*/dws_ssl/”为例，在实际操作中请使用实际路径进行替换。

表 3-9 客户端参数

环境变量	描述	取值说明
PGSSLCERT	指定客户端证书文件，包含客户端的公钥。客户端证书用于表明客户端身份的合法性，公钥将发送给对端用来对数据进行加密。	必须包含文件的绝对路径，如： export PGSSLCERT='*/home/dbadmin/dws_ssl/sslcert/client.crt*' **默认值**：空
PGSSLKEY	指定客户端私钥文件，用于数字签名和对公钥加密的数据进行解密。	必须包含文件的绝对路径，如： export PGSSLKEY='*/home/dbadmin/dws_ssl/sslcert/client.key*' **默认值**：空
PGSSLMODE	设置是否和服务器进行SSL连接协商，以及指定SSL连接的优先级。	**取值及含义：** ● disable：只尝试非SSL连接。 ● allow：首先尝试非SSL连接，如果连接失败，再尝试SSL连接。 ● prefer：首先尝试SSL连接，如果连接失败，将尝试非SSL连接。 ● require：只尝试SSL连接。如果存在CA文件，则按设置成verify-ca的方式验证。 ● verify-ca：只尝试SSL连接，并且验证服务器是否具有由可信任的证书机构签发的证书。 ● verify-full：DWS不支持此模式。 **默认值：**prefer **说明** 若集群外访问客户端时，部分节点出现报错：ssl SYSCALL error。则可执行export PGSSLMODE="allow"或export PGSSLMODE="prefer"。
PGSSLROOTCERT	指定为客户端颁发证书的根证书文件，根证书用于验证服务器证书的有效性。	必须包含文件的绝对路径，如： export PGSSLROOTCERT='*/home/dbadmin/dws_ssl/sslcert/certca.pem*' **默认值：**空
PGSSLCRL	指定证书吊销列表文件，用于验证服务器证书是否在废弃证书列表中，如果在，则服务器证书将会被视为无效证书。	必须包含文件的绝对路径，如： export PGSSLCRL='*/home/dbadmin/dws_ssl/sslcert/sslcrl-file.crl*' **默认值：**空

客户端和服务器端 SSL 连接参数组合情况

客户端最终是否使用SSL加密连接方式、是否验证服务器证书，取决于客户端参数sslmode与服务器端（即DWS集群侧）参数ssl、require_ssl。参数说明如下：

- **ssl（服务器）**

  ssl参数表示是否开启SSL功能。on表示开启，off表示关闭。

  - 默认为on，不支持在DWS管理控制台上设置。

- **require_ssl（服务器）**

  require_ssl参数是设置服务器端是否强制要求SSL连接，该参数只有当ssl为on时才有效。on表示服务器端强制要求SSL连接。off表示服务器端对是否通过SSL连接不作强制要求。

  - 默认为off。require_ssl参数可通过DWS管理控制台上集群的“安全设置”页面中的“服务器端是否强制使用SSL连接”进行设置。

- **sslmode（客户端）**

  可在SQL客户端工具中进行设置。

  - 在gsql命令行客户端中，为“PGSSLMODE”参数。
  - 在Data Studio客户端中，为“SSL模式”参数。

客户端参数sslmode与服务器端参数ssl、require_ssl配置组合结果如下：

表 3-10 客户端与服务器端 SSL 参数组合结果

ssl（服务器）	sslmode（客户端）	require_ssl（服务器）	结果
on	disable	on	由于服务器端要求使用SSL，但客户端针对该连接禁用了SSL，因此无法建立连接。
	disable	off	连接未加密。
	allow	on	连接经过加密。
	allow	off	连接未加密。
	prefer	on	连接经过加密。
	prefer	off	连接经过加密。
	require	on	连接经过加密。
	require	off	连接经过加密。
	verify-ca	on	连接经过加密，且验证了服务器证书。
	verify-ca	off	连接经过加密，且验证了服务器证书。
off	disable	on	连接未加密。
	disable	off	连接未加密。
	allow	on	连接未加密。
	allow	off	连接未加密。
	prefer	on	连接未加密。
	prefer	off	连接未加密。

ssl（服务器）	sslmode（客户端）	require_ssl（服务器）	结果
	require	on	由于客户端要求使用SSL，但服务器端禁用了SSL，因此无法建立连接。
	require	off	由于客户端要求使用SSL，但服务器端禁用了SSL，因此无法建立连接。
	verify-ca	on	由于客户端要求使用SSL，但服务器端禁用了SSL，因此无法建立连接。
	verify-ca	off	由于客户端要求使用SSL，但服务器端禁用了SSL，因此无法建立连接。

3.4 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3-11。

表 3-11 使用 gsql 联机帮助

描述	示例
查看版权信息	\copyright

描述	示例
查看DWS支持的SQL语句的帮助	查看DWS支持的SQL语句的帮助 例如，查看DWS支持的所有SQL语句： \h Available help: ABORT ALTER DATABASE 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

3.5 命令参考

详细的gsql参数请参见**表3-12**、**表3-13**、**表3-14**和**表3-15**。

表 3-12 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。

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

表 3-13 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。	-

参数	参数说明	取值范围
-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。	-
-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 3-14 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。

参数	参数说明	取值范围
-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 3-15 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连，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于身份验证。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** 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。该密码最大长度为999字节，受限于GUC参数password max length的最大值。	符合密码复杂度要求。

3.6 元命令参考

介绍使用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"A weird"" 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命令可以自由地在一行里面混合。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3-16、表3-17、表3-18、表3-19、表3-21、表3-23、表3-24、表3-25和表3-27。

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 3-16 一般的元命令

参数	参数说明	取值范围
\copyright	显示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。

参数	参数说明	取值范围
\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。 ● 不支持在\parallel中使用DISCARD命令。 ● 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。 ● \parallel不支持对VOLATILE/GLOBAL全局临时表进行操作。	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。
\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 3-17 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 3-18 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gsql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。
\qecho [STRING]	把字符串写到查询结果输出流里。

说明

表3-19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 3-19 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`

参数	参数说明	取值范围	示例
\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 *
\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

参数	参数说明	取值范围	示例
\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
\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

参数	参数说明	取值范围	示例
\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+] [PATTERN]	列出可用的程序语言。	如果指定了PATTERN，只列出名字匹配PATTERN的语言。	列出可用的程序语言。 \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3-20**。	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 *
\dRp[+] [PATTERN]	列出所有的发布。该元命令仅8.2.0.100及以上集群版本支持。	如果指定了PATTERN，只列出名字匹配PATTERN的发布。	列出所有的发布。 \dRp

参数	参数说明	取值范围	示例
\dRs[+] [PATTERN]	列出所有的订阅。该元命令仅8.2.0.100及以上集群版本支持。	如果指定了PATTERN，只列出名字匹配PATTERN的订阅。	列出所有的订阅。 \dRs
\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

参数	参数说明	取值范围	示例

\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 "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 3-20 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。

参数	参数说明
T	TEMPORARY：允许创建临时表。
A	ANALYZE|ANALYSE：允许分析表。
L	ALTER：允许修改表、模式或函数。
P	DROP：允许删除表、模式或函数。
v	VACUUM：允许对表执行VACUUM。
arwdDxtA, vLP	ALL PRIVILEGES：一次性给指定用户/角色赋予所有可赋予的权限。 vLP权限组为8.1.3及以上集群版本中新增表级权限ALTER/DROP/VACUUM，新增schema权限ALTER/DROP。
*	给前面权限的授权选项。

表 3-21 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 3-22**。
\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 3-22 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。	-

选项	选项说明	取值范围
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。	-

选项	选项说明	取值范围
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 3-23 连接元命令

参数	参数说明	取值范围
\[connect] [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。该密码最大长度为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。
\conninfo	输出当前连接的数据库的信息。	-

表 3-24 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 3-25 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3-26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 3-26 \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

名称	命令说明	取值范围
\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 ● 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	retry_times取值范围为：5-10

名称	命令说明	取值范围
	● 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 3-27 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DWS大对象及其说明。

表 3-28 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。

参数	参数说明	取值范围
\for \loop \exit-for \end-for	这组元命令可实现循环： • 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 is treated as a string, and the input without quote is 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 offset...
                             ^
```

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。

3.7 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3-29。

  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 3-29 查看会话连接数

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终止，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 GDS

4.1 安装配置和启动 GDS

操作场景

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

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，DWS也会进行两端的版本一致性检测，不一致时会在屏幕上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 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2 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.x.x_suse_x64.zip”上传至上一步所创建的目录中。

步骤3 （可选）如果使用SSL加密传输，请一并上传SSL证书至**步骤1**所创建的目录下。

步骤4 在工具包所在目录下，解压工具包。

```
cd /opt/bin/dws
unzip dws_client_8.x.x_suse_x64.zip
```

步骤5 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6 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7 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8**，直接执行**步骤9**。

若当前集群版本为8.1.x版本，则正常执行以下步骤。

步骤8 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9 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3**中SSL证书已上传至/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。默认值为：8。取值范围为0<worker_num<=200，正整数。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3**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4-1**。

  ```
  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 200. If
without setting, the default value is 8.
 -s status_file    Enable GDS status report.
 -D                Run the GDS as a daemon process.
 -r                Read the working directory recursively.
 -h                Display usage.
```

----结束

gds.conf 参数说明

表 4-1 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。选择true时会递归读取location指定的目录层级下所有的同名文件。	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。

属性	说明	取值范围
parallel	导入工作线程并发数目。	取值范围：0~200，正整数。 默认值：8。

4.2 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

4.3 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**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**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
```

4.4 gds

背景信息

gds可以为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。

  说明

  如果当前目录下GDS进程的日志数目超过--log-filecount时，会触发旧日志的回收。为了保存大量日志，建议每个GDS设置单独目录，并且调高启动参数--log-filecount。

- -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，会递归读取location指定的目录层级下所有的同名文件，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

- --log-filecount

  设置保留的日志文件最大个数，当日志文件个数超过该参数值，按照文件创建时间，保留最近创建的日志文件。

  取值范围：5<=log-filecount<=1024，正整数

  默认值：50

- --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显示指定监听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和密钥文件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。

4.5 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
```

4.6 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4-2所示。

  表 4-2 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 4-3 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“|”被转义为第二个字段的字段值，导致第三个字段值缺失。 `BE|Belgium|1`	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”。即在导入过程中，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。

错误信息	原因	解决办法
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)。

5 DSC

5.1 前言

5.1.1 读者对象

本手册适用于如下使用DSC的用户：

- 数据库迁移工程师
- 数据库管理员
- 技术支持工程师

DSC用户需了解以下概念：

- 数据库迁移基本概念和策略
- Teradata/MySQL(ADB For MySQL)
- DWS

5.1.2 文档约定

本节描述了本手册的内容、符号、和命令约定。

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
	用于警示紧急的危险情形，若不避免，将会导致人员死亡或严重的人身伤害。
	用于警示潜在的危险情形，若不避免，可能会导致人员死亡或严重的人身伤害。

符号	说明
	用于警示潜在的危险情形，若不避免，可能会导致中度或轻微的人身伤害。
	用于传递设备或环境安全警示信息，若不避免，可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “注意”不涉及人身伤害。
	用于突出重要/关键信息、最佳实践和小窍门等。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害。

命令行格式约定

本手册中可能出现下列命令行格式约定，它们所代表的含义如下：

表 5-1 命令行格式列表

格式	说明
粗体	命令行关键字（命令中保持不变、必须照输的部分）采用加粗字体表示。
斜体	命令行参数，路径，文件或文件夹采用斜体表示。
[ ]	表示用“[ ]”括起来的部分（关键词和参数）在命令配置时是可选的。
{ x | y | ... }	表示用“{}”分组选项，各选项之间以“|”分隔。从两个或多个选项中选取一个。
[x | y | ...]	表示用“[ ]”分组选项，各选项之间以“|”分隔。从两个或多个选项中选取一个或者不选。
{ x | y | ... }*	表示用“{}”分组选项，各选项之间以“|”分隔。从两个或多个选项中选取多个，最少选取一个，最多选取所有选项。
[x | y | ...]*	表示用“[ ]”分组选项，各选项之间以“|”分隔。从两个或多个选项中选取多个或者不选。
&<1-n>	表示&符号前的内容可重复1到n次。
#	表示注释。

5.1.3 第三方许可

本节包含适用于该工具的第三方许可。

- ANTLR v4.9.3

- Apache Commons IO 2.11
- Apache Commons CLI 1.5
- Apache Log4j 2.17.2
- JSON.org json 20220320
- postgresql 42.4.1
- sql-formatter 2.0.3

5.2 DSC 简介

5.2.1 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。

⚠ 注意

DSC内置语法解析器只能识别标准的SQL语法，输入脚本中如包含了其他语言的文本，则会导致语法解析异常，请在使用DSC工具前对输入脚本进行清洗。

迁移对象

DSC支持迁移Teradata、MySQL、SQL-Server、Oracle、Netezza、GreenPlum等数据库，详见表5-2，具体支持迁移的数据库对象有：

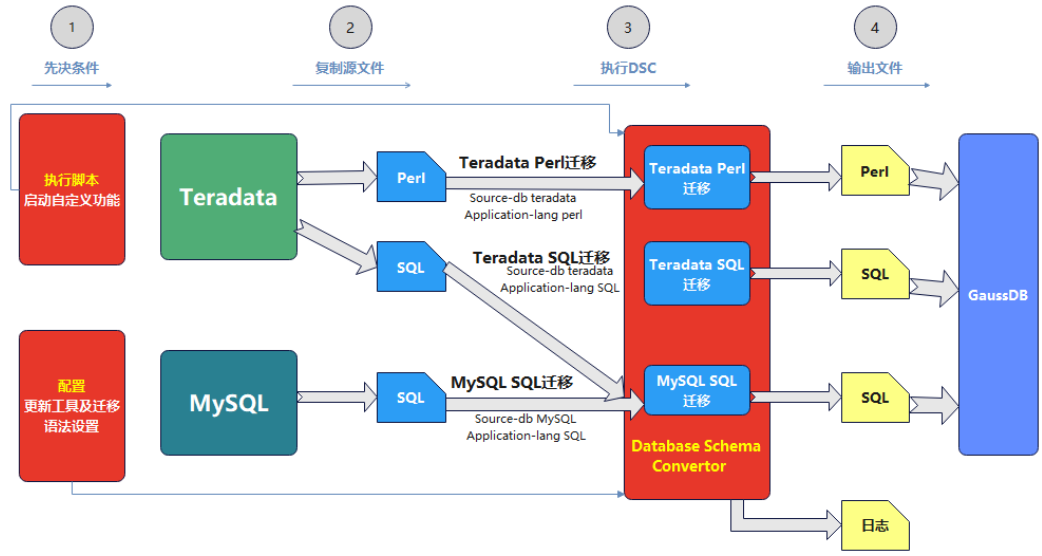
- DDL语句：模式、表、视图、存储过程、自定义函数
- DML语句：select、update、delete、insert、truncate

迁移流程

DSC迁移sql脚本流程如下：

1. 从源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. 执行DSC命令进行语法迁移，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 5-1 DSC 处理流程



5.2.2 运行环境

支持的数据库

DSC支持的源数据库如所示。

表 5-2 支持的源数据库

数据库名称	数据库版本
Teradata	17.20
MySQL	8.0
Oracle	11g Release 2，12c Release 1
AnalyticDB For MySQL	-
BigQuery	-
Postgres	9.2.x，11.x
Redshift	-
Greenplum	7.0.0
Hologres	3.1.17
Netezza	-
Hive	-
其他源：Doris, StarRocks, SQL Server, Synapse, Impala	-

DSC支持的目标数据库如所示。

表 5-3 支持的目标数据库

数据库名称	数据库版本
DWS	DWS 8.1.0及以上集群版本

硬件要求

DSC对硬件的要求如表5-4所示。

表 5-4 DSC 硬件环境要求

硬件	配置
CPU	AMD或Intel Pentium （最小频率：500 MHz）
最小内存	1 GB
磁盘空间	1 GB

软件要求

操作系统要求

DSC兼容的操作系统如表5-5所示。

表 5-5 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）

服务器	操作系统	版本
		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）
		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	7.0，10，11

其他软件要求

DSC对其他软件版本的要求如表5-6所示。

表 5-6 其他软件要求

软件	用途
JDK 1.8.0_141 or later	Used to run DSC.
Perl 5.8.8	Used to migrate Perl files.
Perl 5.28.2 and later	Used to migrate Perl files in Windows.

软件	用途
Python 3.8.2	Used to verify post migration script.

5.3 使用 DSC

5.3.1 概述

本章主要介绍关于DSC使用过程中相关的内容，包括DSC工具的安装，工具配置，DSC工具的迁移流程等内容。

须知

请务必使用最新的补丁更新操作系统和相关软件，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。

5.3.2 下载并安装 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5-5**。

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**运行环境**。

  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。

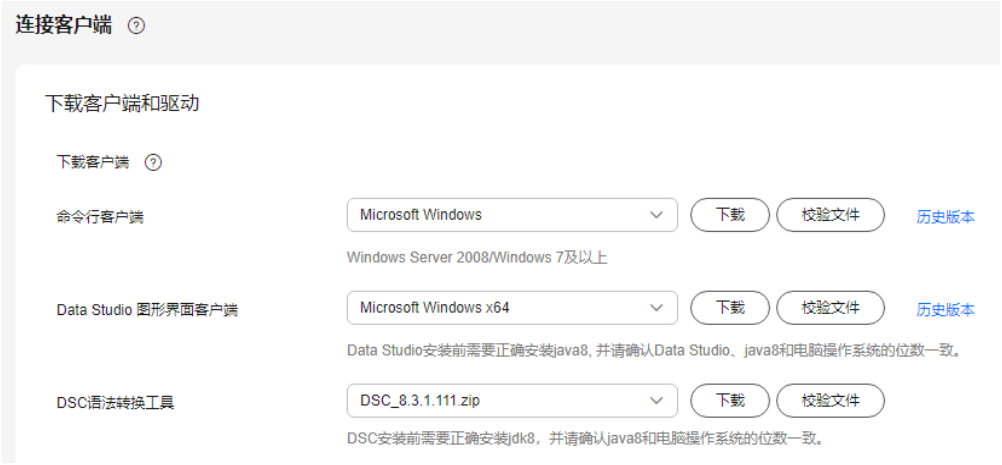
    假设Java安装路径为“C:\Program Files\Java\jdk1.8.0_141\bin”，Path环境变量为“*c:|windows|system32;*，则Path应该设置为“*c:|windows|system32;***C:\Program Files\Java\jdk1.8.0_141\bin;**”。

下载 DSC 工具-界面方式

步骤1 登录DWS管理控制台。在左侧导航栏中，单击“管理 > 连接客户端”，进入“下载客户端和驱动”页面。

步骤2 在“下载客户端和驱动”页面，单击“这里”下载“DSC”软件压缩包。

图 5-2 下载客户端



步骤3 解压下载的客户端软件包到自定义路径下。

----结束

下载 DSC 工具-链接方式

如果没有登录DWS管理控制台，也可以通过以下链接直接获取DSC工具。

表 5-7 DSC 下载地址

适用操作系统	下载地址	校验文件
请参见运行环境	DSC_xxx.zip	DSC_xxx.zip.sha256

安装 DSC 工具

DSC是一款运行在Linux或Windows操作系统上的命令行工具，可免安装使用，下载软件包后，用户解压软件包即可使用。

Windows操作系统:

步骤1 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2 进入DSC目录。

步骤3 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5-8所示。

----结束

Linux操作系统:

步骤1 进入DSC.zip所在目录，解压DSC.zip文件，其中DSC.zip请以实际获取的包名为准。

```
unzip DSC.zip
```

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 5-8 DSC 目录

文件或文件夹	说明
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。

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。

5.3.3 配置 DSC

5.3.3.1 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、MySQL迁移。

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5-9。

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 5-9 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

参数	说明	取值范围	默认值	样例
• 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 • .dsql 等。	• csv • txt • SQL	SQL	fileExtension=SQL
• formattedSourceRequired	指定是否使用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

参数	说明	取值范围	默认值	样例
• 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
• encodingFormat	指定输入/源文件的编码格式。 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。	• UTF8 • UTF16 • UTF32 • GB2312 • ASCII等	基于区域设置的默认编码	encodingFormat=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

参数	说明	取值范围	默认值	样例
• MaxFileSizeWarning	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，如果超过这个限制，文件迁移将被跳过。	-	20MB	MaxFileSize=20MB

参数	说明	取值范围	默认值	样例

• 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

• initialJVMMemory	设置初始内存	NA	256 MB	initialJVMMemory=256MB 表明内存达到256MB时，进程将启动。
• maxJVMMemory	设置最大内存	NA	1024 MB	maxJVMMemory=2048m 表明内存达到2048 MB时，进程将启动。

参数	说明	取值范围	默认值	样例
• executesqlingauss	在DWS中运行迁移后的脚本。参数值范围为true/false。脚本仅可在Linux系统的服务器上运行。	• true • false	false	executesqlingauss=false

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 5-10 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

打开校验模块config文件夹下的gaussdb.properties文件，参照**表5-11**，配置参数以连接DWS。

表 5-11 gaussdb.properties 文件内参数

参数名	描述	取值范围	默认值	样例
gaussdb-user	DWS数据库用户，拥有全部权限。	NA	NA	user1
gaussdb-port	DWS数据库端口号。	NA	NA	8000
gaussdb-name	DWS的数据库名称。	NA	NA	gaussdb
gaussdb-ip	DWS数据库IP地址。	NA	NA	10.XX.XX.XX

5.3.3.2 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5-12中的参数。

表 5-12 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

参数	说明	取值范围	默认值	样例
• 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**Group By**（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，则不可迁移GROUP BY()。	• true • false	false	extendedGroupByClause=false
• inToExists	该参数可用于启用和禁用从IN/NOT IN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**（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

参数	说明	取值范围	默认值	样例
• session_mode	该参数用于在运行**CREATE TABLE**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$开头的对象名称括起来。 若该参数设为false，则直接迁移以$开头的对象。 **说明** 详情请参见**以$开头的对象名称**。	• true • false	true	tdMigrateDOLLAR=true

参数	说明	取值范围	默认值	样例
• 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**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.	• true • false	false	tdMigrateVIEWCHECKOPTION=true

参数	说明	取值范围	默认值	样例
• 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**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。工具按原样复制查询，并在错误日志文件中记录以下消息，包括查询细节（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**。	• true • false	false	tdMigrateCharsetCase=false **说明** 如果tdminatecharsetcase = true，则注释该字符的特殊关键字。
• 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

参数	说明	取值范围	默认值	样例
• 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。 支持以下参数值： • true • false	• true • false	false	tdcolumnInSensitive=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

参数	说明	取值范围	默认值	样例
• 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

5.3.3.3 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5-13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 5-13 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-name	指定要在Perl文件中处理的脚本。 BTEQ：仅处理BTEQ标记的脚本。	• bteq	bteq	db-bteq-tag-name=bteq

参数	说明	取值范围	默认值	样例
• 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

参数	说明	取值范围	默认值	样例

• 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
	`$onesql ="SELECT TRIM( tablename ) FROM dbc.tables WHERE databasename = '${AUTO_DQDB}' AND tablename LIKE 'V_%' ORDER BY 1 ;`
		• true
		• false
			true	migrate-variables=true

参数	说明	取值范围	默认值	样例
	"; $sth_rundq = $dbh->execute_query($one sql);			
• 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。	1MB - 10MB	5MB	log-file-size=10MB

参数	说明	取值范围	默认值	样例

• 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

5.3.3.4 MySQL 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**features-mysql.properties文件中的配置参数**中的参数。

表 5-14 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

参数	说明	取值范围	默认值	样例
• 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、复制表、round-robin表。REPLICATION、HASH、ROUND-ROBIN。	• HASH • REPLICATION • ROUND-ROBIN	• 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.partition-key.name	分区键设置，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。	• 预留参数	• 默认为空	• table.partition-key.name=

参数	说明	取值范围	默认值	样例
• 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。	• YES • NO • YES • NO • LOW • MIDDLE • HIGH	• NO • LOW	• table.compress.row=NO • table.compress.column=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

参数	说明	取值范围	默认值	样例
• table.database.encoding	A database code.	• UTF8 • SQL_ASCII • GBK • Latin1 codes	• UTF8	table.database.encoding=UTF8
• table.index.rename	创建索引时，是否重新命名索引名。	• true • false	false	table.index.rename=false
• table.database.onlyFullGroupBy	select后非聚合列是否全部出现在group by中。	• true • false	true	table.database.onlyFullGroupBy=true
• table.database.realAsFloat	REAL数据类型转换使用，默认false，转换为DOUBLE PRECISION；改为true时，转换为REAL。	• true • false	false	table.database.realAsFloat=false

5.3.3.5 SQL-Server 配置

设置SQL-Server配置参数可在迁移SQL-Server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，在迁移SQL-Server时，必须将源库数据类型修改为SQL-Server，并根据实际需要设置表1中的参数。

表 5-15 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.origin.database.type	源数据库类型	bigquery,doris,sqlserver,mysql,adb	mysql	table.origin.database.type=sqlserver
table.orientation	默认数据存储方式	ROW,COLUMN	ROW	table.orientation=ROW
table.type	默认的表类型	REPLICATION,HASH,ROUND-ROBIN	HASH	table.type=ROUND-ROBIN

5.3.3.6 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5-16中的参数。

表 5-16 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

参数	说明	取值范围	默认值	样例
• maxValInSequence	设置数据库可以支持的最大序列值。目前，数据库支持的最大值是922337203685 4775807。	1-9223372036854775807	9223372036854775807	maxValInSequence=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

参数	说明	取值范围	默认值	样例
• 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释掉。 设为false，可跳过迁移，原样复制语句。	• true • false	true	RemoveListPartition=false
• 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

参数	说明	取值范围	默认值	样例
• RemovePartitionTS	设置是否注释掉PartitionTS 语句。 设为true，注释PartitionTS 语句。 设为false，原样保留PartitionTS语句。	• true • false	true	RemovePartitionTS=true
• 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

参数	说明	取值范围	默认值	样例
• 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
• 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

参数	说明	取值范围	默认值	样例
• 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	支持以下参数值： • true • false 设为true，迁移connectBy。	• true • false	true	migSupportConnectBy = true
• 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

参数	说明	取值范围	默认值	样例
• 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 设为true，会在存储过程/函数声明中的AS|IS前面增加PACKAGE。	• true • false	false	addPackageTag = true
• 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

参数	说明	取值范围	默认值	样例
• 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
• 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。

5.3.3.7 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5-17中的参数。

表 5-17 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 • none	bytea	cstore_blob=bytea
• 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

5.3.4 迁移流程

5.3.4.1 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5-18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 5-18 自定义数据库脚本

自定义脚本	说明
mig_fn_get_datatype_short_name.sql	Teradata函数的自定义数据库脚本

自定义脚本	说明
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

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5-19所示。

SQL文件包含自定义迁移函数。DWS数据库需要通过这些函数支持Teradata的具体特性。

表 5-19 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
------ 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5-20配置对应的参数。

表 5-20 DSC 配置参数

迁移场景	配置文件	配置参数
Teradata SQL迁移	• DSC：*application.properties* • 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 temporary**/unlogged
Teradata Perl迁移	• DSC：*application.properties* • 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

迁移场景	配置文件	配置参数
MySQL SQL迁移	• DSC：*application.properties* • MySQL配置：*features-mysql.properties*	table.databaseAsSchema=true table.defaultSchema=public table.schema= table.orientation=ROW table.type=HASH table.partition-key.choose.strategy=partitionKeyChooserStrategy table.partition-key.name= table.compress.mode=NOCOMPRESS table.compress.level=0 table.compress.row=NO table.compress.column=LOW table.database.template=template0 table.index.rename=false table.database.onlyFullGroupBy=true table.database.realAsFloat=false

5.3.4.2 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：
  1. 打开config文件夹中的application.properties文件。
  2. 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

5.3.4.3 执行 DSC

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**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**日志参考**。
- 单条SQL大小约束为20KB，超过此大小可能会导致执行过慢，从而转换失败。

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5-21**。

表 5-21 Windows 和 Linux 场景迁移

迁移场景	命令行参数
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]
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]

迁移场景	命令行参数
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**]

说明

- 命令行参数说明：
  - source-db指定源数据库，参数值为Teradata等，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。
  - target-db指定目标数据库，参数值为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。

任务示例

- 示例：将Teradata数据库的SQL文件迁移到适用于Linux系统下的DWS的SQL脚本中。

  ./runDSC.sh --source-db *Teradata* **--input-folder** D:\test\conversion\input **--output-folder** *D:|test|conversion|output* **--log-folder** *D:|test|conversion|log* **--conversion-type** *ddl* **--targetdb** gaussdb
- 示例：执行以下命令，将Teradata数据库的SQL文件迁移到适用于Windows操作系统下的DWS的SQL脚本中。

  runDSC.bat --source-db *Teradata* **--input-folder** D:\test\conversion\input **--output-folder** *D:|test|conversion|output* **--log-folder** *D:|test|conversion|log* **--conversion-type** *ddl* **--targetdb** gaussdb

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 **********
```

5.3.4.4 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文件进行格式化。

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**日志参考**。

检查日志文件是否记录错误信息。如是，请参考**故障处理**。

5.3.4.5 故障处理

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**故障处理**。
- 迁移语法问题：由于迁移工具无法正确识别或迁移TD语法的问题。要了解更多遗留问题，请参见**约束和限制**。

5.3.5 数据库模式迁移

功能

runDSC.sh和runDSC.bat分别用于将Teradata、MySQL的schema和query迁移到GaussDB (DWS)上。

5.3.5.1 迁移参数指导

参数说明

表 5-22 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。	• Teradata • MySQL	N/A	--source-db Teradata(or) -S Teradata
--input-folder	-I	字符串	包含Teradata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*

全称	缩写	数据类型	说明	范围	默认值	示例
--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 *ddl* 或 -M *ddl*
--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	• gaussdbA	gaussdbA	--target-db gaussdbA（或） *-T gaussdbA*

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 Teradata --input-folder opt/DSC/DSC/input/oracle/ --output-folder /opt/DSC/DSC/output/ --log-folder /opt/DSC/DSC/log/ --application-lang SQL --conversion-type ddl --targetdb gaussdbA
```

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）

创建DWS数据库和schema

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 -I input/ -S oracle -M ddl -L log_temp -P input/bulk/1_table/
```

5.3.5.2 Teradata SQL 迁移

工具支持从Teradata到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**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**执行DSC**。

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：

  1.teradata-set-table.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。

5.3.5.3 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**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**Teradata SQL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**Teradata Perl配置**。

有关命令行参数，详情请参见**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**故障处理**。
- 日志详情请参见**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

5.3.5.4 MySQL SQL 迁移

工具支持从MySQL到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；应用程序语言类型是SQL。

```
./runDSC.sh
--source-db MySQL
[--input-folder <input-script-path>]
[--output-folder <output-script-path>]
[--application-lang SQL]
[--conversion-type <conversion-type>]
[--log-folder <log-path>]
```

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。

```
./runDSC.sh --source-db MySQL --input-folder /opt/DSC/DSC/input/mysql/ --output-folder /opt/DSC/DSC/
output/ --application-lang SQL --conversion-type BULK --log-folder/opt/DSC/DSC/log/
********** Schema Conversion Started **********
DSC process start time : Mon Jan 20 17:24:49 IST 2020
Statement count progress 100% completed [FILE(1/1)]
Schema Conversion Progress 100% completed
********************************************************************************
Total number of files in input folder : 1
Total number of valid files in input folder : 1
********************************************************************************
Log file path :...../DSC/DSC/log/dsc.log
Error Log file :
DSC process end time : Mon Jan 20 17:24:49 IST 2020
DSC total process time : 0 seconds
********** Schema Conversion Completed **********
```

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。

```
runDSC.bat --source-db MySQL --target-db GaussDBA --input-folder D:\test\conversion\input --output-folder D:\test\conversion\output --log-folder D:\test\conversion\log --application-lang SQL --conversion-type  BULK
********** Schema Conversion Started **********
DSC process start time : Mon Jan 20 17:24:49 IST 2020
Statement count progress 100% completed [FILE(1/1)]
Schema Conversion Progress 100% completed
**********
Total number of files in input folder : 1
**********
Log file path :...../DSC/DSC/log/dsc.log
DSC process end time : Mon Jan 20 17:24:49 IST 2020
DSC total process time : 0 seconds
********** Schema Conversion Completed **********
```

5.3.5.5 Oracle SQL 迁移

工具支持从Oracle到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**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**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;
```

DWS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**执行DSC**。

5.3.5.6 Netezza SQL 迁移

工具支持从Netezza到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**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**执行DSC**。

5.3.5.7 迁移验证

迁移后验证

Database Schema Convertor转换完含有SQL语句的源文件后，在目标DWS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客户端的依赖关系：

   a. 由于在GaussDB上执行脚本时需依赖gsql(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

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 **********
```

5.3.6 Version 命令迁移

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
```

使用指南

Linux:

```
./runDSC.sh --version
```

Windows:

```
runDSC.bat --version
```

系统回显

```
Version: DSC (Gauss Tools v2.0.0)
```

5.3.7 Help 命令迁移

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
```

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 DWS
runDSC.sh -S <source-database> [-T <target-database>] -I <input-script-path> -O <output-script-path> [-M
<conversion-type>]  [-A <application-lang>]  [-L <log-path>]  [-VN <version-number>]


   -S | --source-db
   The source database, which can be either Teradata or Oracle or Netezza or MySql or DB2.


   -T | --target-db
   The target database, which can be either GaussDBT or GaussDBA.


   -I | --input-folder
   The input/source folder that contains the Teradata/Oracle/Netezza/MySql/DB2 scripts to be migrated.


   -O | --output-folder
   The output/target folder where the migrated scripts are placed.


   -M | --conversion-type
   The conversion type, which can be either Bulk or BLogic.


   -A | --application-lang
   The application language type, which can be either SQL or Perl.


   -L | --log-folder
   The log file path where the log files are created.


   -VN | --version-number
   The version number, which can be either V1R7 or V1R8_330.

  " -P | --pre-execution-path%n" + "
   Path containig the dependent Objects scripts which needs to be executed before the verification step.%n"
+ "%n" +



To display DSC version details
runDSC.sh -V | --version


To display DSC help details
runDSC.sh -H | --help


[Internal] To guess encoding for a file
runDSC.sh guessencoding -F <filename>


   -F | --file-name
   The filename for which the encoding must be guessed.


Refer the user manual for more details.
```

Windows操作系统:

```
runDSC.bat --help
To migrate teradata/oracle/netezza/mysql/db2 database scripts to DWS
runDSC.bat -S <source-database> [-T <target-database>] -I <input-script-path> -O <output-script-path> [-M <conversion-type>]  [-A <application-lang>]  [-L <log-path>]  [-VN <version-number>]


   -S | --source-db
   The source database, which can be either Teradata or Oracle or Netezza or MySql or DB2.


   -T | --target-db
   The target database, which can be either GaussDBT or GaussDBA.


   -I | --input-folder
   The input/source folder that contains the Teradata/Oracle/Netezza/MySql/DB2 scripts to be migrated.


   -O | --output-folder
   The output/target folder where the migrated scripts are placed.


   -M | --conversion-type
   The conversion type, which can be either Bulk or BLogic.


   -A | --application-lang
   The application language type, which can be either SQL or Perl.


   -L | --log-folder
   The log file path where the log files are created.


   -VN | --version-number
   The version number, which can be either V1R7 or V1R8_330.


To display DSC version details
runDSC.sh -V | --version


To display DSC help details
runDSC.sh -H | --help


[Internal] To guess encoding for a file
runDSC.sh guessencoding -F <filename>


   -F | --file-name
   The filename for which the encoding must be guessed.


Refer the user manual for more details.
```

5.3.8 日志参考

5.3.8.1 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

5.3.8.2 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 xxxxx
2020-01-22 09:35:10,828 INFO CLMigrationUtility:456 Successfully changed permission of files in
D:\Migration\Gauss_Tools_18_Migration\code\migration\config
2020-01-22 09:35:10,832 INFO PropertyLoader:90 Successfully loaded Property file : D:\Migration
\Gauss_Tools_18_Migration\code\migration\config\application.properties
2020-01-22 09:35:10,833 INFO ApplicationPropertyLoader:42 Application properties have been loaded
Successfully
2020-01-22 09:35:10,917 INFO MigrationValidatorService:549 Files in output directory has been overwritten
as configured by xxxxx
2020-01-22 09:35:10,920 INFO PropertyLoader:90 Successfully loaded Property file : D:\Migration
\Gauss_Tools_18_Migration\code\migration\config\features-oracle.properties
2020-01-22 09:35:10,921 INFO FeatureLoader:41 Features have been loaded Successfully
2020-01-22 09:35:10,926 INFO MigrationService:80 DSC process start time : Wed Jan 22 09:35:10 GMT
+05:30 2020
2020-01-22 09:35:10,933 INFO FileHandler:179  File is not supported. D:\Migration_Output\Source
\ARRYTYPE.sql-
2020-01-22 09:35:10,934 INFO FileHandler:179  File is not supported. D:\Migration_Output\Source
\varray.sql-
2020-01-22 09:35:12,816 INFO PropertyLoader:90 Successfully loaded Property file : D:\Migration
```

```
\Gauss_Tools_18_Migration\code\migration\config\global-temp-tables.properties
2020-01-22 09:35:12,830 INFO PropertyLoader:90 Successfully loaded Property file : D:\Migration
\Gauss_Tools_18_Migration\code\migration\config\create-types-UDT.properties
2020-01-22 09:35:12,834 INFO PropertyLoader:90 Successfully loaded Property file : D:\Migration
\Gauss_Tools_18_Migration\code\migration\config\package-names-oracle.properties
2020-01-22 09:35:12,849 INFO DBMigrationService:76 Number of Available Processors: 4
2020-01-22 09:35:12,850 INFO DBMigrationService:78 Configured simultaneous processes in the Tool : 3
2020-01-22 09:35:13,032 INFO MigrationProcessor:94 File name: D:\Migration_Output\Source\Input.sql is
started
2020-01-22 09:35:13,270 INFO FileHandler:606 guessencoding command output = Error: Unable to access
jarfile D:\Migration\Gauss_Tools_18_Migration\code\migration\RE_migration\target\dsctool.jar , for file=
D:\Migration_Output\Source\Input.sql
2020-01-22 09:35:13,272 INFO FileHandler:625 couldn't get the encoding format, so using the default
charset for D:\Migration_Output\Source\Input.sql
2020-01-22 09:35:13,272 INFO FileHandler:310 File D:\Migration_Output\Source\Input.sql will be read with
charset : UTF-8
2020-01-22 09:35:13,390 INFO FileHandler:668 D:\Migration_Output\target\output\Input.sql - File already
exists/Failed to create target file
2020-01-22 09:35:13,562 INFO FileHandler:606 guessencoding command output = Error: Unable to access
jarfile D:\Migration\Gauss_Tools_18_Migration\code\migration\RE_migration\target\dsctool.jar , for file=
D:\Migration_Output\Source\Input.sql
2020-01-22 09:35:13,563 INFO FileHandler:625 couldn't get the encoding format, so using the default
charset for D:\Migration_Output\Source\Input.sql
2020-01-22 09:35:13,563 INFO FileHandler:675 File D:\Migration_Output\Source\Input.sql will be written
with charset : UTF-8
2020-01-22 09:35:13,604 INFO MigrationProcessor:139 File name: D:\Migration_Output\Source\Input.sql is
processed successfully
2020-01-22 09:35:13,605 INFO MigrationService:147 Total number of files in Input folder : 3
2020-01-22 09:35:13,605 INFO MigrationService:148 Total number of queries : 1
22020-01-22 09:35:13,607 INFO PropertyLoader:164 Successfully updated Property file : D:\Migration
\Gauss_Tools_18_Migration\code\migration\config\global-temp-tables.properties
2020-01-22 09:35:13,630 INFO PropertyLoader:164 Successfully updated Property file : D:\Migration
\Gauss_Tools_18_Migration\code\migration\config\create-types-UDT.properties
2020-01-22 09:35:13,631 INFO PropertyLoader:164 Successfully updated Property file : D:\Migration
\Gauss_Tools_18_Migration\code\migration\config\package-names-oracle.properties
2020-01-22 09:35:13,632 INFO CLMigrationUtility:305 Log file : dsc.log and the file is present in the path :
D:\Migration_Output\log
2020-01-22 09:35:13,632 INFO CLMigrationUtility:312 DSC process end time : Wed Jan 22 09:35:13 GMT
+05:30 2020
2020-01-22 09:35:13,632 INFO CLMigrationUtility:217 Total process time : 2842 seconds
```

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.
```

5.3.8.3 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**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
```

5.4 Teradata 语法迁移

5.4.1 支持的关键词和特性

DSC支持迁移的Teradata关键字和特性如**表5-23**所示。

- “版本”列代表初次支持该关键字/特性的版本。
- “备注”列包含该特性的配置参数，可用于自定义迁移工具迁移相应的关键字/特性。

表 5-23 支持的 Teradata 关键词和特性列表

章节	对象->关键词/特性		版本	备注
数据类型	数据类型		6.5.1、18.2.0	-
函数和操作符	分析函数	ORDER BY中的分析函数	18.0.0	-
		PARTITION BY中的分析函数	18.1.0	-

章节	对象->关键词/特性		版本	备注
	数学函数	• ** • MOD • NULLIFZERO • ZEROIFNULL	V100R003C00	可配置：tdMigrateNULLIFZERO、tdMigrateZEROIFNULL（18.0.0）
	字符串函数	CHAR函数 CHARACTERS函数 INDEX STRREPLACE OREPLACE	V100R003C00	-
	日期和时间函数	DATE TIMESTAMP NEXT	V100R003C00 18.0.0 V100R003C00	-
	比较和列表操作符	• ^=和GT • EQ和NE • LE和GE • NOT=和LT • IN和NOT IN • IS NOT IN • LIKE ALL/NOT LIKE ALL • LIKE ANY/NOT LIKE ANY	V100R003C00	-
	表操作符	-	18.0.0	-
	查询优化操作符	IN和NOT IN转换	V100R003C00	可配置：inToExists
	QUALIFY	-	V100R003C00	可配置：rowstoreToColumnstore
	ALIAS		V100R003C00	可配置：tdMigrateALIAS

章节	对象->关键词/特性		版本	备注
	FORMAT和CAST		V100R003C00	-
	缩写关键字迁移		V100R003C00	-
	以$开头的对象名称迁移		18.0.0	可配置：tdMigrateDollar
表迁移	-	CREATE TABLE	V100R003C00	可配置：session_mode
		CHARACTER SET和CASESPECIFIC	18.1.0	可配置：tdMigrateCharsetCase
		VOLATILE	V100R003C00	可配置： volatile、session_mode
		SET	V100R003C00	-
		MULTISET	V100R003C00	-
		TITLE	V100R003C00	-
		索引	V100R003C00	-
		约束	V100R003C00	-
		COLUMN STORE	V100R003C00	可配置：rowstoreToColumnstore
		PARTITION	18.0.0	-
		ANALYZE	V100R003C30	-
		支持指定部分列	18.0.0	-
索引迁移	-		V100R003C00	-

章节	对象->关键词/特性	版本	备注
视图迁移	REPLACE VIEW CHECK OPTION VIEW WITH RECURSIVE VIEW WITH ACCESS LOCK	V100R003C00	-
COLLECT STATISTICS	-	V100R003C00	-
ACCESS LOCK	-	V100R003C00	可配置：tdMigrateLOCKOption
DBC.COLUMNS	-	18.0.0	需自定义数据库脚本
DBC.TABLES	-	18.1.0	-
DBC.INDICES	-	18.1.0	-
数据操作语句（DML）	SELECT	V100R003C00	-
数据操作语句（DML）	UPDATE	V100R003C00	-
数据操作语句（DML）	DELETE	V100R003C00	-
数据操作语句（DML）	MERGE	V100R003C00	-
数据操作语句（DML）	NAMED	18.0.0	-

章节	对象->关键词/特性	版本	备注
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	V100R003C00	-

5.4.2 约束和限制

DSC迁移Teradata的约束和限制如下：

- 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，则可能导致迁移失败。

Teradata

- 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' then 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
                     );
```

- 带有模式名的数据库更改为“SET SESSION CURRENT_SCHEMA”。

TD 语法	迁移后语法
DATABASE SCHTERA	SET SESSION CURRENT_SCHEMA TO SCHTERA

- 输入文件中包含表的专用关键词MULTISET VOLATILE，但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）。

5.4.3 数据类型

数据类型	输入	输出
数值类型	BIGINT	BIGINT
	BYTEINT	SMALLINT
	DECIMAL [(n[,m])]	DECIMAL [(n[,m])]
	DOUBLE PRECISION	DOUBLE PRECISION
	FLOAT	DOUBLE PRECISION
	INT / INTEGER	INTEGER
	NUMBER / NUMERIC	NUMERIC
	NUMBER(n[,m])	NUMERIC (n[,m])
	REAL	REAL
	SMALLINT	SMALLINT
字符类型	CHAR[(n)] / CHARACTER [(n)]	CHAR(n)
	CLOB	CLOB
	LONG VARCHAR	TEXT
	VARCHAR(n) / CHAR VARYING(n) / CHARACTER VARYING(n)	VARCHAR(n)

数据类型	输入	输出
日期/时间类型	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
范围类型	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
二进制类型	BLOB[(n)]	blob
	BYTE[(n)]	bytea
	VARBYTE[(n)]	bytea

BYTEINT

输入：

```
SELECT cast(col as byteint) FROM tab;
```

输出：

```
SELECT CAST( col AS SMALLINT ) FROM tab ;
```

5.4.4 函数和操作符

5.4.4.1 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。

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;
```

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
;
```

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。

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;
```

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;
```

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 ...;
```

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
;
```

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
'******' */
```

5.4.4.2 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' ;
```

5.4.4.3 字符串函数

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 … ;
```

STRTOK 函数

输入：

```
LENGTH(STRTOK(STRTOK(JOB_NAME_TADD,'-',4),'_',2))
```

输出：

```
LENGTH(split_part(split_part(JOB_NAME_TADD,'-',4),'_',2))
```

5.4.4.4 日期和时间函数

DATE

DSC支持迁移Teradata中包含DATE FORMAT的SELECT语句，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：

```
mig_td_ext.ADD_MONTHS(CAST(substr(T1.GRANT_DATE,1,8)||'01'AS DATE FORMAT 'YYYY-MM-DD'),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);
```

5.4.4.5 比较和列表操作符

以下章节介绍了支持的比较和列表操作符。

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。

^=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;
```

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;
```

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;
```

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;
```

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

DWS支持IN和NOT IN操作符，特殊场景除外。

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);
```

LIKE ALL/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%' ]);
```

LIKE ANY/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%' ]);
```

5.4.4.6 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);
```

5.4.4.7 查询优化操作符

本章节主要介绍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。

迁移到DWS的SQL查询，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;
```

5.4.4.8 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
;
```

5.4.4.9 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
;
```

5.4.4.10 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**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DWS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;
```

5.4.4.11 缩写关键字迁移

表5-24列出了Teradata支持的缩写关键字及其语法在DWS中对应的语法。

表 5-24 缩写关键字列表

Teradata语法	对应的DWS语法
SEL	SELECT
INS	INSERT
UPD	UPDATE
DEL	DELETE
CT	CREATE TABLE
CV	CREATE VIEW
BT	START TRANSACTION

Teradata语法	对应的DWS语法
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;
```

5.4.4.12 以$开头的对象名称迁移

本章节介绍如何迁移以$（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 5-25 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} 动态对象名称	${V_SQL} 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 ;
```

5.4.5 表迁移

5.4.5.1 CREATE TABLE

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中复制的结构将用于创建表tab2。如果CREATE TABLE语句包含WITH DATA选项，则会将tab1的数据也复制到tab2中。使用CREATE AS时，源表中的CONSTRAINT行将保留在新表中。

- 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;
```

5.4.5.2 CHARACTER SET 和 CASESPECIFIC

CHARACTER SET用于指定字符列的服务器字符集，CASESPECIFIC用于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)
        ) ;
```

5.4.5.3 VOLATILE

输入文件中包含表的专用关键词VOLATILE，但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
;
/
```

5.4.5.4 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 ;
```

5.4.5.5 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
     )
;
```

5.4.5.6 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
;
```

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;
```

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};
```

5.4.5.7 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

由于索引列的列表（organic_name）不是DISTRIBUTE BY列的列表（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 )
  )
;
```

创建带索引分区表

如果配置参数**tdMigrateRANGE_N**为**true**。

输入：

```
CREATE SET TABLE SC.TAB , NO FALLBACK,
NO BEFORE JOURNAL,
NO AFTER JOURNAL,
CHECKSUM=DEFAULT,
DEFAULT MERGEBLOCKRATIO
(
ACCOUNT_NUM VARCHAR(255) CHARACTER SET LATIN NOT CASESPECIFIC NOT NULL
,ACCOUNT_MODIFIER_NUM CHAR(18) CHARACTER SET LATIN NOT CASESPECIFIC NOT NULL
,END_DT DATE FORMAT 'YYYY-MM-DD'
,UPD_TXF_BATCHTD INTEGER COMPRESS
)
PRIMARY INDEX XPKT0300_AGREEMENT (ACCOUNT_NUM,ACCOUNT_MODIFIER_NUM)
PARTITION BY RANGE_N(END_DT BETWEEN '2001-01-01' AND '2020-12-31' EACH INTERVAL '1' DAY, NO
RANGE ,UNKNOWN)
INDEX (UPD_TXF_BATCHTD)
;
```

输出：

```
CREATE
    TABLE
        SC.TAB (
            ACCOUNT_NUM VARCHAR( 255 ) /* CHARACTER SET LATIN*/
             /* NOT CASESPECIFIC*/              NOT NULL
            ,ACCOUNT_MODIFIER_NUM CHAR( 18 ) /* CHARACTER SET LATIN*/              /* NOT
CASESPECIFIC*/              NOT NULL
            ,END_DT DATE
            ,UPD_TXF_BATCHTD INTEGER /* COMPRESS */
        ) DISTRIBUTE BY HASH (
            ACCOUNT_NUM
            ,ACCOUNT_MODIFIER_NUM
        ) PARTITION BY RANGE (END_DT) (
            PARTITION TAB_1 start ('2001-01-01')
        END ('2020-12-31') EVERY (
            INTERVAL '1' DAY )
        ) ;
CREATE INDEX ON SC.TAB (UPD_TXF_BATCHTD) LOCAL;
```

5.4.5.8 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。

5.4.5.9 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);
```

5.4.5.10 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。

输入：

```
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 )
                    ) */
;
```

RANGE_N 在字符串列的分区

输入：

```
CREATE SET TABLE SC.TAB , NO FALLBACK,
NO BEFORE JOURNAL,
NO AFTER JOURNAL,
CHECKSUM=DEFAULT,
DEFAULT MERGEBLOCKRATIO
(
ACCOUNT_NUM VARCHAR(255) CHARACTER SET LATIN NOT CASESPECIFIC NOT NULL
,ACCOUNT_MODIFIER_NUM CHAR(18) CHARACTER SET LATIN NOT CASESPECIFIC NOT NULL
,DATA_SOURCE_ID CHAR(10) CHARACTER SET LATIN NOT CASESPECIFIC
,END_DT DATE FORMAT 'YYYY-MM-DD'
,UPD_TXF_BATCHTD INTEGER COMPRESS
)
PRIMARY INDEX XPKT0300_AGREEMENT (ACCOUNT_NUM,ACCOUNT_MODIFIER_NUM)
PARTITION BY ( RANGE_N(DATA_SOURCE_ID BETWEEN
'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z' AND 'ZZ', NO
RANGE ,UNKNOWN) ,CASE_N(END_DT IS NULL , NO CASE , UNKNOWN))
;
```

输出：

```
CREATE
    TABLE
        SC.TAB (
            ACCOUNT_NUM VARCHAR( 255 ) /* CHARACTER SET LATIN*/       /* NOT
CASESPECIFIC*/               NOT NULL
            ,ACCOUNT_MODIFIER_NUM CHAR( 18 ) /* CHARACTER SET LATIN*/              /* NOT
CASESPECIFIC*/               NOT NULL
            ,DATA_SOURCE_ID CHAR( 10 ) /* CHARACTER SET LATIN*/              /* NOT CASESPECIFIC*/
            ,END_DT DATE
            ,UPD_TXF_BATCHTD INTEGER /* COMPRESS */
        ) DISTRIBUTE BY HASH (
            ACCOUNT_NUM
            ,ACCOUNT_MODIFIER_NUM
        )/* PARTITION BY (
```



```
            RANGE_N (
                DATA_SOURCE_ID BETWEEN 'A'
                ,'B'
                ,'C'
                ,'D'
                ,'E'
                ,'F'
                ,'G'
                ,'H'
                ,'I'
                ,'J'
                ,'K'
                ,'L'
                ,'M'
                ,'N'
                ,'O'
                ,'P'
                ,'Q'
                ,'R'
                ,'S'
                ,'T'
                ,'U'
                ,'V'
                ,'W'
                ,'X'
                ,'Y'
                ,'Z' AND 'ZZ'
                ,NO RANGE
                ,UNKNOWN
            )
            ,*/
/* CASE_N(END_DT IS NULL , NO CASE , UNKNOWN))  */
            ;
```

RANGE_N with different partition INTERVAL

输入：

```
CREATE MULTISET TABLE tab1
    ( TICD                VARCHAR(10)
    , TCIT                VARCHAR(10)
    , TCCM                 VARCHAR(50)
    , DW_Stat_Dt          DATE
   )
PRIMARY INDEX ( TICD )
PARTITION BY RANGE_N
  ( DW_Stat_Dt BETWEEN DATE '0001-01-01' AND DATE '0001-01-04' EACH INTERVAL '1' DAY,
    DATE '0001-01-05' AND DATE '1899-12-31',
    DATE '1900-01-01' AND DATE '1900-01-01',
    DATE '1900-01-02' AND DATE '1999-12-31',
    DATE '2000-01-01' AND DATE '2009-12-31' EACH INTERVAL '1' YEAR,
    DATE '2010-01-01' AND DATE '2021-12-31' EACH INTERVAL '1' DAY,
    DATE '9999-12-31' AND DATE '9999-12-31',
    NO RANGE );
```

输出：

```
CREATE TABLE tab1
    ( TICD                VARCHAR( 10 )
    , TCIT                VARCHAR( 10 )
    , TCCM                 VARCHAR( 50 )
    , DW_Stat_Dt          DATE
    )
DISTRIBUTE BY HASH (TICD)
PARTITION BY RANGE (DW_Stat_Dt)
  ( PARTITION tab1_0 START (DATE '0001-01-01') END (DATE '0001-01-04') EVERY (INTERVAL '1' DAY),
    PARTITION tab1_1 START (DATE '0001-01-04') END (DATE '1899-12-31'),
    PARTITION tab1_2 START (DATE '1899-12-31') END (DATE '1900-01-01'),
    PARTITION tab1_ 3 START (DATE '1900-01-01') END (DATE '1999-12-31'),
```

```
    PARTITION tab1_4 START (DATE '1999-12-31') END (DATE '2009-12-31') EVERY (INTERVAL '1' YEAR) ,
    PARTITION tab1_5 START (DATE '2009-12-31') END (DATE '2021-12-31') EVERY (INTERVAL '1' DAY) ,
    PARTITION tab1_6 START (DATE '2021-12-31') END (DATE '9999-12-31')
 );
```

RANGE_N with * for start-date

输入：

```
CREATE MULTISET TABLE Orders5 (
  StoreNo SMALLINT,
  OrderNo INTEGER,
  OrderDate DATE,
  OrderTotal INTEGER
)
PRIMARY INDEX(OrderNo)
PARTITION BY RANGE_N  (
  OrderDate BETWEEN DATE * AND DATE '2016-12-31' EACH INTERVAL '1' YEAR,
    DATE '2017-01-01'  EACH INTERVAL '1' MONTH,
    DATE '2020-01-01' AND DATE '2020-12-31' EACH INTERVAL '1' DAY
);
```

输出：

```
CREATE TABLE Orders5 (
  StoreNo SMALLINT,
  OrderNo INTEGER,
  OrderDate DATE,
  OrderTotal INTEGER
)
DISTRIBUTE BY HASH (OrderNo)
PARTITION BY RANGE (OrderDate)
  ( PARTITION Orders5_0 START (DATE '0001-01-01') END (DATE '2016-12-31') EVERY (INTERVAL '1' YEAR),
PARTITION Orders5_1 START (DATE '2016-12-31') END (DATE '2020-01-01') EVERY (INTERVAL '1' MONTH),
PARTITION Orders5_2 START (DATE '2020-01-01') END (DATE '2020-12-31') EVERY (INTERVAL '1' DAY)
);
```

RANGE_N with * for end-date

输入：

```
CREATE SET TABLE Orders4 (
  StoreNo SMALLINT,
  OrderNo INTEGER,
  OrderDate DATE,
  OrderTotal INTEGER
)
PRIMARY INDEX(OrderNo)
PARTITION BY RANGE_N  (
  OrderDate BETWEEN DATE '2010-01-01' AND '2016-12-31' EACH INTERVAL '1' YEAR,
    DATE '2017-01-01' EACH INTERVAL '1' MONTH,
    DATE '2019-01-01' AND *
);
```

输出：

```
CREATE TABLE Orders4 (
  StoreNo SMALLINT,
  OrderNo INTEGER,
  OrderDate DATE,
  OrderTotal INTEGER
)
DISTRIBUTE BY HASH (OrderNo)
PARTITION BY RANGE (OrderDate)
  ( PARTITION Orders4_0 START (DATE '2010-01-01') END (DATE '2016-12-31') EVERY (INTERVAL '1' YEAR),
PARTITION Orders4_1 START (DATE '2016-12-31') END (DATE '2020-01-01') EVERY (INTERVAL '1' MONTH) ,
PARTITION Orders4_2 START (DATE '2020-01-01') END (MAXVALUE)
);
```

RANGE_N with comma separated values

输入：

```
CREATE TABLE orders10
    (storeid INTEGER NOT NULL
    ,productid INTEGER NOT NULL
    ,orderdate DATE NOT NULL
    ,totalorders INTEGER NOT NULL)
    PRIMARY INDEX (storeid, productid)
     PARTITION BY ( RANGE_N(totalorders BETWEEN *, 100, 1000 AND *) );
```

输出：

```
CREATE TABLE orders10
    (storeid INTEGER NOT NULL
    ,productid INTEGER NOT NULL
    ,orderdate DATE NOT NULL
    ,totalorders INTEGER NOT NULL)
DISTRIBUTE BY HASH (storeid, productid)
PARTITION BY RANGE (totalorders)
   ( PARTITION Orders10_0 END (100),
     PARTITION Orders10_1 END (1000),
     PARTITION Orders10_2 END (MAXVALUE)
   );
```

5.4.5.11 ANALYZE

Teradata的ANALYZE语句用于对表的迁移。

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);
```

5.4.5.12 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
;
```

5.4.6 索引迁移

Teradata中CREATE INDEX的列和表名的顺序和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 )
  )
;
```

5.4.7 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DWS均支持关键词VIEW，但SELECT语句在迁移过程中会用()。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 OPTION关键词的视图，则工具在目标数据库中注释掉CHECK OPTION。

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

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 5-3 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 5-4 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 ) ;
```

dbc.columnsV

输入：

```
SELECT A.ColumnName
 AS V_COLS              ,A.columnname  || ' ' ||CASE WHEN columnType in ('CF','CV')
THEN CASE WHEN columnType='CV' THEN 'VAR' ELSE ''              END||'CHAR('||TRIM(columnlength
(INT))||                ') CHARACTER SET LATIN'||                  CASE WHEN UpperCaseFlag='N'
THEN ' NOT' ELSE ''
END || ' CASESPECIFIC'
WHEN columnType='DA' THEN 'DATE'
WHEN columnType='TS' THEN 'TIMESTAMP(' || TRIM(DecimalFractionalDigits)||')'
WHEN columnType='AT' THEN 'TIME('|| TRIM(DecimalFractionalDigits)||')'                                  WHEN
columnType='I' THEN 'INTEGER'
 WHEN columnType='I1' THEN 'BYTEINT'
WHEN columnType='I2' THEN 'SMALLINT'
WHEN columnType='I8' THEN 'BIGINT'
WHEN columnType='D' THEN 'DECIMAL('||TRIM(DecimalTotalDigits)||','||
TRIM(DecimalFractionalDigits)||')'                                  ELSE 'Unknown'
END||CASE WHEN Nullable='Y'
THEN '' ELSE ' NOT NULL' END||'0A'XC
AS V_ColT
,D.ColumnName
AS V_PICol          --获得目标表主索引
FROM dbc.columnsV A LEFT JOIN dbc.IndicesV B  ON A.columnName = B.columnName AND B.IndexType
IN ('Q','P') AND B.DatabaseName = '${V_TDDLDB}'  AND B.tablename='${TARGET_TABLE}' WHERE
A.databasename='${V_TDDLDB}' AND A.tablename = '${TARGET_TABLE}' AND A.columnname NOT IN
( 'ETL_JOB_NAME'                                                                                ,'ETL_TX_DATE' ,
'ETL_PROC_DATE'
                        )ORDER BY A.columnid;
```

输出：

```
D DECLARE lv_mig_V_COLS    TEXT;         lv_mig_V_ColT        TEXT;         lv_mig_V_PICol       TEXT; BEGIN
SELECT STRING_AGG(A.ColumnName, ',')                  , STRING_AGG(A.columnname  || ' ' ||CASE WHEN
columnType in ('CF','CV')                                  THEN CASE WHEN columnType='CV' THEN 'VAR' ELSE
''             END||'CHAR('||TRIM(mig_td_ext.mig_fn_castasint(columnlength))||                ') /*CHARACTER SET
LATIN*/'||                  CASE WHEN UpperCaseFlag='N'                    THEN ' NOT' ELSE ''                  END
|| ' /*CASESPECIFIC*/'                                WHEN columnType='DA' THEN 'DATE'
WHEN columnType='TS' THEN 'TIMESTAMP(' || TRIM(DecimalFractionalDigits)||')'
WHEN columnType='AT' THEN 'TIME('|| TRIM(DecimalFractionalDigits)||')'                                  WHEN
columnType='I' THEN 'INTEGER'                                  WHEN columnType='I1' THEN
```

```
'BYTEINT'                                WHEN columnType='I2' THEN 'SMALLINT'                                WHEN
columnType='I8' THEN 'BIGINT'                                WHEN columnType='D' THEN 'DECIMAL('||
TRIM(DecimalTotalDigits)||','||TRIM(DecimalFractionalDigits)||')'                                ELSE
'Unknown'                                END||CASE WHEN Nullable='Y'        THEN '' ELSE ' NOT NULL' END||E'\x0A',
',')                , STRING_AGG(B.ColumnName, ',')                INTO lv_mig_V_COLS, lv_mig_V_ColT,
lv_mig_V_PICol FROM mig_td_ext.vw_td_dbc_columnsV A LEFT JOIN mig_td_ext.vw_td_dbc_IndicesV B  ON
A.columnName = B.columnName AND B.IndexType IN ('Q','P') AND B.DatabaseName = 'public'  AND
B.tablename='emp2' WHERE A.databasename='public' AND A.tablename = 'emp2'; -- ORDER BY
A.columnid; END; /
```

5.4.8 COLLECT STATISTICS

在Teradata中，COLLECT STAT采集优化器统计信息，用于查询性能。DWS使用ANALYZE语句来替代COLLECT STAT。

详情请参见**ANALYZE**。

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 ;
```

5.4.9 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 ...;
```

5.4.10 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**运行环境**和**前提条件**。

迁移工具将以下dbc.columns列迁移为对应的information_schema列：

表 5-26 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
;
```

5.4.11 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 > ] ) ) )
;
```

5.4.12 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();
```

5.4.13 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"*/
```

5.4.14 COMMENT 语句

COMMENT语句的迁移过程如下所示，输入一个复杂语句，迁移之后语句更加简化。

输入：

```
CREATE MULTISET TABLE PDAT.t_tbl_comment
   (
      Data_Dt  DATE           NOT NULL
     ,Data_Src  VARCHAR(4)    NOT NULL
     ,List_Make_Stat  CHAR(1)
   ) PRIMARY INDEX(Data_Dt,Data_Src) ;
COMMENT ON PDAT.t_tbl_comment IS 'comment test on table';
---------------
REPLACE VIEW  PVW.v_vw_comment AS
LOCKING ROW FOR ACCESS
SELECT Data_Dt, Data_Src, List_Make_Stat
FROM PDAT.t_tbl_comment;
COMMENT ON PVW.v_vw_comment IS 'comment test on view';
COMMENT ON PVW.v_vw_comment.Data_Dt IS 'comment test on view column';
```

输出：

```
COMMENT ON TABLE PDAT.t_tbl_comment IS 'comment test on table';
COMMENT ON VIEW PVW.v_vw_comment IS 'comment test on view';
COMMENT ON COLUMN PVW.v_vw_comment.Data_Dt IS 'comment test on view column';
```

5.4.15 数据操作语句（DML）

5.4.15.1 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DWS不支持。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;
```

5.4.15.2 SELECT

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;
```

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
        alias1 <> 'IC'
        AND alias2 <> 'IC'
        AND alias3 <> '' ;
```

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
        )
;
```

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
        alias1 <> 'IC'
        AND alias2 <> 'IC'
        AND alias3 <> '' ;
```

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
       );
```

SELECT AS

DWS变量名不区分大小写，TD变量名区分大小写。为保证TD脚本迁移前后正确，需要在SELECT语句变量定义中保留原变量名大小写形式。因此，转换后变量采用AS "变量名"形式定义。

输入示例

```
SELECT  TRIM('${JOB_NAME}')                                                    AS JOB_NAME
      ,CASE WHEN LENGTH(trim(STRTOK('${JOB_NAME}','-',4)))=2
           THEN trim(STRTOK('${JOB_NAME}','-',4))
           ELSE ''
           END                                                                 AS EDW_BANK_NM
      ,TRIM('${TX_DATE}')                                                      AS TX_DATE
      ,USER                                                                 AS ETL_USER
      ,CAST( CURRENT_TIMESTAMP(0) AS VARCHAR(19))                                    AS CURR_STIME
      ,'${ETL_DATA}'                                                        AS ETL_DATA
      ,'T61_INDV_CUST_ACCT_ORG_AUM'                                                AS TARGET_TABLE
      ,'CAST(''8999-12-31'' AS DATE)'                                           AS MAXDATE
;
.IF ERRORCODE <> 0 THEN .QUIT 12
```

输出示例

```
SELECT
        TRIM( '${job_name}' ) AS "JOB_NAME"
        ,CASE
                WHEN LENGTH( TRIM( split_part ( '${job_name}' ,'-' ,4 ) ) ) = 2 THEN TRIM( split_part ( '$
{job_name}' ,'-' ,4 ) )
             ELSE ''
```

```
        END AS "EDW_BANK_NM"
        ,TRIM( '${tx_date}' ) AS "TX_DATE"
        ,USER AS "ETL_USER"
        ,CAST( CURRENT_TIMESTAMP( 0 ) AS VARCHAR( 19 ) ) AS "CURR_STIME"
        ,'${etl_data}' AS "ETL_DATA"
        ,'T61_INDV_CUST_ACCT_ORG_AUM' AS "TARGET_TABLE"
        ,'CAST(''8999-12-31'' AS DATE)' AS "MAXDATE" ;
\if ${ERROR} != 'false'
 \q 12
\endif

;
```

嵌套AS表达式定义，必须拆解多条语句实现

输入示例

```
SELECT  TRIM('${JOB_NAME}')                                                    AS JOB_NAME
      ,'CAST(''0001-01-02'' AS DATE)'                                          AS ILLDATE
      ,'T61_INDV_CUST_HOLD_PROD_IND_AUM'                                       AS
TARGET_TABLE
      ,0                                                                 AS NULLNUMBER
      ,'CAST(''00:00:00.999'' AS TIME(3))'                                     AS NULLTIME
      ,'CAST(''0001-01-01 00:00:00.000000'' AS TIMESTAMP(6))'                  AS NULLTIMESTAMP
      ,'VT_'||TARGET_TABLE                                                AS VT_TABLE
      ,'V'||SUBSTR(TARGET_TABLE,2,CHAR(TARGET_TABLE)-1)                        AS
TARGET_TABLE_V
      ,'${GDM_DETAIL_DDL}'                                                     AS V_TDDLDB
      ,'${GDM_DETAIL_VIEW}'                                                    AS V_TARGETDB
      ,'${UDF}'                                                          AS V_PUB_UDF

;
.IF ERRORCODE <> 0 THEN .QUIT 12
```

输出示例

```
SELECT
        TRIM( '${job_name}' ) AS "JOB_NAME"
        ,'CAST(''0001-01-02'' AS DATE)' AS "ILLDATE"
        ,'T61_INDV_CUST_HOLD_PROD_IND_AUM' AS "TARGET_TABLE"
        ,0 AS "NULLNUMBER"
        ,'CAST(''00:00:00.999'' AS TIME(3))' AS "NULLTIME"
        ,'CAST(''0001-01-01 00:00:00.000000'' AS TIMESTAMP(6))' AS "NULLTIMESTAMP"
        ,'${gdm_detail_ddl}' AS "V_TDDLDB"
        ,'${gdm_detail_view}' AS "V_TARGETDB"
        ,'${udf}' AS "V_PUB_UDF" ;
SELECT
        'VT_' || '${TARGET_TABLE}' AS "VT_TABLE" ;
SELECT
        'V' || SUBSTR( '${TARGET_TABLE}' ,2 ,LENGTH( '${TARGET_TABLE}' ) - 1 ) AS "TARGET_TABLE_V" ;
\if ${ERROR} != 'false'
 \q 12
\endif

;
```

TOP 子句

DSC支持迁移使用动态参数的TOP语句。Teradata的TOP子句在DWS中迁移为LIMIT。

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
;
```

SAMPLE 子句

Teradata的SAMPLE子句在DWS中迁移为LIMIT。

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;
```

5.4.15.3 UPDATE

DSC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;
```

5.4.15.4 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;
```

5.4.15.5 MERGE

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
                        )
;
```

5.4.15.6 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);
```

5.4.15.7 ACTIVITYCOUNT

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 ;
/
```

5.4.15.8 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 类型转换

输入

```
COALESCE( a.Snd_Tm ,TIMESTAMP '0001-01-01 00:00:00' )
```

输出

```
COALESCE( a.Snd_Tm , CAST('0001-01-01 00:00:00' AS TIMESTAMP) )
```

5.4.16 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**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
 ;
```

十六进制字符字面值

输入	输出
'CASE WHEN Nullable='Y' THEN '' ELSE ' NOT NULL' END || '0A'XC	CASE WHEN Nullable='Y' THEN '' ELSE ' NOT NULL' END || E'\x0A'

Hexadecimal Character literal value

输入	输出

'SELECT CASE WHEN Nullable='Y' THEN ''
ELSE NOT NULL
END || '0A'XC AS SP_DATA_DT
FROM tbl_table;
.IF ERRORCODE <> 0 THEN .QUIT 12
	DECLARE lv_mig_errorcode NUMBER (4) ;
	lv_mig_SP_DATA_DT TEXT ;
	BEGIN
	BEGIN
	SELECT
	STRING_AGG (
	CASE
	WHEN Nullable = 'Y' THEN ''
	ELSE NOT NULL
	END || E'\x0A' /* ????????? */
	,''
	) INTO
	lv_mig_SP_DATA_DT
	FROM
	tbl_table ;
	lv_mig_errorcode := 0 ;
	EXCEPTION
	WHEN OTHERS THEN lv_mig_errorcode := - 1 ;
	END ;
	IF lv_mig_errorcode <> 0 THEN RAISE EXCEPTION
	'12' ;
	END IF ;
	END ;
	/

TRIM 含 INT 类型转换

输入	输出
TRIM(columnlength (INT))	TRIM(mig_td_ext.mig_fn_castasint(columnlength))

5.4.17 BTEQ 工具命令

DWS提供了一系列gsql元命令，可以等价替换常用的BTEQ工具命令。常用BTEQ命令的转换行为如下：

.QUIT | .EXIT | .RETURN

元命令\q [value]支持退出gsql程序，且可以通过value值指定退出码。.QUIT、.EXIT、.RETURN命令均可以通过\q等价转换。

输入	输出
.QUIT 0	\q 0
.EXIT	\q
.RETURN	\q

.LABEL 和 .GOTO

Teradata命令.LABEL用于创建标签，通常与.GOTO配对使用。.GOTO会跳过所有中间的BTEQ命令和SQL语句，指导到达指定标签位置，执行相应的恢复处理。

gsql元命令\goto LABEL ... \label LABEL可以等价实现无条件跳转。

输入	输出
.IF CHECK_PK='' THEN .GOTO NOCHECK ${CHECK_PK}; .LABEL NOCHECK .QUIT 0	\if ${CHECK_PK} == '' \goto NOCHECK \endif ${CHECK_PK} \label NOCHECK \q 0

.RUN FILE

执行指定文件的SQL请求，可等价转换成gsql元命令\i。

输入	输出
.RUN FILE=sampfile	\i 'sampfile'

.EXPORT FILE

将SQL语句执行结果导出到指定文件，可等价转换成gsql元命令\o。

输入	输出
.EXPORT REPORT FILE=resultfile	\o 'resultfile'

..SET ..END

设置变量记录指定值，可以是单行命令，也可以是以..END结束的多行命令，通过SELECT查询指定变量名或\set命令转换。

输入	输出
.SET SEPARATOR ' | '	\set SEPARATOR '|'
..SET NAME 'Jack' ..END	SELECT 'Jack' AS "NAME";

.IF

.IF是重要的流程控制命令之一，用于有条件地执行输入脚本的部分内容。可以是单分支命令，与THEN子句配对使用；也可以是多分支命令，多分支IF结构允许嵌套多层，但每层必须以IF命令开始，以ENDIF命令结束。

gsql提供流控元命令\if ... \else ... \endif可以等价转换此BTEQ命令。

输入	输出
.IF ERRORCODE <> 0 THEN .QUIT 12;	\if ${ERROR} != 'false' \q 12 \endif

..FOR

循环控制命令中，在满足循环条件时，可以不断执行循环体的脚本，直到.QUIT退出循环。DWS提供\for ... \loop ... \exit-for ... \end-for结构命令实现循环逻辑。

输入	输出
..IF ACTIVITYCOUNT > 0 THEN ..FOR SEL SqlStr AS V_SqlStr FROM ${ ETL_DATA}.TB_DWDATA_UPDATE WHERE JobName = '${JOB_NAME}' AND TXDATE = ${ TX_DATE} - 19000000 ORDER BY ExcuteSeq ASC; ..DO ${V_SqlStr} .IF ERRORCODE<>0 THEN .QUIT 12 ..END-FOR ..END-IF	\if ${ERROR} != 'false' \q 12 \endif \if ${ACTIVITYCOUNT} != 0 \for SELECT SqlStr AS V_SqlStr FROM ${ETL_DATA}.TB_DWDATA_UPDATE WHERE JobName = '${JOB_NAME}' AND TXDATE = to_date($ {TX_DATE} ,'yyyymmdd') ORDER BY ExcuteSeq ASC \loop ${V_SqlStr} ; \if ${ERROR} != 'false' \q 12 \endif \end-for \endif

5.4.18 Teradata 格式

以 YYYYMMDD 格式输入的日期

输入	输出
SELECT 1 FROM tb_dt_fmtyyyymmdd WHERE JobName ='${JOB_NAME}' AND TXDATE = ${TX_DATE} - 19000000;	SELECT 1 FROM tb_dt_fmtyyyymmdd WHERE JobName ='${JOB_NAME}' AND TXDATE = TO_DATE(${TX_DATE}, 'YYYYMMDD');

以 YYYYDDD 格式输入日期

输入	输出
REPLACE VIEW SC.VIEW_1 (col_1) LOCKING TABLE sc.tab FOR ACCESS AS SEL --tgt.col_1 is date type CAST(CAST(TGT.col_1 AS DATE FORMAT 'YYYYDDD') AS CHAR(7)) AS col_1 FROM sc.tab TGT ;	CREATE OR REPLACE VIEW SC.VIEW_1 (col_1) /*LOCKING TABLE sc.tab FOR ACCESS */ AS (SELECT /* tgt.col_1 is date type */ CAST(TO_DATE(TGT.col_1, 'YYYYDDD') AS CHAR(7)) AS col_1 FROM sc.tab TGT) ;

以#开头的列名

输入	输出
REPLACE VIEW SC.VIEW_1 (,col_1 ,#_col_2 ,#_col_3) LOCKING TABLE sc.tab FOR ACCESS AS SEL Tgt.col1 ,Tgt.#_col_2 ,Tgt.#_col_3 FROM sc.tab TGT ;	CREATE OR REPLACE VIEW SC.VIEW_1 (,col_1 ,"#_COL_2" ,"#_COL_3") /*LOCKING TABLE sc.tab FOR ACCESS */ AS (SELECT Tgt.col1 ,Tgt."#_COL_2" ,Tgt."#_COL_3" FROM sc.tab TGT) ;

类型转换时优先执行数据库操作

输入	输出
REPLACE VIEW SC.VIEW_1 (col_1) LOCKING TABLE sc.tab FOR ACCESS AS SEL (COALESCE(TRIM(TGT.col_1),'')) || '_' || (COALESCE(TRIM(TGT.col_1),'')) (CHAR(22)) AS col_1 FROM sc.tab TGT ;	CREATE OR REPLACE VIEW SC.VIEW_1 (col_1) /*LOCKING TABLE sc.tab FOR ACCESS */ AS (SELECT CAST((COALESCE(TRIM(TGT.col_1) ,'')) || '_' || (COALESCE(TRIM(TGT.col_1) ,'')) AS CHAR(22)) AS col_1 FROM sc.tab TGT) ;

5.4.19 系统视图

DSC将系统视图dbc.columnsV和dbc.IndicesV进行迁移，输出如下结果。

输入：

```
SELECT A.ColumnName
AS V_COLS
     ,A.columnname  || ' ' ||CASE WHEN columnType in ('CF','CV')
                                  THEN CASE WHEN columnType='CV' THEN 'VAR' ELSE ''
            END||'CHAR('||TRIM(columnlength (INT))||
               ') CHARACTER SET LATIN'||
                CASE WHEN UpperCaseFlag='N'
                  THEN ' NOT' ELSE ''
                END || ' CASESPECIFIC'
                            WHEN columnType='DA' THEN 'DATE'
                            WHEN columnType='TS' THEN 'TIMESTAMP(' || TRIM(DecimalFractionalDigits)||')'
                            WHEN columnType='AT' THEN 'TIME('|| TRIM(DecimalFractionalDigits)||')'
                            WHEN columnType='I' THEN 'INTEGER'
                            WHEN columnType='I1' THEN 'BYTEINT'
                            WHEN columnType='I2' THEN 'SMALLINT'
                            WHEN columnType='I8' THEN 'BIGINT'
                            WHEN columnType='D' THEN 'DECIMAL('||TRIM(DecimalTotalDigits)||','||
TRIM(DecimalFractionalDigits)||')'
                            ELSE 'Unknown'
                        END||CASE WHEN Nullable='Y'
       THEN '' ELSE ' NOT NULL' END||'0A'XC
AS V_ColT           -      ,B.ColumnName
AS V_PICol
 FROM dbc.columnsV A LEFT JOIN dbc.IndicesV B
   ON A.columnName = B.columnName AND B.IndexType IN ('Q','P')
  AND B.DatabaseName = '${V_TDDLDB}'  AND B.tablename='${TARGET_TABLE}'
WHERE A.databasename='${V_TDDLDB}' AND A.tablename = '${TARGET_TABLE}'
  AND A.columnname NOT IN
( 'ETL_JOB_NAME'                                                                                ,'ETL_TX_DATE'
                        ,'ETL_PROC_DATE'
                        )
 ORDER BY A.columnid;
```

输出：

```
DECLARE lv_mig_V_COLS   TEXT;
        lv_mig_V_ColT        TEXT;
        lv_mig_V_PICol       TEXT;
BEGIN
SELECT STRING_AGG(A.ColumnName, ',')
     , STRING_AGG(A.columnname  || ' ' ||CASE WHEN columnType in ('CF','CV')
                                  THEN CASE WHEN columnType='CV' THEN 'VAR' ELSE ''
            END||'CHAR('||TRIM(mig_td_ext.mig_fn_castasint(columnlength))||
               ') /*CHARACTER SET LATIN*/'||
                CASE WHEN UpperCaseFlag='N'
                  THEN ' NOT' ELSE ''
                END || ' /*CASESPECIFIC*/'
                            WHEN columnType='DA' THEN 'DATE'
                            WHEN columnType='TS' THEN 'TIMESTAMP(' || TRIM(DecimalFractionalDigits)||')'
                            WHEN columnType='AT' THEN 'TIME('|| TRIM(DecimalFractionalDigits)||')'
                            WHEN columnType='I' THEN 'INTEGER'
                            WHEN columnType='I1' THEN 'BYTEINT'
                            WHEN columnType='I2' THEN 'SMALLINT'
                            WHEN columnType='I8' THEN 'BIGINT'
                            WHEN columnType='D' THEN 'DECIMAL('||TRIM(DecimalTotalDigits)||','||
TRIM(DecimalFractionalDigits)||')'
                            ELSE 'Unknown'
                        END||CASE WHEN Nullable='Y'
       THEN '' ELSE ' NOT NULL' END||E'\x0A', ',')                , STRING_AGG(B.ColumnName, ',')
INTO lv_mig_V_COLS, lv_mig_V_ColT, lv_mig_V_PICol
FROM mig_td_ext.vw_td_dbc_columnsV A LEFT JOIN mig_td_ext.vw_td_dbc_IndicesV B
   ON A.columnName = B.columnName AND B.IndexType IN ('Q','P')
  AND B.DatabaseName = 'public'  AND B.tablename='emp2'
```

```
WHERE A.databasename='public' AND A.tablename = 'emp2';
-- ORDER BY A.columnid;
END;
/
```

5.5 MySQL 语法迁移

5.5.1 支持的关键词和特性

DSC支持迁移的MySQL关键字和特性如表5-27所示。

- “版本”列代表初次支持该关键字/特性的DWS集群版本。
- “备注”列包含该特性的配置参数，可用于自定义迁移工具如何迁移相应的关键字/特性。

表 5-27

章节	对象->关键词/特性	版本	备注
数据类型	数字类型	8.0.0	可配置： table.orientation table.type table.compress.mode table.compress.row table.compress.column table.compress.level
	日期和时间类型	8.0.0	
	字符串类型	8.0.0	
	空间数据类型	8.0.0	
	大对象类型	8.0.0	
	集合类型	8.0.0	
	布尔类型	8.0.0	
	二进制类型	8.0.0	
	JSON类型	8.0.0	
函数和表达式	类型对照表	8.0.0	-

章节	对象->关键词/特性		版本	备注
表（可选参数、操作）	表（可选参数、操作）	• AUTO_INCREMENT • AVG_ROW_LENGTH • CHARSET • CHECKSUM • COLLATE • COMMENT • CONNECTION • DELAY_KEY_WRITE • DIRECTORY • ENGINE • KEY_BLOCK_SIZE • INSERT_METHOD • MAX_ROWS • MIN_ROWS • PACK_KEYS • PASSWORD • ROW_FORMAT • STATS_AUTO_RECALC • STATS_PERSISTENT • STATS_SAMPLE_PAGES • UNION • LIKE 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	8.0.0	- -

章节	对象->关键词/特性		版本	备注
		• TRUNCATE 删除表 • LOCK • RENAME 重命名表名		
	索引	• 唯一索引 • 普通索引和前缀索引 • HASH索引 • BTREE索引 • SPATIAL空间索引 • FULLTEXT全文索引 • 删除索引 • 索引重命名	8.0.0	-
	注释		8.0.0	-
	数据库		8.0.0	-
数据操作语句（DML）	• 引号 • INTERVAL • 除法表达式 • group by转换 • ROLLUP		8.0.0	-
	INSERT	• HIGH_PRIORITY • LOW_PRIORITY • PARTITION • DELAYED • IGNORE • VALUES • ON DUPLICATE KEY UPDATE • SET	8.0.0	-

章节	对象->关键词/特性		版本	备注
	UPDATE	• LOW_PRIORITY • ORDER BY • LIMIT • IGNORE	8.0.0	-
	REPLACE	• LOW_PRIORITY • PARTITION • DELAYED • VALUES • SET	8.0.0	-
事务管理和数据库管理	事务管理	• TRANSACTION • LOCK	8.0.0	-
	数据库管理		8.0.0	-

5.5.2 数据类型

5.5.2.1 数字类型

概述

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。下表列出了MySQL数字类型到DWS的转换示例。

类型对照

表 5-28 数字类型对照表

MySQL数字类型	MySQL INPUT	DWS OUTPUT
DEC	DEC DEC[(M[,D])] [UNSIGNED] [ZEROFILL]	DECIMAL DECIMAL[(M[,D])]
DECIMAL	DECIMAL[(M[,D])] [UNSIGNED] [ZEROFILL]	DECIMAL[(M[,D])]

MySQL数字类型	MySQL INPUT	DWS OUTPUT
DOUBLE PRECISION	DOUBLE PRECISION DOUBLE PRECISION [(M[,D])] [UNSIGNED] [ZEROFILL]	DOUBLE PRECISION DOUBLE PRECISION
DOUBLE	DOUBLE[(M[,D])] [UNSIGNED] [ZEROFILL]	DOUBLE PRECISION
FIXED	FIXED FIXED[(M[,D])] [UNSIGNED] [ZEROFILL]	DECIMAL DECIMAL[(M[,D])]
FLOAT	FLOAT FLOAT [(M[,D])] [UNSIGNED] [ZEROFILL] FLOAT(p) [UNSIGNED] [ZEROFILL]	REAL REAL REAL
INT	INT INT(p) [UNSIGNED] [ZEROFILL]	INTEGER INTEGER(p)
INTEGER	INTEGER INTEGER(p) [UNSIGNED] [ZEROFILL]	INTEGER INTEGER(p)
MEDIUMINT	MEDIUMINT MEDIUMINT(p) [UNSIGNED] [ZEROFILL]	INTEGER INTEGER(p)
NUMERIC	NUMERIC NUMERIC [(M[,D])] [UNSIGNED] [ZEROFILL]	DECIMAL DECIMAL[(M[,D])]
REAL	REAL[(M[,D])]	REAL/DOUBLE PRECISION
SMALLINT	SMALLINT SMALLINT(p) [UNSIGNED] [ZEROFILL]	SMALLINT
TINYINT	TINYINT TINYINT(n) TINYINT(n) ZEROFILL TINYINT(n) UNSIGNED ZEROFILL	SMALLINT SMALLINT SMALLINT TINYINT

说明

- TINYINT类型做转换时，如果存在无符号类型(UNSIGNED)修饰则转换为TINYINT，否则转换为SMALLINT。
- REAL类型做转换时，默认转换为DOUBLE PRECISION，如果配置文件(features-mysql.properties)中table.database.realAsFlag标志为true时(默认false)，转换为REAL

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 IF NOT EXISTS "public"."runoob_datatype_test"
(
  "datatype_1" SMALLINT,
  "datatype_2" SMALLINT,
  "datatype_3" SMALLINT,
  "datatype_4" TINYINT,
  "datatype_5" SMALLINT
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_1");
```

5.5.2.2 日期和时间类型

概述

本节介绍如下日期和时间类型：DATETIME、TIME、TIMESTAMP、YEAR。DWS不支持以上类型，DSC工具将会对其转换。

类型对照

表 5-29 日期和时间类型对照表

MySQL日期时间类型	MySQL INPUT	DWS OUTPUT
DATETIME	DATETIME[(fsp)]	TIMESTAMP[(fsp)] WITHOUT TIME ZONE
TIME	TIME[(fsp)]	TIME[(fsp)] WITHOUT TIME ZONE
TIMESTAMP	TIMESTAMP[(fsp)]	TIMESTAMP[(fsp)] WITH TIME ZONE
YEAR	YEAR[(4)]	SMALLINT(4)

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 IF NOT EXISTS "public"."runoob_datatype_test"
(
  "datatype_1" TIMESTAMP WITHOUT TIME ZONE,
  "datatype_2" TIMESTAMP(0) WITHOUT TIME ZONE,
  "datatype_3" TIMESTAMP(6) WITHOUT TIME ZONE,
  "datatype_4" TIMESTAMP WITHOUT TIME ZONE DEFAULT NULL,
  "datatype_5" TIMESTAMP WITHOUT TIME ZONE DEFAULT '2018-10-12 15:27:33.999999'
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_1");
```

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 IF NOT EXISTS "public"."runoob_datatype_test"
(
  "datatype_1" TIME WITHOUT TIME ZONE DEFAULT '20:58:10',
  "datatype_2" TIME(3) WITHOUT TIME ZONE DEFAULT '20:58:10',
  "datatype_3" TIME(6) WITHOUT TIME ZONE DEFAULT '20:58:10',
  "datatype_4" TIME(6) WITHOUT TIME ZONE DEFAULT '2018-10-11 20:00:00',
  "datatype_5" TIME(6) WITHOUT TIME ZONE DEFAULT '20:58:10.01234',
  "datatype_6" TIME(6) WITHOUT TIME ZONE DEFAULT '2018-10-11 20:00:00.01234',
  "datatype_7" TIME WITHOUT TIME ZONE DEFAULT NULL,
  "datatype_8" TIME(6) WITHOUT TIME ZONE DEFAULT NULL,
  PRIMARY KEY ("datatype_1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_1");
```

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 IF NOT EXISTS "public"."runoob_datatype_test"
(
  "datatype_1" TIMESTAMP WITH TIME ZONE,
  "datetype_4" TIMESTAMP WITH TIME ZONE DEFAULT '2018-10-12 15:27:33',
  "datetype_5" TIMESTAMP WITH TIME ZONE DEFAULT '2018-10-12 15:27:33.999999',
  "datetype_6" TIMESTAMP WITH TIME ZONE DEFAULT '2018-10-12 15:27:33',
  "datetype_7" TIMESTAMP WITH TIME ZONE DEFAULT '2018-10-12 15:27:33',
  "datatype_8" TIMESTAMP(0) WITH TIME ZONE DEFAULT '2018-10-12 15:27:33',
  "datetype_9" TIMESTAMP(6) WITH TIME ZONE DEFAULT '2018-10-12 15:27:33'
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_1");
```

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 IF NOT EXISTS "public"."runoob_datatype_test"
(
  "datatype_1" SMALLINT,
  "datatype_2" SMALLINT,
  "datatype_3" VARCHAR(4) DEFAULT '2018',
  "datatype_4" TIME WITHOUT TIME ZONE DEFAULT NULL
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_1");
```

5.5.2.3 字符串类型

概述

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。DSC工具支持以下列出类型转换。

类型对照

表 5-30 字符串类型对照表

MySQL字符串类型	MySQL INPUT	DWS OUTPUT
CHAR	CHAR[(0)] CHAR[(n)]	VARCHAR[(1)] VARCHAR[(4n)]
CHARACTER	CHARACTER[(0)] CHARACTER[(n)]	CHAR[(1)] CHAR[(4n)]

MySQL字符串类型	MySQL INPUT	DWS OUTPUT
NCHAR	NCHAR[(0)] NCHAR[(n)]	CHAR[(1)] CHAR[(4n)]
LONGTEXT	LONGTEXT	TEXT
MEDIUMTEXT	MEDIUMTEXT	TEXT
TEXT	TEXT	TEXT
TINYTEXT	TINYTEXT	TEXT
VARCHAR	VARCHAR[(0)] VARCHAR[(n)]	VARCHAR[(1)] VARCHAR[(4n)]
NVARCHAR	NVARCHAR[(0)] NVARCHAR[(n)]	VARCHAR[(1)] VARCHAR[(4n)]
CHARACTE VARYING	CHARACTE VARYING	VARCHAR

说明

- CHARACTER/NCHAR进行转换时，如果其精度小于等于0时，转换后为CHAR(1)，如果精度大于0，则转换为CHAR类型4倍的精度大小。
- VARCHAR/NVARCHAR/CHAR进行转换时，如果其精度小于等于0时，转换后为VARCHAR(1)，如果精度大于0，则转换为VARCHAR类型4倍的精度大小。

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 IF NOT EXISTS "public"."runoob_datatype_test"
(
  "datatype_1" CHAR NOT NULL,
  "datatype_2" CHAR(1) NOT NULL,
  "datatype_3" CHAR(1020) NOT NULL
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_1");
```

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 IF NOT EXISTS "public"."runoob_datatype_test"
(
  "datatype_1" TEXT,
  "datatype_2" TEXT,
  "datatype_3" TEXT,
  "datatype_4" TEXT
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_1");
```

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 IF NOT EXISTS "public"."runoob_datatype_test"
(
  "datatype_1" VARCHAR(1),
  "datatype_2" VARCHAR(7380)
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_1");
```

5.5.2.4 空间数据类型

概述

MySQL具有对应于OpenGIS类的空间数据类型。DSC工具支持以下列出类型转换。

类型对照

表 5-31 空间数据类型对照表

MySQL空间数据类型	MySQL INPUT	DWS OUTPUT
GEOMETRY	GEOMETRY	GEOMETRY
POINT	POINT	POINT
LINESTRING	LINESTRING	POLYGON
POLYGON	POLYGON	POLYGON
MULTIPOINT	MULTIPOINT	BOX
MULTILINESTRING	MULTILINESTRING	BOX
MULTIPOLYGON	MULTIPOLYGON	POLYGON

MySQL空间数据类型	MySQL INPUT	DWS OUTPUT
GEOMETRYCOLLECTION	GEOMETRYCOLLECTION	CIRCLE

说明

- 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 ）限制集合成员像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" GEOMETRY NOT NULL,
  "point_1" POINT NOT NULL,
  "linestring_1" POLYGON NOT NULL,
  "polygon_1" POLYGON NOT NULL,
  "multipoint_1" BOX NOT NULL,
  "multilinestring_1" BOX NOT NULL,
  "multipolygon_1" POLYGON NOT NULL,
  "geometrycollection_1" CIRCLE NOT NULL,
  PRIMARY KEY ("id")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("id");
```

5.5.2.5 大对象类型

概述

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各自可以容纳的值的最大长度不同。DSC工具支持以下列出类型转换。

说明

BLOB类型可以存储图片，列存储不支持BLOB。

类型对照

表 5-32 大对象类型对照表

MySQL大对象类型	MySQL INPUT	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 IF NOT EXISTS "public"."runoob_datatype_test"
(
  "datatype_1" BIGINT,
  "datatype_2" BLOB,
  "datatype_3" BLOB,
  "datatype_4" BLOB,
  "datatype_5" BLOB
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_1");
```

5.5.2.6 集合类型

概述

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，每个值必须从创建表时指定的允许值列表中选择。

类型对照

表 5-33 集合类型对照表

MySQL集合类型	MySQL INPUT	DWS OUTPUT
ENUM	ENUM	VARCHAR
SET	SET	VARCHAR

说明

- 对于ENUM的类型转换，将转换为VARCHAR类型，精度大小为枚举值中最长字段长度的4倍，并使用CHECK()函数确保输入枚举值的正确性
- 对于SET的类型转换，将转换为VARCHAR类型，精度大小为各枚举值字段长度与分隔符数量和的4倍。

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 IF NOT EXISTS "public"."runoob_datatype_test"
(
 "id" INTEGER(2) PRIMARY KEY,
 "datatype_17" VARCHAR(20) CHECK (dataType_17 IN('dws-1','dws-2','dws-3','', null))
)
 WITH ( ORIENTATION = ROW, COMPRESSION = NO )
 NOCOMPRESS
 DISTRIBUTE BY HASH ("id");
```

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 IF NOT EXISTS "public"."runoob_tbl_test"
(
 "datatype_18" VARCHAR(68)
)
 WITH ( ORIENTATION = ROW, COMPRESSION = NO )
 NOCOMPRESS
 DISTRIBUTE BY HASH ("datatype_18");
```

5.5.2.7 布尔类型

概述

MySQL 支持两种布尔写法：BOOL、BOOLEAN。DSC工具支持以下列出类型转换。

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 IF NOT EXISTS "public"."runoob_datatype_test"
(
 "datatype_1" INTEGER,
 "datatype_2" BOOLEAN,
 "datatype_3" BOOLEAN
)
```

```
WITH ( ORIENTATION = ROW, COMPRESSION = NO )
NOCOMPRESS
DISTRIBUTE BY HASH ("datatype_1");
```

5.5.2.8 二进制类型

概述

- 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。
- MySQL BINARY和VARBINARY 类似CHAR并且VARCHAR，只不过它们包含二进制字符串。

类型对照

表 5-34 二进制类型对照表

MySQL二进制类型	MySQL INPUT	DWS OUTPUT
BIT[(M)]	BIT[(M)]	BIT[(M)]
BINARY[(M)]	BINARY[(M)]	BYTEA
CHAR BYTE[(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 IF NOT EXISTS "public"."runoob_datatype_test"
(
  "datatype_1" INTEGER,
  "datatype_2" BIT(1),
  "datatype_3" BIT(64)
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_1");
```

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 IF NOT EXISTS "public"."runoob_datatype_test"
(
```

```
  "datatype_1" INTEGER,
  "datatype_2" BYTEA,
  "datatype_3" BYTEA,
  "datatype_4" BYTEA,
  "datatype_5" BYTEA,
  "datatype_6" BYTEA
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_1");
```

5.5.2.9 JSON 类型

概述

JSON数据类型可以用来存储JSON（JavaScript Object Notation）数据，DSC工具支持以下列出类型转换。

类型对照

输入示例JSON

```
CREATE TABLE IF NOT EXISTS `runoob_dataType_test`(
    `dataType_1` INT,
    `dataType_2` VARCHAR,
    `dataType_3` JSON
);
ALTER TABLE `runoob_dataType_test` ADD COLUMN `dataType_4` JSON NOT NULL;
ALTER TABLE `runoob_dataType_test` CHANGE COLUMN `dataType_4` `dataType_5` JSON NOT NULL;
```

输出示例

```
CREATE TABLE IF NOT EXISTS "public"."runoob_datatype_test"
(
  "datatype_1" INTEGER,
  "datatype_2" VARCHAR,
  "datatype_3" JSONB
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_1");
ALTER TABLE
  "public"."runoob_datatype_test"
ADD
  COLUMN "datatype_4" JSONB;
ALTER TABLE
 "public"."runoob_datatype_test" CHANGE COLUMN "datatype_4" "datatype_5" JSONB;
```

5.5.3 函数和表达式

概述

由于MySQL中的函数与表达式，在DWS中不存在或者存在一定的差异，DSC工具会根据DWS的支持情况做相应迁移。（兼容ADB for MySQL的语法支持）

类型对照

表 5-35 类型对照表

MySQL/ADB函数类型	描述	MySQL INPUT	DWS OUTPUT
CONVERT	将值转换为指定的数据类型或字符集	CONVERT(A, B)	CAST(A AS B)
CURDATE	当前日期，是CURDATE()的同义词	CURDATE/CURDATE()	CURRENT_DATE ()
GET_JSON_OBJECT	解析json字符串的一个字段	GET_JSON_OBJECT(column, '$.obj.arg') GET_JSON_OBJECT(column, '$[i]')	JSON_EXTRACT(column, 'obj', 'arg') JSON_ARRAY_ELEMENT(column, 0)
JSON_EXTRACT	查询json中某个字段的值	JSON_EXTRACT(column, '$.obj')	JSON_EXTRACT(column, 'obj')
REGEXP	模糊匹配	REGEXP A NOT REGEXP A	~ A !~ A
UUID	生成唯一值(序列号)	UUID	SYS_GUID
SPLIT()[]	根据delimiter分隔string返回生成的第field个子字符串（从出现第一个delimiter的text为基础）	SPLIT(string text, delimiter text)[field int]	SPLIT(string text, delimiter text)[field int]
RAND	获取0.0到1.0之间的随机数	RAND()	RANDOM()
SLICE	字符串切割，以第一个参数为分隔符，链接第二个以后的所有参数	SLICE()	CONCAT_WS(sep text, str"any" [, str"any" [, ...]])
TRY_CAST	类型转换函数，将x转换成y指定的类型	TRY_CAST(X AS Y)	CAST(X AS Y)

MySQL/ADB函数类型	描述	MySQL INPUT	DWS OUTPUT
CAST	类型转换函数，将x转换成y指定的类型 (ADB中cast函数第二个参数，强转数据类型可以为string和double，dws中没有对应类型，因此转换为varchar和double precision类型)	CAST(X AS Y)	CAST(X AS Y)

输入示例CONVERT

```
SELECT CONVERT (IFNULL (BUSINESS_ID, 0) , DECIMAL(18, 2)) FROM ACCOUNT;
```

输出示例

```
SELECT cast (ifnull (business_id, 0) AS decimal(18, 2))FROM account;
```

输入示例CURDATE

```
SELECT CURDATE;
SELECT CURDATE();
```

输出示例

```
SELECT CURRENT_DATE();
SELECT CURRENT_DATE();
```

输入示例GET_JSON_OBJECT

```
SELECT GET_JSON_OBJECT(COL_JSON, '$.STORE.BICYCLE.PRICE');

SELECT GET_JSON_OBJECT(COL_JSON, '$.STORE.FRUIT[0]');
```

输出示例

```
SELECT
  JSON_EXTRACT_PATH(COL_JSON, 'STORE', 'BICYCLE', 'PRICE');
SELECT
  JSON_ARRAY_ELEMENT(JSON_EXTRACT_PATH(COL_JSON, 'STORE', 'FRUIT'), 0);
```

输入示例JSON_EXTRACT

```
SELECT JSON_EXTRACT(EVENT_ATTR,'$.TOPIC_ID');
```

输出示例

```
SELECT JSON_EXTRACT_PATH(EVENT_ATTR, 'TOPIC_ID');
```

输入示例REGEXP

```
SELECT * FROM USERS WHERE NAME NOT REGEXP '^王';
SELECT * FROM USERS WHERE TEL REGEXP '[^4-5]{11}';
```

输出示例

```
SELECT * FROM USERS WHERE NAME !~ '^王';
SELECT * FROM USERS WHERE TEL ~ '[^4-5]{11}';
```

输入示例UUID

```
SELECT CURDATE(str1), UUID(str2, str3) FROM T1;
SELECT A FROM B WHERE uuid() > 2;
```

输出示例

```
SELECT current_date (str1),sys_guid (str2, str3) FROM T1;
SELECT A FROM  B WHERE sys_guid () > 2;
```

输入示例SPLIT()[]

```
SELECT split('a-b-c-d-e', '-')[4];
```

输出示例

```
SELECT split_part('a-b-c-d-e', '-')[4];
```

输入示例RAND

```
SELECT rand();
```

输出示例

```
SELECT random ();
```

输入示例SLICE

```
SELECT slice(split('2021_08_01','_'),1,3) from dual;
```

输出示例

```
SELECT
  concat_ws(
    split_part('2021_08_01', '_', 1),
    split_part('2021_08_01', '_', 2),
    split_part('2021_08_01', '_', 3)
  )
FROM
  dual;
```

输入示例TRY_CAST

```
select * from ods_pub where try_cast(pay_time AS timestamp) >= 1;
select try_cast(pay_time as timestamp) from obs_pub;
```

输出示例

```
SELECT * FROM ods_pub WHERE cast (pay_time AS timestamp) >= 1;
SELECT cast (pay_time as timestamp) FROM obs_pub;
```

输入示例CAST

```
select cast(ifnull(c1, 0) as string) from t1;
select cast(ifnull(c1, 0) as varchar) from t1;
select cast(ifnull(c1, 0) as double) from t1;
select cast(ifnull(c1, 0) as int) from t1;
```

输出示例

```
SELECT cast (ifnull (c1, 0) as varchar) FROM t1;
SELECT cast (ifnull (c1, 0) as varchar) FROM t1;
SELECT cast (ifnull (c1, 0) as double precision) FROM t1;
SELECT cast (ifnull (c1, 0) as int) FROM t1;
```

5.5.4 表（可选参数、操作）

本节主要介绍表（可选参数、操作）的迁移语法。迁移语法决定了关键字/功能的迁移方式。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。

5.5.4.1 ALGORITHM

MySQL扩展了对ALTER TABLE … ALGORITHM=INSTANT的支持：用户可以在表的任何位置即时添加列、即时删除列、添加列时评估行大小限制。

DWS不支持此属性，并在迁移过程中被DSC删除。

输入示例

```
ALTER TABLE runoob_alter_test ALGORITHM=DEFAULT;
ALTER TABLE runoob_alter_test ALGORITHM=INPLACE;
ALTER TABLE runoob_alter_test ALGORITHM=COPY;
ALTER TABLE runoob_alter_test ADD COLUMN COL_18 VARCHAR(64)  DEFAULT '00', ALGORITHM=INSTANT;
ALTER TABLE runoob_alter_test MODIFY COLUMN dataType7 BIGINT, ALGORITHM=COPY;
ALTER TABLE `runoob_alter_test` ALGORITHM=DEFAULT, ALGORITHM=INPLACE, ALGORITHM=COPY;
ALTER TABLE `runoob_alter_test` ADD COLUMN dataType11 INT, ALGORITHM=DEFAULT,
ALGORITHM=INPLACE, ALGORITHM=COPY;
ALTER TABLE runoob_alter_test CHANGE COLUMN dataType11 dataType12
SMALLINT ,ALGORITHM=INPLACE, ALGORITHM=COPY;
ALTER TABLE runoob_alter_test ALGORITHM=INPLACE, ALGORITHM=COPY, DROP COLUMN dataType12;
```

输出示例

```
ALTER TABLE
  "public"."runoob_alter_test"
ADD
  COLUMN "col_18" VARCHAR(256) DEFAULT '00';
ALTER TABLE
  "public"."runoob_alter_test" MODIFY "datatype7" BIGINT NULL DEFAULT NULL;
ALTER TABLE
  "public"."runoob_alter_test"
ADD
  COLUMN "datatype11" INTEGER;
ALTER TABLE
  "public"."runoob_alter_test" CHANGE COLUMN "datatype11" "datatype12" SMALLINT NULL DEFAULT
NULL;
ALTER TABLE
  "public"."runoob_alter_test" DROP COLUMN "datatype12" RESTRICT;
DROP TABLE IF EXISTS "public"."runoob_alter_test";
```

5.5.4.2 ALTER TABLE RENAME

DWS不支持rename子句包含schema名，因此DSC工具只支持同schema下的rename。同schema下rename，转换结果去掉子句schema，跨schema的rename报错。

输入示例

```
ALTER TABLE `shce1`.`t1` rename to `t2`;
ALTER TABLE `shce1`.`t1` rename to t2;
ALTER TABLE `charge_data`.`group_shengfen2022` RENAME `charge_data`.`group_shengfen2022_jiu`;
ALTER TABLE `charge_data`.`group_shengfen2022` RENAME `charge_data`.`group_shengfen2022_jiu`,
RENAME `charge_data`.`group_shengfen2023_jiu`, RENAME `charge_data`.`group_shengfen2024_jiu`;
```

输出示例

```
ALTER TABLE "shce1"."t1" RENAME TO "t2";
ALTER TABLE "shce1"."t1" RENAME TO "t2";
ALTER TABLE "charge_data"."group_shengfen2022" RENAME TO "group_shengfen2022_jiu";
ALTER TABLE "charge_data"."group_shengfen2022" RENAME TO "group_shengfen2022_jiu", RENAME TO
"group_shengfen2023_jiu", RENAME TO "group_shengfen2024_jiu";
```

5.5.4.3 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。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，转换如下表。

表 5-36 数据类型转换

MySQL数字类型	MySQL INPUT	DWS OUTPUT
TINYINT	TINYINT	SMALLSERIAL
SMALLINT	SMALLINT UNSIGNED SMALLINT	SERIAL SMALLSERIAL
DOUBLE/FLOAT	DOUBLE/FLOAT	BIGSERIAL
INT/INTEGER	INT/INTEGER UNSIGNED INT/INTEGER	BIGSERIAL SERIAL
BIGINT/SERIAL	BIGINT/SERIAL	BIGSERIAL

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(400) NOT NULL DEFAULT '',
  PRIMARY KEY ("job_sche_id")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("job_sche_id");
```

此外，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 IF NOT EXISTS "public"."runoob_alter_test"
(
  "datatype1" SERIAL NOT NULL,
  "datatype2" REAL,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");
```

5.5.4.4 AVG_ROW_LENGTH

在MySQL中，AVG_ROW_LENGTH表示平均每行的长度。 DWS不支持此属性，并在迁移过程中被DSC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)
)AVG_ROW_LENGTH=10000;
```

输出示例

```
CREATE TABLE "public"."runoob_tbl_test"
(
  "runoob_id" VARCHAR(120),
  "runoob_title" VARCHAR(400) NOT NULL,
  "runoob_author" VARCHAR(160) NOT NULL,
  "submission_date" VARCHAR(120)
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("runoob_id");
```

5.5.4.5 BLOCK_SIZE

在ADB中，指定列式存储中每个block存储的Value的个数，也是最小的IO单元。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

```
DROP TABLE IF EXISTS exists unsupport_parse_test;
CREATE TABLE `unsupport_parse_test` (
    `username` int,
    `update` timestamp not null default current_timestamp on update current_timestamp ,
     clustered key clustered_key(shopid ASC, datetype ASC)
)BLOCK_SIZE = 1024 index_ALL = 'y';
DROP TABLE IF EXISTS unsupport_parse_test;
```

输出示例

```
DROP TABLE IF EXISTS "public"."unsupport_parse_test";
CREATE TABLE "public"."unsupport_parse_test" (
  "username" INTEGER,
  "update" TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT CURRENT_TIMESTAMP
```

```
) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH ("username");
DROP TABLE IF EXISTS "public"."unsupport_parse_test";
```

5.5.4.6 CHARSET

CHARSET指定表的默认字符集。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(120),
  "runoob_title" VARCHAR(400) NOT NULL,
  "runoob_author" VARCHAR(160) NOT NULL,
  "submission_date" VARCHAR(120)
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("runoob_id");
```

5.5.4.7 CHECKSUM

在MySQL中，CHECKSUM表示对所有的行维护实时校验和。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" REAL,
  "datatype3" DOUBLE PRECISION,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");
```


5.5.4.8 CLUSTERED KEY

在ADB中，CLUSTERED KEY：聚集索引，定义表中的排序列，聚集索引中键值的逻辑顺序决定了表中相应行的物理顺序。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

```
DROP TABLE IF EXISTS unsupport_parse_test;
CREATE TABLE `unsupport_parse_test` (
    `username` int,
     clustered key clustered_key(shopid ASC, datetype ASC)
);
DROP TABLE IF EXISTS unsupport_parse_test;
```

输出示例

```
DROP TABLE IF EXISTS "public"."unsupport_parse_test";
CREATE TABLE "public"."unsupport_parse_test" (
  "username" INTEGER
) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH ("username");
DROP TABLE IF EXISTS "public"."unsupport_parse_test";
```

5.5.4.9 COLLATE

在MySQL中，COLLATE表示默认的数据库排序规则。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(120),
  "runoob_title" VARCHAR(400) NOT NULL,
  "runoob_author" VARCHAR(160) NOT NULL,
  "submission_date" VARCHAR(120)
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("runoob_id");
```

5.5.4.10 COMMENT

在MySQL中，COMMENT对表进行注释。DWS支持该属性修改表定义信息，DSC工具迁移时会添加额外的表属性信息。

输入示例

```
CREATE TABLE `public`.`runoob_alter_test`(
    `dataType1` int NOT NULL AUTO_INCREMENT,
    `dataType2` FLOAT(10,2) COMMENT 'dataType2列',
    PRIMARY KEY(`dataType1`)
) comment='表的注释';
```

```
ALTER TABLE `public`.`runoob_alter_test` COMMENT '修改后的表的注释';
ALTER TABLE `public`.`runoob_alter_test` ADD INDEX age_index(dataType2) COMMENT '索引';
```

输出示例

```
CREATE TABLE "public"."runoob_alter_test"
(
  "datatype1" SERIAL NOT NULL,
  "datatype2" REAL COMMENT 'dataType2列',
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1")
  COMMENT '表的注释';
ALTER TABLE "public"."runoob_alter_test" COMMENT '修改后的表的注释';
CREATE INDEX "age_index" ON "public"."runoob_alter_test" ("dataType2") COMMENT '索引';
```

5.5.4.11 CONNECTION

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" DOUBLE PRECISION,
  "datatype3" TEXT NOT NULL,
  "datatype4" SMALLINT NOT NULL DEFAULT '2018',
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");
```

5.5.4.12 DEFAULT

alter table增加列包含not null约束时，如果无default值，在MySQL中会插入默认值，而在DWS中插入到非空表时会报错，因此针对常见数据类型，alter table add column包含not null约束补充**默认default值（详情见表1）**。

输入示例

```
DROP TABLE IF EXISTS default_test_1;
CREATE TABLE default_test_1
```

```
(
    c1 tinyint,
    c2 smallint,
    c3 MEDIUMINT,
    c4 int,
    c5 bigint
);
INSERT INTO default_test_1
VALUES (1, 1, 1, 1, 1);
SELECT *
FROM default_test_1;
ALTER TABLE default_test_1 add COLUMN (c1_1 tinyint not null, c2_1 SMALLINT not null, c3_1 MEDIUMINT not null, c4_1 int not null, c5_1 BIGINT not null);
SELECT *
FROM default_test_1;
```

输出示例

```
DROP TABLE IF EXISTS "public"."default_test_1";
CREATE TABLE "public"."default_test_1" (
  "c1" SMALLINT,
  "c2" SMALLINT,
  "c3" INTEGER,
  "c4" INTEGER,
  "c5" BIGINT
) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH ("c1");
INSERT INTO
  "public"."default_test_1"
VALUES
  (1, 1, 1, 1, 1);
SELECT
  *
FROM
  default_test_1;
ALTER TABLE
  "public"."default_test_1"
ADD
  COLUMN "c1_1" SMALLINT NOT NULL DEFAULT 0,
ADD
  COLUMN "c2_1" SMALLINT NOT NULL DEFAULT 0,
ADD
  COLUMN "c3_1" INTEGER NOT NULL DEFAULT 0,
ADD
  COLUMN "c4_1" INTEGER NOT NULL DEFAULT 0,
ADD
  COLUMN "c5_1" BIGINT NOT NULL DEFAULT 0;
SELECT
  *
FROM
  default_test_1;
```

表 5-37

数据类型	MYSQL	DWS对应数据类型	转换结果
有符号整型	TINYINT	SMALLINT	0
	SMALLINT	SMALLINT	0
	MEDIUMINT	INT/INTEGER	0
	INT/INTEGER	INT/INTEGER	0
	BIGINT	BIGINT	0

数据类型	MYSQL	DWS对应数据类型	转换结果
无符号整型	TINYINT UNSIGNED	TINYINT	0
	SMALLINT UNSIGNED	INT	0
	MEDIUMINT UNSIGNED	INT	0
	INT UNSIGNED	BIGINT	0
	BIGINT UNSIGNED	NUMERIC	0
浮点型	FLOAT	REAL	0
	FLOAT UNSIGNED	REAL	0
	DOUBLE	DOUBLE PRECISION	0
	DOUBLE UNSIGNED	DOUBLE PRECISION	0
	DOUBLE PRECISION (DOUBLE)	DOUBLE PRECISION	0
	DOUBLE PRECISION [UNSIGNED] (DOUBLE)	DOUBLE PRECISION	0
	REAL（默认按double处理，设置REAL_AS_FLOAT时，按float处理）	REAL/DOUBLE PRECISION	0
	REAL [UNSIGNED]（默认按double处理，设置REAL_AS_FLOAT时，按float处理）	REAL/DOUBLE PRECISION	0
	NUMERIC/ DECIMAL	NUMERIC/ DECIMAL	0
布尔类型	BOOLEAN/BOOL	BOOLEAN	0
字符类型	CHAR/ CHARACTER	CHAR/ CHARACTER * 4	空字符串

数据类型	MYSQL	DWS对应数据类型	转换结果
	NCHAR	CHAR/ CHARACTER * 4	空字符串
	VARCHAR	VARCHAR * 4	空字符串
文本类型	TINYTEXT	TEXT	空字符串
	TEXT	TEXT	空字符串
	MEDIUMTEXT	TEXT	空字符串
	LONGTEXT	TEXT	空字符串
二进制类型	BINARY	BYTEA	空字符串
	VARBINARY	BYTEA	空字符串
	TINYBLOB	BYTEA(列存)/ BLOB(非列存)	空字符串
	BLOB	BYTEA(列存)/ BLOB(非列存)	空字符串
	MEDIUMBLOB	BYTEA(列存)/ BLOB(非列存)	空字符串
	LONGBLOB	BYTEA(列存)/ BLOB(非列存)	空字符串
	CHAR BYTE （ BINARY ）	BYTEA	空字符串
日期类型	DATE	DATE	1970-01-01
	TIME	TIME(P) WITHOUT TIME ZONE	00:00:00
	DATETIME	TIMESTAMP WITHOUT TIME ZONE	1970-01-01 00:00:00
	TIMESTAMP	TIMESTAMP(P) WITH TIME ZONE	1970-01-01 00:00:00
	datetime/ datetime(0)	datetime->timestamp with out time zone datetime(0)->timestamp(0) with out time zone	1970-01-01 00:00:00.xx(x个数与精度位数有关)
	YEAR	SMALLINT	0000

数据类型	MYSQL	DWS对应数据类型	转换结果
位串类型	BIT	BIT	DEFAULT 0:: BIT(x)
集合类型	ENUM	CHARACTER VARYING()	默认第一个元素
	SET	CHARACTER VARYING()	空字符串
自增序列	SERIAL（ BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE ）	CREATE SEQUENCE serial_name INCREMENT 2 START 1	不作处理
JSON类型	JSON	jsonb	不作处理
空间类型	point	point	不做处理
	polygon	polygon	不做处理
	geometry	不支持，原状返回	不做处理
	linestring	polygon	不做处理
	geometrycollection	不支持，原状返回	不做处理
	multipoint	box	不做处理
	multilinestring	box	不做处理
	multipolygon	polygon	不做处理

5.5.4.13 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。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(120),
  "runoob_title" VARCHAR(400) NOT NULL,
  "runoob_author" VARCHAR(160) NOT NULL,
  "submission_date" VARCHAR(120)
```

```
)
 WITH ( ORIENTATION = ROW, COMPRESSION = NO )
 NOCOMPRESS
 DISTRIBUTE BY HASH ("runoob_id");
```

5.5.4.14 DISTRIBUTE BY

在ADB中支持分布键，DSC迁移过程中会保留对应分布键。

输入示例

```
CREATE TABLE COPY_DI_DISTRIBUTOR_BUYER_CONTRIBUTION_RANKING_V2 (
    SHOP_ID VARCHAR ,
    DISTRIBUTOR_ID VARCHAR ,
    BUYER_ID VARCHAR ,
    DATE_TYPE BIGINT ,
    PRIMARY KEY (SHOP_ID,DISTRIBUTOR_ID,DATE_TYPE,BUYER_ID)
) DISTRIBUTE BY HASH(SHOP_ID,DISTRIBUTOR_ID,DATE_TYPE);
```

输出示例

```
CREATE TABLE "public"."copy_di_distributor_buyer_contribution_ranking_v2" (
  "shop_id" VARCHAR,
  "distributor_id" VARCHAR,
  "buyer_id" VARCHAR,
  "date_type" BIGINT,
  PRIMARY KEY (
    "shop_id",
    "distributor_id",
    "date_type",
    "buyer_id"
  )
) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH (SHOP_ID,
DISTRIBUTOR_ID, DATE_TYPE);
```

5.5.4.15 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。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" DOUBLE PRECISION,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");
```

```
CREATE TABLE "public"."runoob_tbl_test2"
(
  "datatype1" SERIAL NOT NULL,
  "datatype2" DOUBLE PRECISION,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");
```

5.5.4.16 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" DOUBLE PRECISION,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");

-- A.

-- B.

-- C.
```

5.5.4.17 FOREIGN_KEY_CHECKS

MySQL中的外键约束，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

```
SET foreign_key_checks = 0;
CREATE TABLE mall_order_dc (
    id bigint NOT NULL AUTO_INCREMENT,
    order_id varchar(50) NOT NULL,
```

```
    key order_id(order_id)
);
```

输出示例

```
CREATE TABLE "public"."mall_order_dc" (
  "id" BIGSERIAL NOT NULL,
  "order_id" VARCHAR(200) NOT NULL
) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH ("id");
CREATE INDEX "order_id" ON "public"."mall_order_dc" USING BTREE ("order_id");
```

5.5.4.18 IF NOT EXISTS

DSC支持转换**IF NOT EXISTS**关键字，迁移过程保留。

输入示例

```
DROP TABLE IF EXISTS `categories`;
CREATE TABLE IF NOT EXISTS `categories`
(
    `CategoryID` tinyint(5) unsigned NOT NULL PRIMARY KEY AUTO_INCREMENT,
    `CategoryName` varchar(15) CHARACTER SET utf8 COLLATE utf8_unicode_ci NOT NULL DEFAULT '' ,
    `Description` mediumtext CHARACTER SET utf8 COLLATE utf8_unicode_ci NOT NULL ,
    `Picture` varchar(50) CHARACTER SET utf8 COLLATE utf8_unicode_ci NOT NULL DEFAULT '',
    UNIQUE   (`CategoryID`)
)ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=latin1;
DROP TABLE IF EXISTS `categories`;
```

输出示例

```
DROP TABLE IF EXISTS "public"."categories";
CREATE TABLE IF NOT EXISTS "public"."categories" (
  "categoryid" SMALLSERIAL NOT NULL PRIMARY KEY,
  "categoryname" VARCHAR(60) NOT NULL DEFAULT '',
  "description" TEXT NOT NULL,
  "picture" VARCHAR(200) NOT NULL DEFAULT ''
) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH ("categoryid");
DROP TABLE IF EXISTS "public"."categories";
```

5.5.4.19 INDEX_ALL

在ADB中，创建全列索引index_all='Y'。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

```
DROP TABLE IF EXISTS unsupport_parse_test;
CREATE TABLE `unsupport_parse_test` (
    `username` int,
    `update` timestamp not null default current_timestamp on update current_timestamp ,
     clustered key clustered_key(shopid ASC, datetype ASC)
)index_ALL = 'Y';
DROP TABLE IF EXISTS unsupport_parse_test;
```

输出示例

```
DROP TABLE IF EXISTS "public"."unsupport_parse_test";
CREATE TABLE "public"."unsupport_parse_test" (
  "username" INTEGER,
  "update" TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT CURRENT_TIMESTAMP
) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH ("username");
DROP TABLE IF EXISTS "public"."unsupport_parse_test";
```

5.5.4.20 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" DOUBLE PRECISION,
  "datatype3" TEXT NOT NULL,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");
```

5.5.4.21 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）。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(120),
  "runoob_title" VARCHAR(400) NOT NULL,
  "runoob_author" VARCHAR(160) NOT NULL,
  "submission_date" VARCHAR(120)
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("runoob_id");
```

5.5.4.22 LOCK

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 IF NOT EXISTS "public"."runoob_alter_test"
(
  "datatype1" SERIAL NOT NULL,
  "datatype2" REAL,
  "datatype4" TEXT NOT NULL,
  "datatype5" SMALLINT NOT NULL DEFAULT '2018',
  "datatype6" TIMESTAMP WITHOUT TIME ZONE NOT NULL,
  "datatype7" CHAR(4) NOT NULL DEFAULT '',
  "datatype8" VARCHAR(200),
  "datatype9" VARCHAR(200) NOT NULL DEFAULT '',
  "datatype10" TIME WITHOUT TIME ZONE NOT NULL DEFAULT '10:20:59',
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");

-- A.

-- B.

-- C.

-- D.
```

5.5.4.23 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" DOUBLE PRECISION,
  "datatype3" TEXT NOT NULL,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");
```

5.5.4.24 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" DOUBLE PRECISION,
  "datatype3" TEXT NOT NULL,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");
```

5.5.4.25 PACK_KEYS

在MySQL中，PACK_KEYS表示MyISAM存储引擎中的压缩索引。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" DOUBLE PRECISION,
  "datatype3" TEXT NOT NULL,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");

--A

--B

--C
```

5.5.4.26 PARTITION BY

在MySQL中，PARTITION BY用于创建分区表。DWS目前仅对MySQL中的RANGE，LIST分区进行支持。

⚠ 注意

对于PARTITION BY的HASH分区，DSC暂不支持该特性的完整迁移，将其移除。对于表的当前功能暂时没有影响，性能方面可能存在些许差异。

PARTITION BY RANGE

输入示例

```
CREATE TABLE IF NOT EXISTS `runoob_tbl_part_test`(
  `runoob_id` INT NOT NULL,
  `runoob_title` VARCHAR(100) NOT NULL,
  `runoob_author` VARCHAR(40) NOT NULL,
  `submission_date` VARCHAR(30)
)ENGINE=InnoDB DEFAULT CHARSET=utf8
PARTITION BY RANGE (`runoob_id`)(
    PARTITION p0 VALUES LESS THAN(100),
    PARTITION p1 VALUES LESS THAN(200),
    PARTITION p2 VALUES LESS THAN(300),
    PARTITION p3 VALUES LESS THAN(400),
    PARTITION p4 VALUES LESS THAN(500),
    PARTITION p5 VALUES LESS THAN (MAXVALUE)
);

ALTER TABLE `runoob_tbl_part_test` ADD PARTITION (PARTITION p6 VALUES LESS THAN (600));

ALTER TABLE `runoob_tbl_part_test` ADD PARTITION (PARTITION p7 VALUES LESS THAN (700),PARTITION
p8 VALUES LESS THAN (800));
```

输出示例

```
CREATE TABLE IF NOT EXISTS "public"."runoob_tbl_part_test" (
  "runoob_id" INTEGER NOT NULL,
  "runoob_title" VARCHAR(400) NOT NULL,
  "runoob_author" VARCHAR(160) NOT NULL,
  "submission_date" VARCHAR(120)
) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH ("runoob_id")
PARTITION BY RANGE ("runoob_id") (
  PARTITION p0
  VALUES
    LESS THAN (100),
    PARTITION p1
  VALUES
    LESS THAN (200),
    PARTITION p2
  VALUES
    LESS THAN (300),
    PARTITION p3
  VALUES
    LESS THAN (400),
    PARTITION p4
  VALUES
    LESS THAN (500),
    PARTITION p5
  VALUES
    LESS THAN (MAXVALUE)
);
ALTER TABLE "public"."runoob_tbl_part_test" ADD PARTITION p6 VALUES LESS THAN (600);
ALTER TABLE "public"."runoob_tbl_part_test" ADD PARTITION p7 VALUES LESS THAN (700), ADD
PARTITION p8 VALUES LESS THAN (800);
```

PARTITION BY LIST

输入示例

```
CREATE TABLE IF NOT EXISTS `runoob_tbl_part_test`(
   `runoob_id` INT NOT NULL,
   `runoob_title` VARCHAR(100) NOT NULL,
   `runoob_author` VARCHAR(40) NOT NULL,
   `submission_date` VARCHAR(30),
   PRIMARY KEY (`runoob_id`)
   )ENGINE=InnoDB DEFAULT CHARSET=utf8
   PARTITION BY LIST (runoob_id)(
   PARTITION r0 VALUES IN (1, 5, 9, 13, 17, 21),
   PARTITION r1 VALUES IN (2, 6, 10, 14, 18, 22),
   PARTITION r2 VALUES IN (3, 7, 11, 15, 19, 23),
   PARTITION r3 VALUES IN (4, 8, 12, 16, 20, 24)
   );
ALTER TABLE `runoob_tbl_part_test` ADD PARTITION (PARTITION p5 VALUES IN(30, 40, 50, 60, 70, 80));
```

输出示例

```
CREATE TABLE IF NOT EXISTS "public"."runoob_tbl_part_test" (
  "runoob_id" INTEGER NOT NULL,
  "runoob_title" VARCHAR(400) NOT NULL,
  "runoob_author" VARCHAR(160) NOT NULL,
  "submission_date" VARCHAR(120),
  PRIMARY KEY ("runoob_id")
) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH ("runoob_id")
PARTITION BY LIST (runoob_id) (
  PARTITION r0
  VALUES
    (1, 5, 9, 13, 17, 21),
    PARTITION r1
  VALUES
    (2, 6, 10, 14, 18, 22),
    PARTITION r2
  VALUES
```

```
    (3, 7, 11, 15, 19, 23),
    PARTITION r3
  VALUES
    (4, 8, 12, 16, 20, 24)
);
ALTER TABLE "public"."runoob_tbl_part_test" ADD PARTITION p5 VALUES (30, 40, 50, 60, 70, 80);
```

5.5.4.27 PASSWORD

在MySQL中，PASSWORD表示用户密码。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" DOUBLE PRECISION,
  "datatype3" TEXT NOT NULL,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");
```

5.5.4.28 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。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" REAL,
  "datatype3" DOUBLE PRECISION,
  "datatype4" TEXT NOT NULL,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");

-- A.
ALTER TABLE "public"."runoob_alter_test" SET NOCOMPRESS;
ALTER TABLE "public"."runoob_alter_test" SET NOCOMPRESS;

-- B.

-- C.
ALTER TABLE "public"."runoob_alter_test" SET COMPRESS;
ALTER TABLE "public"."runoob_alter_test" SET COMPRESS;

-- D.

-- E.
```

5.5.4.29 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。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(120),
  "runoob_title" VARCHAR(400) NOT NULL,
```

```
  "runoob_author" VARCHAR(160) NOT NULL,
  "submission_date" VARCHAR(120)
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("runoob_id");

-- A.

-- B.

-- C.
```

5.5.4.30 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" DOUBLE PRECISION,
  "datatype3" TEXT NOT NULL,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");

-- A.

-- B.

-- C.
```

5.5.4.31 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" DOUBLE PRECISION,
  "datatype3" TEXT NOT NULL,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");
```

5.5.4.32 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=MyISAM UNION=(t1,t2) INSERT_METHOD=LAST;
```

输出示例

```
CREATE TABLE "public"."t1" (
    "a" SERIAL NOT NULL PRIMARY KEY,
    "message" CHAR(80)
)
WITH ( ORIENTATION = ROW, COMPRESSION = NO )
NOCOMPRESS
DISTRIBUTE BY HASH ("a");

CREATE TABLE "public"."t2" (
    a SERIAL NOT NULL PRIMARY KEY,
    message CHAR(80)
)
WITH ( ORIENTATION = ROW, COMPRESSION = NO )
NOCOMPRESS
DISTRIBUTE BY HASH ("a");

CREATE VIEW "public"."total"(a, message) AS
SELECT * FROM "public"."t1"
UNION ALL
SELECT * FROM "public"."t2";
```

5.5.4.33 WITH AS

WITH AS 在DWS中用于声明一个或多个可以在主查询中通过名字引用的子查询，相当于临时表。DSC工具支持该关键字，迁移工程中保留。

输入示例

```
WITH e AS (
  SELECT city, sum(population) FROM t1 group by city
  ),
  d AS (
  SELECT max(music) as max_music, min(music) as min_music from student
  ),
  s AS (
  SELECT * FROM subject
  )
SELECT * FROM e;
```

输出示例

```
WITH e AS (
  SELECT
    city, sum(population)
  FROM
    t1
  GROUP BY
    city
),
d AS (
  SELECT
    max(music) AS "max_music", min(music) AS "min_music"
  FROM
    student
),
s AS (
  SELECT
    *
  FROM
    subject
)
SELECT
  *
FROM
  e;
```

5.5.4.34 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DWS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(512),
  "datatype1" BIGINT,
  "datatype2" BIGINT,
  "datatype3" BIGINT,
  "datatype4" BIGINT
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype0");

-- A.
ALTER TABLE "public"."runoob_alter_test" CHANGE COLUMN "datatype1" "datatype1new" VARCHAR(200)
NULL DEFAULT NULL;

-- B.
ALTER TABLE "public"."runoob_alter_test" CHANGE COLUMN "datatype2" "datatype2new" VARCHAR(200)
NOT NULL;

-- C.
ALTER TABLE "public"."runoob_alter_test" CHANGE COLUMN "datatype3" "datatype3new" VARCHAR(400)
NULL DEFAULT NULL;

-- D.
ALTER TABLE "public"."runoob_alter_test" CHANGE COLUMN "datatype4" "datatype4new" VARCHAR(200)
NULL DEFAULT NULL;
```

5.5.4.35 CHECK 约束

CHECK约束声明一个布尔表达式，每次要插入的新行或者要更新的行的新值必须使表达式结果为真或未知才能成功，否则会上报一个异常并且不会修改数据库。

声明为字段约束的检查约束应该只引用该字段的数值，而在表约束里出现的表达式可以引用多个字段。

CREATE TABLE CHECK

在DWS中，创建表时定义某一列的CHECK约束可以放在列字段后面也可放在下面，语法为：CHECK (column name > 0)，如需命名 CHECK 约束，并定义多个列的 CHECK 约束则可使用下面的语法：CONSTRAINT chk_name CHECK (column_namw1 >0 AND column_name2='x>x")。

⚠ 注意

check约束也将随着表的删除而删除。

输入示例

```
DROP TABLE IF EXISTS `t1`;
CREATE TABLE IF NOT EXISTS t1  (
  id int(25) not null primary key check (id > 1 and id < 100),
  city varchar(255) not null unique check (city='城市1' or city='城市2' or city='城市3'),
  population int(25) not null ,
```

```
  constraint t1_1 check (population>0 and population<10000)
) ;
```

输出示例

```
DROP TABLE IF EXISTS "public"."t1";
CREATE TABLE IF NOT EXISTS "public"."t1" (
  "id" INTEGER NOT NULL PRIMARY KEY CHECK (
    id > 1
    and id < 100
  ),
  "city" VARCHAR(1020) NOT NULL CHECK (
    city = '城市1'
    or city = '城市2'
    or city = '城市3'
  ),
  "population" INTEGER NOT NULL,
  CONSTRAINT t1_1 CHECK (
    population > 0
    and population < 10000
  )
) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH ("id");
```

ALTER TABLE CHECK

对表字段操作时，支持无命名约束和有命名约束，在某一列创建CHECK约束则可使用语法：**ALTER TABLE 表名 ADD CHECK(<检查约束>);**

如需命名CHECK约束，并定义多个列的CHECK约束则可使用语法：**ALTER TABLE 表名 ADD CONSTRAINT <检查约束名> CHECK(<检查约束>);**

输入示例

```
ALTER TABLE t1 add check (id>1 and id > 2  and id < 100 or id =50);
ALTER TABLE student add constraint check02 check (id > 1 and class = '3');
ALTER TABLE t1 add check (class=1 or class =2  or class = 3 or class=4 or class = 5 or class = 6);
ALTER TABLE t1 add check (city='城市1' or city='城市2' or city='城市3');
```

输出示例

```
ALTER TABLE "public"."t1" ADD CHECK ( id > 1  and id > 2 and id < 100  or id = 50 );
ALTER TABLE "public"."student" ADD CONSTRAINT check02 CHECK (  id > 1  and class = '3');
ALTER TABLE "public"."t1" ADD CHECK ( class = 1 or class = 2 or class = 3  or class = 4 or class = 5 or class
= 6);
ALTER TABLE "public"."t1" ADD CHECK ( city = '城市1' or city = '城市2' or city = '城市3' );
```

5.5.4.36 DROP 删除表

DWS与MySQL都支持使用DROP语句删除表，但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 IF NOT EXISTS "public"."express_elb_server"
(
  "runoob_id" VARCHAR(40),
```

```
  "runoob_title" VARCHAR(400) NOT NULL,
  "runoob_author" VARCHAR(160) NOT NULL,
  "submission_date" VARCHAR(40)
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("runoob_id");
DROP TABLE "public"."express_elb_server";
```

5.5.4.37 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。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 IF NOT EXISTS "public"."runoob_tbl_old"
(
  "datatype_1" SMALLINT,
  "datatype_2" SMALLINT,
  "datatype_3" SMALLINT DEFAULT '2018',
  "datatype_4" TIME WITHOUT TIME ZONE DEFAULT NULL
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype_1");

CREATE TABLE "public"."runoob_tbl"( LIKE "public"."runoob_tbl_old"
   INCLUDING COMMENTS INCLUDING CONSTRAINTS INCLUDING DEFAULTS INCLUDING INDEXES
INCLUDING STORAGE);
```

5.5.4.38 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DWS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(400),
  "datatype1" BIGINT,
  "datatype2" BIGINT,
  "datatype3" BIGINT
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype0");

-- A.
ALTER TABLE "public"."runoob_alter_test" MODIFY "datatype1" SMALLINT NULL DEFAULT NULL;

-- B.
ALTER TABLE "public"."runoob_alter_test" MODIFY "datatype1" SMALLINT NOT NULL;

-- C.
ALTER TABLE "public"."runoob_alter_test" MODIFY "datatype1" SMALLINT NOT NULL;

-- D.
ALTER TABLE "public"."runoob_alter_test" MODIFY "datatype1" SMALLINT NOT NULL;
```

5.5.4.39 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，DWS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;
```

5.5.4.40 ROUNDROBIN 表

DWS支持建立roundrobin表，根据实际需要设置**表5-14**中的参数**table.type**进行配置。设置table.type=ROUND-ROBIN。

输入示例

```
CREATE TABLE charge_snapshot (
    id bigint NOT NULL,
    profit_model integer,
    ladder_rebate_rule text
);
```

输出示例

```
CREATE TABLE "public"."charge_snapshot" (
  "id" BIGINT NOT NULL,
  "profit_model" INTEGER,
  "ladder_rebate_rule" TEXT
) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY ROUNDROBIN;
```

5.5.4.41 RENAME 重命名表名

MySQL重命名表名的语句与DWS有一些差异。DSC工具迁移时会根据DWS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";
   ```

5.5.4.42 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DWS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 IF NOT EXISTS "public"."runoob_alter_test"
(
  "datatype1" SERIAL NOT NULL,
  "datatype2" REAL,
  "datatype3" DOUBLE PRECISION,
  "datatype4" TEXT NOT NULL,
  "datatype5" SMALLINT NOT NULL DEFAULT '2018',
  "datatype6" TIMESTAMP WITHOUT TIME ZONE NOT NULL DEFAULT '2018-10-12 15:27:33.999999',
  "datatype7" CHAR(4) NOT NULL DEFAULT '',
  "datatype8" VARCHAR(200),
  "datatype9" VARCHAR(200) NOT NULL DEFAULT '',
  "datatype10" TIME WITHOUT TIME ZONE NOT NULL DEFAULT '10:20:59',
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");

ALTER TABLE "public"."runoob_alter_test" ALTER COLUMN "datatype2" SET DEFAULT '1';
ALTER TABLE "public"."runoob_alter_test" ALTER COLUMN "datatype2" SET DEFAULT '3';
ALTER TABLE "public"."runoob_alter_test" ALTER COLUMN "datatype2" DROP DEFAULT;
ALTER TABLE "public"."runoob_alter_test" ALTER COLUMN "datatype2" DROP DEFAULT;
```

5.5.4.43 字段名重命名

保留关键字作字段名

在DWS中保留关键字作字段名需要加双引号，目前工具支持的保留关键字有desc、checksum、operator、size等。

输入示例

```
CREATE TABLE `desc` (
    user char,
    checksum int,
    operator smallint,
    desc char,
    size bigint
);
```

输出示例

```
CREATE TABLE "public"."desc" (
  "user" CHAR(4),
  "checksum" INTEGER,
  "operator" SMALLINT,
  "desc" CHAR(4),
  "size" BIGINT
) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH ("user");
```

系统表隐藏字段名作字段名

定义字段名和DWS中系统表隐藏字段重名，需重新命名：包括xc_node_id，tableoid, cmax, xmax, cmin, xmin, ctid, tid;重新命名方式，字段后加_new，例如xc_node_id改为xc_node_id_new。

输入示例


```
DROP TABLE IF EXISTS `renameFieldName`;
CREATE TABLE IF NOT EXISTS `renameFieldName`(
    xc_node_id int,
    tableoid char(3),
    cmax smallint,
    xmax bigint auto_increment,
    cmin varchar(10),
    xmin int,
    ctid int auto_increment,
    tid int
);
DROP TABLE IF EXISTS `renameFieldName`;
```

输出示例

```
DROP TABLE IF EXISTS "public"."renamefieldname";
CREATE TABLE IF NOT EXISTS "public"."renamefieldname" (
  "xc_node_id_new" INTEGER,
  "tableoid_new" CHAR(12),
  "cmax_new" SMALLINT,
  "xmax_new" BIGSERIAL,
  "cmin_new" VARCHAR(40),
  "xmin_new" INTEGER,
  "ctid_new" SERIAL,
  "tid_new" INTEGER
) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH
("xc_node_id_new");
DROP TABLE IF EXISTS "public"."renamefieldname";
```

5.5.4.44 行列存压缩

DWS中，只支持列存表压缩功能，暂不支持行存表压缩功能。优化行列存压缩机制，DSC工具迁移时会根据DWS的特性进行相应适配。

压缩参数：

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。

table.compress.row和**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。

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。

行存表输入示例

```
DROP TABLE IF EXISTS `public`.`runoob_tbl`;
CREATE TABLE IF NOT EXISTS `public`.`runoob_tbl`(
   `runoob_id` VARCHAR,
   `runoob_title` VARCHAR(100) NOT NULL,
   `runoob_author` VARCHAR(40) NOT NULL,
   `submission_date` VARCHAR
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

行存表输出示例

```
DROP TABLE IF EXISTS "public"."runoob_tbl";
CREATE TABLE IF NOT EXISTS "public"."runoob_tbl" (
  "runoob_id" VARCHAR,
```

```
  "runoob_title" VARCHAR(400) NOT NULL,
  "runoob_author" VARCHAR(160) NOT NULL,
  "submission_date" VARCHAR
) WITH (ORIENTATION = ROW, COMPRESSION = YES) COMPRESS DISTRIBUTE BY HASH ("runoob_id");
```

列存表输入示例

```
DROP TABLE IF EXISTS `public`.`runoob_tbl`;
CREATE TABLE IF NOT EXISTS `public`.`runoob_tbl`(
   `runoob_id` VARCHAR,
   `runoob_title` VARCHAR(100) NOT NULL,
   `runoob_author` VARCHAR(40) NOT NULL,
   `submission_date` VARCHAR
)ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

列存表输出示例

```
DROP TABLE IF EXISTS "public"."runoob_tbl";
CREATE TABLE IF NOT EXISTS "public"."runoob_tbl" (
  "runoob_id" VARCHAR,
  "runoob_title" VARCHAR(400) NOT NULL,
  "runoob_author" VARCHAR(160) NOT NULL,
  "submission_date" VARCHAR
) WITH (
  COMPRESSLEVEL = 1,
  ORIENTATION = COLUMN,
  COMPRESSION = LOW
) DISTRIBUTE BY HASH ("runoob_id");
```

5.5.4.45 添加与删除列

MySQL添加、删除列语句与DWS存在差异。DSC工具迁移时会根据DWS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FIRST，AFTER特性的完整迁移。基于当前的临时方案，工具仅移除该关键字。

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" REAL,
  "datatype3" DOUBLE PRECISION,
  "datatype4" TEXT NOT NULL,
  "datatype5" SMALLINT NOT NULL DEFAULT '2018',
  "datatype6" TIMESTAMP WITHOUT TIME ZONE NOT NULL DEFAULT '2018-10-12 15:27:33.999999',
  "datatype7" CHAR(4) NOT NULL DEFAULT '',
  "datatype8" VARCHAR(200),
  "datatype9" VARCHAR(200) NOT NULL DEFAULT '',
  "datatype10" TIME WITHOUT TIME ZONE NOT NULL DEFAULT '10:20:59',
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");

-- A.
ALTER TABLE "public"."runoob_alter_test" ADD COLUMN "datatype1_1" INTEGER NOT NULL DEFAULT 0;
ALTER TABLE "public"."runoob_alter_test" DROP COLUMN "datatype1_1" RESTRICT;

-- B.
ALTER TABLE "public"."runoob_alter_test" ADD COLUMN "datatype1_1" INTEGER NOT NULL DEFAULT 0;
ALTER TABLE "public"."runoob_alter_test" DROP COLUMN "datatype1_1" RESTRICT;

-- C.
ALTER TABLE "public"."runoob_alter_test" ADD COLUMN "datatype1_1" INTEGER NOT NULL DEFAULT 0;
ALTER TABLE "public"."runoob_alter_test" DROP COLUMN "datatype1_1" RESTRICT;

-- D.
ALTER TABLE "public"."runoob_alter_test" ADD COLUMN "datatype1_1" INTEGER NOT NULL DEFAULT 0;
ALTER TABLE "public"."runoob_alter_test" DROP COLUMN "datatype1_1" RESTRICT;

-- E.
ALTER TABLE "public"."runoob_alter_test" ADD COLUMN "datatype1_1" INTEGER NOT NULL, ADD
COLUMN "datatype1_2" VARCHAR(800) NOT NULL DEFAULT '';
ALTER TABLE "public"."runoob_alter_test" DROP COLUMN "datatype1_1" RESTRICT, DROP COLUMN
"datatype1_2" RESTRICT;
```

5.5.5 索引

5.5.5.1 唯一索引

DWS不支持唯一索引(约束)与主键约束联合使用。DSC工具迁移时会根据DWS的特性进行相应适配。

⚠ 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，如存在主键索引与唯一索引是相同列，DSC工具迁移时会将唯一索引移除。

   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 IF NOT EXISTS "public"."runoob_datatype_test"
   (
     "id" SERIAL PRIMARY KEY,
     "name" VARCHAR(128) NOT NULL
   )
     WITH ( ORIENTATION = ROW, COMPRESSION = NO )
     NOCOMPRESS
     DISTRIBUTE BY HASH ("id");
   ```

2. ALTER TABLE创建唯一索引，DSC工具迁移时会根据DWS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 IF NOT EXISTS "public"."runoob_alter_test"
   (
     "datatype1" INTEGER,
     "datatype2" REAL,
   ```

```
  "datatype3" DOUBLE PRECISION
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");

CREATE INDEX "idx_runoob_alter_test_datatype1" ON "public"."runoob_alter_test" ("datatype1");
CREATE INDEX "idx_runoob_alter_test_datatype1" ON "public"."runoob_alter_test" ("datatype2");
CREATE INDEX "idx_runoob_alter_test_datatype1" ON "public"."runoob_alter_test" ("datatype3");

CREATE TABLE IF NOT EXISTS "public"."runoob_alter_test"
(
  "datatype1" INTEGER,
  "datatype2" REAL,
  "datatype3" DOUBLE PRECISION,
  "datatype4" TEXT NOT NULL,
  "datatype5" SMALLINT NOT NULL DEFAULT '2018',
  "datatype6" TIMESTAMP WITHOUT TIME ZONE NOT NULL DEFAULT '2018-10-12 15:27:33.999999'
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");

CREATE INDEX "idx_runoob_alter_test_datatype1" ON "public"."runoob_alter_test" ("datatype1");
CREATE INDEX "idx_runoob_alter_test_datatype2" ON "public"."runoob_alter_test" ("datatype2");
CREATE INDEX "idx_runoob_alter_test_datatype3" ON "public"."runoob_alter_test" ("datatype3");
CREATE INDEX "idx_runoob_alter_test_datatype4" ON "public"."runoob_alter_test" ("datatype4");
CREATE INDEX "idx_runoob_alter_test_datatype5" ON "public"."runoob_alter_test" ("datatype5");
CREATE INDEX "idx_runoob_alter_test_datatype6" ON "public"."runoob_alter_test" ("datatype6");
```

3. CREATE INDEX创建唯一索引，DSC工具迁移时会根据DWS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 NOT NULL,
     "table01_theme" VARCHAR(400) DEFAULT NULL,
     "author_name" CHAR(40) DEFAULT NULL,
     "author_id" INTEGER DEFAULT NULL,
     "create_time" INTEGER DEFAULT NULL,
     PRIMARY KEY ("table01_id")
   )
     WITH ( ORIENTATION = ROW, COMPRESSION = NO )
     NOCOMPRESS
     DISTRIBUTE BY HASH ("table01_id");
   CREATE INDEX "author_index" ON "public"."test_index_table01" ("author_id");
   ```

4. CREATE TABLE中存在多个唯一索引，DSC工具迁移时会根据DWS的特性将所有唯一索引创建为普通索引。

   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,
   ```

```
  UNIQUE(`TABLE01_ID`),
  UNIQUE(`AUTHOR_ID`)
);
```

输出示例

```
CREATE TABLE "public"."test_index_table01" (
 "table01_id" INTEGER NOT NULL,
 "table01_theme" VARCHAR(400) DEFAULT NULL,
 "author_name" CHAR(40) DEFAULT NULL,
 "author_id" INTEGER DEFAULT NULL,
 "create_time" INTEGER DEFAULT NULL
) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH
("table01_id");
CREATE INDEX "idx_test_index_table01_table01_id" ON "public"."test_index_table01"("TABLE01_ID");
CREATE INDEX "idx_test_index_table01_author_id" ON "public"."test_index_table01"("AUTHOR_ID");
```

5. CREATE TABLE中存在一个唯一索引，并不存在主键索引时，DSC工具迁移时会根据DWS的特性保留该唯一索引。

   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,
     UNIQUE(`AUTHOR_ID`)
   );
   ```

   输出示例

   ```
   CREATE TABLE "public"."test_index_table01" (
    "table01_id" INTEGER NOT NULL,
    "table01_theme" VARCHAR(400) DEFAULT NULL,
    "author_name" CHAR(40) DEFAULT NULL,
    "author_id" INTEGER DEFAULT NULL,
    "create_time" INTEGER DEFAULT NULL,
    UNIQUE ("author_id")
   ) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH
   ("author_id");
   ```

6. CREATE TABLE中存在主键索引时，DSC工具迁移时会根据DWS的特性将所有的唯一索引创建为普通索引。

   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`),
     UNIQUE(`AUTHOR_ID`)
   );
   ```

   输出示例

   ```
   CREATE TABLE "public"."test_index_table01" (
    "table01_id" INTEGER NOT NULL,
    "table01_theme" VARCHAR(400) DEFAULT NULL,
    "author_name" CHAR(40) DEFAULT NULL,
    "author_id" INTEGER DEFAULT NULL,
    "create_time" INTEGER DEFAULT NULL,
    PRIMARY KEY ("table01_id")
   ) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH
   ("table01_id");
   CREATE INDEX "idx_test_index_table01_author_id" ON "public"."test_index_table01"("AUTHOR_ID");
   ```

5.5.5.2 普通索引和前缀索引

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DWS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 IF NOT EXISTS "public"."runoob_datatype_test"
   (
     "id" SERIAL PRIMARY KEY,
     "name" VARCHAR(512) NOT NULL
   )
     WITH ( ORIENTATION = ROW, COMPRESSION = NO )
     NOCOMPRESS
     DISTRIBUTE BY HASH ("id");
   CREATE INDEX "index_single" ON "public"."runoob_datatype_test" USING BTREE ("name");
   ```

2. 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 NOT NULL,
     "user_name" CHAR(80) DEFAULT NULL,
     "detail" VARCHAR(400) DEFAULT NULL,
     PRIMARY KEY ("id")
   )
     WITH ( ORIENTATION = ROW, COMPRESSION = NO )
     NOCOMPRESS
     DISTRIBUTE BY HASH ("id");

   CREATE INDEX "user_name_index_02" ON "public"."test_create_table05" ("user_name");
   ```

3. 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 IF NOT EXISTS "public"."customer"
   (
   ```

```
  "name" VARCHAR(256) PRIMARY KEY,
  "id" INTEGER,
  "id2" INTEGER
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("name");

CREATE INDEX "part_of_name" ON "public"."customer" USING BTREE ("name");
```

5.5.5.3 HASH 索引

DWS不支持HASH索引。DSC工具迁移时会根据DWS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(400) NOT NULL,
    "theme" VARCHAR(800) DEFAULT NULL,
    "send_id" INTEGER DEFAULT NULL,
    "send_name" CHAR(80) DEFAULT NULL,
    "send_time" TIMESTAMP WITHOUT TIME ZONE DEFAULT NULL,
    "demand_content" TEXT NOT NULL,
    PRIMARY KEY ("demand_id")
   )
    WITH ( ORIENTATION = ROW, COMPRESSION = NO )
    NOCOMPRESS
    DISTRIBUTE BY HASH ("demand_id");
   CREATE INDEX "con_index" ON "public"."test_create_table03" ("demand_content");
   CREATE INDEX "send_info_index" ON "public"."test_create_table03"
   ("send_id","send_name","send_time");
   ```

2. 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 IF NOT EXISTS "public"."runoob_alter_test"
   (
    "datatype1" SERIAL NOT NULL,
    "datatype2" REAL,
    PRIMARY KEY ("datatype1")
   )
    WITH ( ORIENTATION = ROW, COMPRESSION = NO )
   ```

```
 NOCOMPRESS
 DISTRIBUTE BY HASH ("datatype1");

CREATE INDEX "altertable_addkey_indextype" ON "public"."runoob_alter_test" ("datatype1");
```

3. 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(120) NOT NULL,
 "iname" VARCHAR(120) NOT NULL,
 PRIMARY KEY ("id")
)
 WITH ( ORIENTATION = ROW, COMPRESSION = NO )
 NOCOMPRESS
 DISTRIBUTE BY HASH ("id");
CREATE INDEX "fname_index" ON "public"."test_index_table06" ("fname");
CREATE INDEX "name_01" ON "public"."test_index_table06" ("fname","iname");
```

5.5.5.4 BTREE 索引

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DWS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(400) NOT NULL,
 "theme" VARCHAR(800) DEFAULT NULL,
 "send_id" INTEGER DEFAULT NULL,
 "send_name" CHAR(80) DEFAULT NULL,
 "send_time" TIMESTAMP WITHOUT TIME ZONE DEFAULT NULL,
 "demand_content" TEXT NOT NULL,
 PRIMARY KEY ("demand_id")
)
 WITH ( ORIENTATION = ROW, COMPRESSION = NO )
 NOCOMPRESS
 DISTRIBUTE BY HASH ("demand_id");
```

```
CREATE INDEX "theme_index" ON "public"."test_create_table03" USING BTREE ("theme");
CREATE INDEX "name_index" ON "public"."test_create_table03" USING BTREE ("send_name");
```

2. 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 IF NOT EXISTS "public"."runoob_alter_test"
(
  "datatype1" SERIAL NOT NULL,
  "datatype2" REAL,
  PRIMARY KEY ("datatype1")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("datatype1");

CREATE INDEX "altertable_addkey_indextype" ON "public"."runoob_alter_test" ("datatype1");
```

3. CREATE INDEX创建BTREE索引。

输入示例

```
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 NOT NULL,
  "user_name" CHAR(80) DEFAULT NULL,
  "detail" VARCHAR(400) DEFAULT NULL,
  PRIMARY KEY ("id")
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("id");
CREATE INDEX "user_id_index" ON "public"."test_index_table05" ("user_id");
CREATE INDEX "user_name_index" ON "public"."test_index_table05" USING BTREE ("user_name");
CREATE INDEX "detail_index" ON "public"."test_index_table05" USING BTREE ("detail");
CREATE INDEX "user_info_index" ON "public"."test_index_table05" USING BTREE
("user_id","user_name");
```

5.5.5.5 SPATIAL 空间索引

DWS不支持SPATIAL空间索引。DSC工具迁移时会根据DWS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" GEOMETRYCOLLECTION NOT NULL,
  "d" POLYGON NOT NULL,
  "e" BOX NOT NULL,
  "f" BOX NOT NULL,
  "g" POLYGON NOT NULL
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("id");
CREATE INDEX "a_index" ON "public"."test_create_table04" USING GIST ("a");
CREATE INDEX "b_index" ON "public"."test_create_table04" USING GIST ("b");
CREATE INDEX "c_index" ON "public"."test_create_table04" USING GIST ("c");
CREATE INDEX "d_index" ON "public"."test_create_table04" USING GIST ("d");
CREATE INDEX "e_index" ON "public"."test_create_table04" USING GIST ("e");
CREATE INDEX "f_index" ON "public"."test_create_table04" USING GIST ("f");
CREATE INDEX "g_index" ON "public"."test_create_table04" USING GIST ("g");
```

2. 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" GEOMETRYCOLLECTION NOT NULL,
  "d" POLYGON NOT NULL,
  "e" BOX NOT NULL,
  "f" BOX NOT NULL,
  "g" POLYGON NOT NULL
)
```

```
 WITH ( ORIENTATION = ROW, COMPRESSION = NO )
 NOCOMPRESS
 DISTRIBUTE BY HASH ("id");

CREATE INDEX "a_index" ON "public"."test_create_table04" USING GIST ("a");
CREATE INDEX "e_index" ON "public"."test_create_table04" USING GIST ("e");
```

3. 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" GEOMETRYCOLLECTION NOT NULL,
 "d" POLYGON NOT NULL,
 "e" BOX NOT NULL,
 "f" BOX NOT NULL,
 "g" POLYGON NOT NULL
)
 WITH ( ORIENTATION = ROW, COMPRESSION = NO )
 NOCOMPRESS
 DISTRIBUTE BY HASH ("id");

CREATE INDEX "a_index" ON "public"."test_create_table04" USING GIST ("a");
```

5.5.5.6 FULLTEXT 全文索引

DWS不支持FULLTEXT全文索引。DSC工具迁移时会根据DWS的特性进行相应适配。

1. 内联FULLTEXT全文索引。

   输入示例

```
## A.
CREATE TABLE `public`.`test_create_table02` (
    `ID` INT(11) NOT NULL PRIMARY KEY,
    `TITLE` CHAR(255) NOT NULL,
    `CONTENT` TEXT NULL,
    `CREATE_TIME` DATETIME NULL DEFAULT NULL,
     FULLTEXT (`CONTENT`)
);

## B.
CREATE TABLE IF NOT EXISTS `public`.`runoob_dataType_test`
(
  `id` INT PRIMARY KEY AUTO_INCREMENT,
  `name` VARCHAR(128) NOT NULL,
  FULLTEXT INDEX (name)
);

## C.
CREATE TABLE IF NOT EXISTS `public`.`runoob_dataType_test`
(
```

```
 `id` INT PRIMARY KEY AUTO_INCREMENT,
 `name` VARCHAR(128) NOT NULL,
 FULLTEXT INDEX (name ASC)
);
```

输出示例

```
-- A.
CREATE TABLE "public"."test_create_table02"
(
 "id" INTEGER NOT NULL PRIMARY KEY,
 "title" CHAR(1020) NOT NULL,
 "content" TEXT,
 "create_time" TIMESTAMP WITHOUT TIME ZONE DEFAULT NULL
)
 WITH ( ORIENTATION = ROW, COMPRESSION = NO )
 NOCOMPRESS
 DISTRIBUTE BY HASH ("id");
CREATE INDEX "idx_test_create_table02_content" ON "public"."test_create_table02" USING
GIN(to_tsvector(coalesce("content","")));

-- B.
CREATE TABLE IF NOT EXISTS "public"."runoob_datatype_test"
(
 "id" SERIAL PRIMARY KEY,
 "name" VARCHAR(512) NOT NULL
)
 WITH ( ORIENTATION = ROW, COMPRESSION = NO )
 NOCOMPRESS
 DISTRIBUTE BY HASH ("id");
CREATE INDEX "idx_runoob_datatype_test_name" ON "public"."runoob_datatype_test" USING
GIN(to_tsvector(coalesce("name","")));

-- C.
CREATE TABLE IF NOT EXISTS "public"."runoob_datatype_test"
(
 "id" SERIAL PRIMARY KEY,
 "name" VARCHAR(512) NOT NULL
)
 WITH ( ORIENTATION = ROW, COMPRESSION = NO )
 NOCOMPRESS
 DISTRIBUTE BY HASH ("id");
CREATE INDEX "idx_runoob_datatype_test_name" ON "public"."runoob_datatype_test" USING
GIN(to_tsvector(coalesce("name","")));
```

2. ALTER TABLE创建全文索引。

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 FULLTEXT INDEX USER_ID_INDEX_02(USER_ID);
ALTER TABLE TEST_CREATE_TABLE05 ADD FULLTEXT USER_NAME_INDEX_02(USER_NAME);
```

输出示例

```
CREATE TABLE "public"."test_create_table05"
(
 "id" SERIAL NOT NULL,
 "user_id" INTEGER NOT NULL,
 "user_name" CHAR(80) DEFAULT NULL,
 "detail" VARCHAR(400) DEFAULT NULL,
 PRIMARY KEY ("id")
)
 WITH ( ORIENTATION = ROW, COMPRESSION = NO )
 NOCOMPRESS
 DISTRIBUTE BY HASH ("id");
CREATE INDEX "user_id_index_02" ON "public"."test_create_table05" USING
```

```
GIN(to_tsvector(coalesce("user_id",'')));
CREATE INDEX "user_name_index_02" ON "public"."test_create_table05" USING
GIN(to_tsvector(coalesce("user_name",'')));
```

3. CREATE INDEX创建全文索引。

输入示例

```
CREATE TABLE `public`.`test_index_table02` (
    `ID` INT(11) NOT NULL PRIMARY KEY,
    `TITLE` CHAR(255) NOT NULL,
    `CONTENT` TEXT NULL,
    `CREATE_TIME` INT(10) NULL DEFAULT NULL
);
CREATE FULLTEXT INDEX CON_INDEX ON TEST_INDEX_TABLE02(CONTENT);
```

输出示例

```
CREATE TABLE "public"."test_index_table02"
(
  "id" INTEGER NOT NULL PRIMARY KEY,
  "title" CHAR(1020) NOT NULL,
  "content" TEXT,
  "create_time" INTEGER DEFAULT NULL
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("id");
CREATE INDEX "con_index" ON "public"."test_index_table02" USING
GIN(to_tsvector(coalesce("content",'')));
```

5.5.5.7 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DWS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 NOT NULL,
  "demand_name" CHAR(400) NOT NULL,
  "theme" VARCHAR(800) DEFAULT NULL,
  "send_id" INTEGER DEFAULT NULL,
  "send_name" CHAR(80) DEFAULT NULL,
  "send_time" TIMESTAMP WITHOUT TIME ZONE DEFAULT NULL,
  "demand_content" TEXT NOT NULL
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
```

```
 DISTRIBUTE BY HASH ("demand_id");

CREATE INDEX "demand_name_index" ON "public"."test_create_table03" ("demand_name");
DROP INDEX "public"."demand_name_index" RESTRICT;

CREATE INDEX "send_id_index" ON "public"."test_create_table03" USING BTREE ("send_id");
DROP INDEX "public"."send_id_index" RESTRICT;
```

2. ALTER TABLE DROP INDEX

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 NOT NULL,
  "demand_name" CHAR(400) NOT NULL,
  "theme" VARCHAR(800) DEFAULT NULL,
  "send_id" INTEGER DEFAULT NULL,
  "send_name" CHAR(80) DEFAULT NULL,
  "send_time" TIMESTAMP WITHOUT TIME ZONE DEFAULT NULL,
  "demand_content" TEXT NOT NULL
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("demand_id");

CREATE INDEX "test_create_table03_name_index" ON "public"."test_create_table03"
("demand_name");
DROP INDEX "public"."test_create_table03_name_index" RESTRICT;
```

5.5.5.8 索引重命名

DSC工具支持索引重命名，对索引名前加表名前缀防止索引名冲突（只支持创建有具体索引名的DDL语句，目前不支持删除索引的重命名，修改该参数需慎重）。

修改配置

打开**表1 features-mysql.properties文件中的配置参数**配置文件，修改如下参数为true。(默认false：不进行重命名)

```
# 创建索引时，是否重新命名索引名
table.index.rename=true
```

输入示例

```
CREATE TABLE IF NOT EXISTS `CUSTOMER`(
    `NAME` VARCHAR(64) PRIMARY KEY,
    ID INTEGER,
    ID2 INTEGER);
CREATE INDEX ID_INDEX USING BTREE ON CUSTOMER (ID);
ALTER TABLE CUSTOMER ADD INDEX ID3_INDEX(ID2);
```

输出示例

```
CREATE TABLE IF NOT EXISTS "public"."customer" (
    "name" VARCHAR(256) PRIMARY KEY,
    "id" INTEGER,
    "id2" INTEGER) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY
HASH ("name");
CREATE INDEX customer_id_index ON "public"."customer" USING BTREE ("id");
CREATE INDEX customer_id3_index ON "public"."customer" ("id2");
```

5.5.6 注释

MySQL支持由 '#' 或 '--' 字符引起的单行注释，而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 IF NOT EXISTS "public"."runoob_tbl"
(
  "runoob_id" VARCHAR,
  "runoob_title" VARCHAR(400) NOT NULL,
  "runoob_author" VARCHAR(160) NOT NULL,
  "submission_date" VARCHAR
)
  WITH ( ORIENTATION = ROW, COMPRESSION = NO )
  NOCOMPRESS
  DISTRIBUTE BY HASH ("runoob_id");
```

5.5.7 数据库

在MySQL中，DATABASE 是一种模式对象，等同于Oracle、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";
```

5.5.8 数据操作语句（DML）

5.5.8.1 INSERT

INSERT插入形式包括：HIGH_PRIORITY、LOW_PRIORITY、PARTITION、DELAYED、IGNORE、VALUES以及ON DUPLICATE KEY UPDATE。

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);
```

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);
```

PARTITION

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);
```

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);
```

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);
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);
```

VALUES

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_gender,tb1_address,tb1_number)
VALUES('David','male','NewYork','01015827875'),('Rachel','female','NewYork','01015827749'),
('Monica','female','NewYork','010158996743');
```

输出示例

```
INSERT INTO "public"."exmp_tb1" ("tb1_name","tb1_gender","tb1_address","tb1_number") VALUES
('David','male','NewYork','01015827875');
INSERT INTO "public"."exmp_tb1" ("tb1_name","tb1_gender","tb1_address","tb1_number") VALUES
('Rachel','female','NewYork','01015827749');
INSERT INTO "public"."exmp_tb1" ("tb1_name","tb1_gender","tb1_address","tb1_number") VALUES
('Monica','female','NewYork','010158996743');
```

ON DUPLICATE KEY UPDATE

INSERT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);
```

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');
```

5.5.8.2 UPDATE

MySQL的UPDATE操作形式包括：LOW_PRIORITY、ORDER BY、LIMIT 、IGNORE。

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;
```

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;
```

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;
```

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;
```

5.5.8.3 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。

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);
```

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);
```

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);
```

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_gender,tb1_address,tb1_number)
VALUES(17,'David','male','NewYork11','01015827875'),(18,'Rachel','female','NewYork22','01015827749'),
(20,'Monica','female','NewYork','010158996743');
Replace INTO  exmp_tb1 (tb1_id,tb1_name,tb1_gender,tb1_address,tb1_number)
VALUES(17,'David1','male','NewYork11','01015827875'),(21,'Rachel','female','NewYork22','01015827749'),
(22,'Monica','female','NewYork','010158996743');
Replace INTO  exmp_tb1 (tb1_id,tb1_name,tb1_gender,tb1_address,tb1_number,tb1_date)
VALUES(17,'David2',DEFAULT,'NewYork11','01015827875',DEFAULT),
(18,'Rachel','female',DEFAULT,'01015827749','2018-12-14 10:44:20'),
(DEFAULT,'Monica','female',DEFAULT,DEFAULT,'2018-12-14 10:44:20');
Replace INTO  exmp_tb1 VALUES(DEFAULT,'David',DEFAULT,'NewYork11','01015827875',DEFAULT),
(18,'Rachel','female',DEFAULT,'01015827749','2018-12-14 10:44:20'),
(DEFAULT,'Monica','female',DEFAULT,DEFAULT,'2018-12-14 10:44:20');
```

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_gender","tb1_address","tb1_number") VALUES
(17,'David','male','NewYork11','01015827875');
INSERT INTO "public"."exmp_tb1" ("tb1_id","tb1_name","tb1_gender","tb1_address","tb1_number") VALUES
(18,'Rachel','female','NewYork22','01015827749');
INSERT INTO "public"."exmp_tb1" ("tb1_id","tb1_name","tb1_gender","tb1_address","tb1_number") VALUES
(20,'Monica','female','NewYork','010158996743');
INSERT INTO "public"."exmp_tb1" ("tb1_id","tb1_name","tb1_gender","tb1_address","tb1_number") VALUES
(17,'David1','male','NewYork11','01015827875');
INSERT INTO "public"."exmp_tb1" ("tb1_id","tb1_name","tb1_gender","tb1_address","tb1_number") VALUES
(21,'Rachel','female','NewYork22','01015827749');
INSERT INTO "public"."exmp_tb1" ("tb1_id","tb1_name","tb1_gender","tb1_address","tb1_number") VALUES
(22,'Monica','female','NewYork','010158996743');
INSERT INTO "public"."exmp_tb1"
("tb1_id","tb1_name","tb1_gender","tb1_address","tb1_number","tb1_date") VALUES
(17,'David2',DEFAULT,'NewYork11','01015827875',DEFAULT);
INSERT INTO "public"."exmp_tb1"
("tb1_id","tb1_name","tb1_gender","tb1_address","tb1_number","tb1_date") VALUES
(18,'Rachel','female',DEFAULT,'01015827749','2018-12-14 10:44:20');
INSERT INTO "public"."exmp_tb1"
("tb1_id","tb1_name","tb1_gender","tb1_address","tb1_number","tb1_date") VALUES
(DEFAULT,'Monica','female',DEFAULT,DEFAULT,'2018-12-14 10:44:20');
INSERT INTO "public"."exmp_tb1"  VALUES (DEFAULT,'David',DEFAULT,'NewYork11','01015827875',DEFAULT);
INSERT INTO "public"."exmp_tb1"  VALUES (18,'Rachel','female',DEFAULT,'01015827749','2018-12-14
10:44:20');
INSERT INTO "public"."exmp_tb1"  VALUES (DEFAULT,'Monica','female',DEFAULT,DEFAULT,'2018-12-14
10:44:20');
```

SET

MySQL REPLACE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);
```

5.5.8.4 引号

单引号

MySQL中别名带单引号，DWS不支持，DSC迁移改为双引号。

输入示例

```
select name as 'mingzi' from t1;
```

输出示例

```
SELECT
  name AS "mingzi"
FROM
  t1;
```

反引号

MySQL中别名、列名带反引号，DWS不支持，DSC迁移改为双引号。

输入示例

```
select `name` as `mingzi` from t1;
```

输出示例

```
SELECT
  "name" AS "mingzi"
FROM
  t1;
```

双引号

MySQL中常量字符串用双引号扩住，DWS不支持，DSC迁移改为单引号。

输入示例

```
select name from t1 where name = "test2";
```

输出示例

```
SELECT
  name
FROM
  t1
```

```
WHERE
  name = 'test2';
```

5.5.8.5 INTERVAL

MySQL中使用interval表达式格式为**INTERVAL N**时间单位，DWS不支持，需要转换为**INTERVAL 'N'**时间单位。

输入示例

```
SELECT CURRENT_TIME() - INTERVAL 4 DAY;
SELECT NOW() - INTERVAL 5 HOUR;
SELECT CURRENT_TIME() - INTERVAL '4' DAY;
SELECT NOW() - INTERVAL '5' HOUR;
SELECT CURRENT_TIME() - INTERVAL "4" DAY;
SELECT NOW() - INTERVAL "5" HOUR;
```

输出示例

```
SELECT  (CURRENT_TIME () - INTERVAL '4' DAY);
SELECT  (NOW () - INTERVAL '5' HOUR);
SELECT  (CURRENT_TIME () - INTERVAL '4' DAY);
SELECT  (NOW () - INTERVAL '5' HOUR);
SELECT  (CURRENT_TIME () - INTERVAL '4' DAY);
SELECT  (NOW () - INTERVAL '5' HOUR);
```

5.5.8.6 除法表达式

MySQL中，除法表达式中，当除数为0时，会返回null值。DWS会报错，因此对除法表达式进行转换，增加一个if条件表达式。

输入示例

```
select sum(c1) / c2 as result from table_t1;
select sum(c1) / count (c3/c4) as result from table_t1;
```

输出示例

```
SELECT  (if (c2 = 0, null, sum(c1) / c2)) AS "result" FROM  table_t1;
SELECT  (if (count(if (c4 = 0, null, c3 / c4)) = 0, null, sum(c1) / count(if (c4 = 0, null, c3 / c4)))) AS "result"
FROM  table_t1;
```

5.5.8.7 GROUP BY 转换

MySQL/ADB分组查询的时候允许查询非分组字段，不报错；DWS分组查询时只能查询分组字段和聚集函数，报错。需要在MYSQL兼容模式下，并开启disable_full_group_by_mysql配置项兼容此语法。

输入示例

```
SELECT e.department_id, department_name, ROUND(AVG(salary), 0) avg_salary FROM employees e JOIN
departments d on e.department_id = d.department_id GROUP BY department_name ORDER BY
department_name;
```

输出示例

```
SELECT
  e.department_id,
  department_name,
  ROUND (AVG(salary), 0) AS "avg_salary"
FROM
  employees "e"
  JOIN departments "d" ON e.department_id = d.department_id
GROUP BY
  department_name
```

```
ORDER BY
  department_name;
```

5.5.8.8 ROLLUP

MySQL中的group by column with rollup需要转换为DWS中的group by rollup (column);

输入示例

```
select id,product_id,count(1) from czb_account.equity_account_log
where id in (6957343,6957397,6957519,6957541,6957719)
group by 1, 2 with rollup;
```

输出示例

```
SELECT
  id,
  product_id,
  count(1)
FROM
  czb_account.equity_account_log
WHERE
  id IN (6957343, 6957397, 6957519, 6957541, 6957719)
GROUP BY
  ROLLUP(1, 2);
```

5.5.9 事务管理和数据库管理

5.5.9.1 事务管理

TRANSACTION

DSC工具在迁移MySQL事务处理语句时会根据DWS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;
```

LOCK

DSC工具在迁移MySQL事务处理锁表语句时会根据DWS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;
```

5.5.9.2 数据库管理

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 5-38 字符集对照表

MySQL CHARACTER SET	DWS SESSION NAMES
ASCII	SQL_ASCII
BIG5	BIG5
CP1250	WIN1250
CP1251	WIN1251

MySQL CHARACTER SET	DWS SESSION NAMES
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';
```

5.6 SQL-Server 语法迁移

5.6.1 表迁移

表名

DWS不支持“数据库名.模式名.表名”的形式，需要对应转换为“模式名.表名”的形式。

SQL-Server语法	迁移后语法
CREATE TABLE `analytics-di-dev.abase.buyer_location` (id_buyer INT, id_location INT);	CREATE TABLE "abase"."buyer_location" ("id_buyer" INT, "id_location" INT) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH ("id_buyer");

表级参数相关迁移

SQL-Server支持创建行压缩表，DWS不支持，迁移过程中删除。

SQL-Server语法	迁移后语法
CREATE TABLE dbo.T1 (c1 INT, c2 NVARCHAR(200)) WITH (DATA_COMPRESSION = ROW);	CREATE TABLE "dbo"."t1" ("c1" INT, "c2" VARCHAR(200)) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH ("c1");

SQL-Server支持创建XML压缩表，DWS不支持，迁移过程中删除。

SQL-Server语法	迁移后语法
CREATE TABLE dbo.T1 (c1 INT, c2 XML) WITH (XML_COMPRESSION = ON);	CREATE TABLE "dbo"."t1" ("c1" INT, "c2" TEXT) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH ("c1");

SQL-Server 支持TEXTIMAGE_ON参数，指示某些类型存储在指定文件组，DWS不支持，迁移过程中删除。

SQL-Server语法	迁移后语法
CREATE TABLE dbo.T1 (c1 INT, c2 text) TEXTIMAGE_ON "default";	CREATE TABLE "dbo"."t1" ("c1" INT, "c2" TEXT) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH ("c1");

SQL-Server 支持SYSTEM_VERSIONING参数，创建系统版本控制表，DWS不支持，迁移过程中删除。

SQL-Server语法	迁移后语法
CREATE TABLE Department (DepartmentNumber CHAR(10) NOT NULL PRIMARY KEY, DepartmentName VARCHAR(50) NOT NULL, ManagerID INT NULL) WITH (SYSTEM_VERSIONING = ON);	CREATE TABLE "department" ("departmentnumber" CHAR(10) NOT NULL PRIMARY KEY, "departmentname" VARCHAR(50) NOT NULL, "managerid" INT) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH ("departmentnumber");

列级参数相关迁移

SQL-Server支持创建具有稀疏列的表，DWS不支持，迁移过程中删除。

SQL-Server语法	迁移后语法
CREATE TABLE dbo.T1 (c1 INT PRIMARY KEY, c2 VARCHAR(50) SPARSE NULL);	CREATE TABLE "dbo"."t1" ("c1" INT PRIMARY KEY, "c2" VARCHAR(50)) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH ("c1");

SQL-Server支持FILESTREAM关键词，指定表的FILESTREAM数据位置，DWS不支持，迁移过程中删除。

SQL-Server语法	迁移后语法
CREATE TABLE dbo.EmployeePhoto (EmployeeId INT NOT NULL PRIMARY KEY, Photo VARBINARY(MAX) FILESTREAM NULL, MyRowGuidColumn UNIQUEIDENTIFIER NOT NULL ROWGUIDCOL UNIQUE DEFAULT NEWID());	CREATE TABLE "dbo"."employeephoto" ("employeeid" INT NOT NULL PRIMARY KEY, "photo" BYTEA, "myrowguidcolumn" TEXT NOT NULL) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH ("employeeid"); CREATE INDEX "idx_employeephoto_myrowguidcolumn" ON "dbo"."employeephoto"("myrowguidcolumn");

SQL-Server 支持创建聚集索引或非聚集索引，DWS不支持，迁移过程中删除。

主键聚集索引

SQL-Server语法	迁移后语法
CREATE TABLE Department (DepartmentNumber CHAR(10) NOT NULL PRIMARY KEY CLUSTERED, DepartmentName VARCHAR(50) NOT NULL, ManagerID INT NULL);	CREATE TABLE "department" ("departmentnumber" CHAR(10) NOT NULL PRIMARY KEY, "departmentname" VARCHAR(50) NOT NULL, "managerid" INT) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH ("departmentnumber");

唯一索引非聚集索引。

SQL-Server语法	迁移后语法
CREATE TABLE Department (DepartmentNumber CHAR(10) NOT NULL UNIQUE NONCLUSTERED, DepartmentName VARCHAR(50) NOT NULL, ManagerID INT NULL);	CREATE TABLE "department" ("departmentnumber" CHAR(10) NOT NULL UNIQUE, "departmentname" VARCHAR(50) NOT NULL, "managerid" INT) WITH (ORIENTATION = ROW, COMPRESSION = NO) NOCOMPRESS DISTRIBUTE BY HASH ("departmentnumber");

5.6.2 数据类型迁移

概述

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。下表列出了SQL-Server类型到DWS的转换示例。

类型对照

数据类型	SQL-Server类型	DWS OUTPUT
时间类型	datetimeoffset [(n)]	timestamptz(n)
	datetime2 [(n)]	timestamp(n)
	datetime	timestamp
	smalldatetime	timestamp
	date	date
	time [(n)]	time(n)
数字类型	float [(n)]	float(n)
	real [(n)]	float(n)
	decimal [(precision [, scale])]	decimal
	numeric [(precision [, scale])]	numeric
	money	money
	smallmoney	money
	bigint	bigint
	int	int
	smallint	smallint
	tinyint	tinyint
	bit	bit
字符类型	nvarchar [(n | max)]	varchar
	nchar [(n)]	nchar(n)
	varchar [(n | max)]	varchar(n)
	char [(n)]	char(n)
二进制类型	varbinary [(n | max)]	BYTEA
	binary [(n)]	BYTEA
其它类型	uniqueidentifier	text

5.7 Oracle 语法迁移

5.7.1 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**表（Oracle）**~**数据库关键字**章节。

5.7.1.1 表（Oracle）

CREATE TABLE

Oracle的CREATE TABLE语句用于创建表。DWS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。DWS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DWS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 (
```

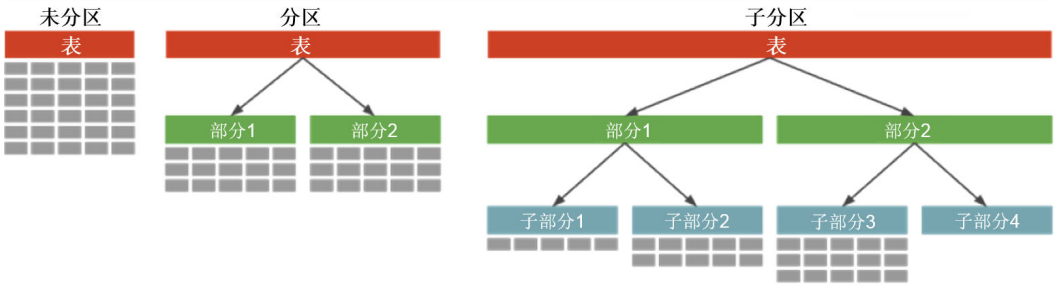


```
38
,0
)
,MODIFYDATE DATE
,CONSTRAINT PKCTP_ARM_CONFIG PRIMARY KEY (
HOSTNAME
,OPNAME
)
) /*SEGMENT CREATION DEFERRED*/
/*PCTFREE 10*/
/*PCTUSED 0*/
/*INITRANS 1*/
/*MAXTRANS 255*/
/*NOCOMPRESS*/
/*LOGGING*/
/*STORAGE( BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
DEFAULT)*/
/*TABLESPACE 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 5-5 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**Oracle配置参数**。

- **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 TBL_ORACLE (ID Number, Name VARCHAR2(100 BYTE), ADDRESS VARCHAR2(200 BYTE));	CREATE TABLE TBL_ORACLE (ID NUMBER ,Name VARCHAR2 (100) ,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), 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（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),
  Product_Name VARCHAR2(50),
  Year_Manufacture vARCHAR2(10)
)
partition by list (Year_Manufacture)
(
  partition P_2020 VALUES (2020)
  pctfree 10
  initrans 1
);
 	
CREATE OR REPLACE PROCEDURE Range_test
IS
V_ID VARCHAR2(10);
BEGIN
EXECUTE IMMEDIATE 'ALTER TABLE product
TRUNCATE PARTITION PART'||V_ID;
NULL;
END;
/
CREATE OR REPLACE PROCEDURE List_test
IS
V_ID VARCHAR2(10);
BEGIN
EXECUTE IMMEDIATE 'ALTER TABLE product
TRUNCATE PARTITION PART'||V_ID;
NULL;
END;
/
	CREATE TABLE product_range
	(
	  product_id VARCHAR2(20),
	  Product_Name VARCHAR2(50),
	  Year_Manufacture DATE
	)
	partition by range (Year_Manufacture)
	(
	  partition Year_Manufacture values less than
	(TO_DATE(' 2007-01-01 00:00:00', 'SYYYY-MM-DD
	HH24:MI:SS'))
	  pctfree 10
	  initrans 1
	);
	
	CREATE TABLE product_list
	(
	  product_id VARCHAR2(20),
	  Product_Name VARCHAR2(50),
	  Year_Manufacture vARCHAR2(10)
	)
	/*partition by list (Year_Manufacture)
	(
	  partition P_2020 VALUES (2020)
	  pctfree 10
	  initrans 1
	)*/;
	
	CREATE OR REPLACE PROCEDURE Range_test
	IS
	V_ID VARCHAR2(10);
	BEGIN
	EXECUTE IMMEDIATE 'ALTER TABLE product
	TRUNCATE PARTITION PART'||V_ID;
	NULL;
	END;
	/
	CREATE OR REPLACE PROCEDURE List_test
	IS
	V_ID VARCHAR2(10);
	BEGIN
	/*EXECUTE IMMEDIATE 'ALTER TABLE product
	TRUNCATE PARTITION PART'||V_ID;*/
	NULL;
	END;
	/

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 ,
 partition PART_2018 VALUES (2018) pctfree 10
initrans 1 ,
 partition PART_2019 VALUES (2019) pctfree 10
initrans 1 ,
 partition PART_2020 VALUES (2020) pctfree 10
initrans 1 ,
 PARTITION PART_unknown VALUES (DEFAULT));
 	
CREATE OR REPLACE PROCEDURE List_test
IS
V_ID VARCHAR2(10);
BEGIN
 EXECUTE IMMEDIATE 'ALTER TABLE product_list
TRUNCATE PARTITION PART_2020;
 NULL;
END;
/
 	
CREATE OR REPLACE PROCEDURE List_test
IS
V_ID VARCHAR2(10);
BEGIN
 EXECUTE IMMEDIATE 'ALTER TABLE product_list
TRUNCATE PARTITION PART_' || V_ID;
 NULL;
END;
/
	CREATE TABLE product_list
	(
	  product_id VARCHAR2(20),
	  Product_Name VARCHAR2(50),
	  Year_Manufacture vARCHAR2(10)
	)
	/*partition by list (Year_Manufacture)
	(partition PART_2015 VALUES
	(2011,2012,2013,2014,2015) pctfree 10 initrans 1 ,
	 partition PART_2016 VALUES (2016) pctfree 10
	initrans 1 ,
	 partition PART_2017 VALUES (2017) pctfree 10
	initrans 1 ,
	 partition PART_2018 VALUES (2018) pctfree 10
	initrans 1 ,
	 partition PART_2019 VALUES (2019) pctfree 10
	initrans 1 ,
	 partition PART_2020 VALUES (2020) pctfree 10
	initrans 1 ,
	 PARTITION PART_unknown VALUES
	(DEFAULT))*/;
	
	CREATE OR REPLACE PROCEDURE List_test
	IS
	V_ID VARCHAR2(10);
	BEGIN
	 EXECUTE IMMEDIATE 'ALTER TABLE product_list
	TRUNCATE PARTITION PART_' || V_ID;
	 NULL;
	END;
	/
	
	CREATE OR REPLACE PROCEDURE List_test
	IS
	V_ID VARCHAR2(10);
	BEGIN
	 /* EXECUTE IMMEDIATE 'ALTER TABLE
	product_list TRUNCATE PARTITION PART_' || V_ID;
	*/
	 IF 'PART_' || V_ID = 'PART_2015' THEN
	  DELETE FROM product_list WHERE
	Year_Manufacture IN (2011,2012,2013,2014,2015);
	 ELSIF 'PART_' || V_ID = 'PART_2016' THEN
	  DELETE FROM product_list WHERE
	Year_Manufacture IN (2016);
	 ELSIF 'PART_' || V_ID = 'PART_2017' THEN
	  DELETE FROM product_list WHERE
	Year_Manufacture IN (2017);
	 ELSIF 'PART_' || V_ID = 'PART_2018' THEN
	  DELETE FROM product_list WHERE
	Year_Manufacture IN (2018);
	 ELSIF 'PART_' || V_ID = 'PART_2019' THEN
	  DELETE FROM product_list WHERE
	Year_Manufacture IN (2019);
	 ELSIF 'PART_' || V_ID = 'PART_2020' THEN
	  DELETE FROM product_list WHERE
	Year_Manufacture IN (2020);
	 ELSE
	  DELETE FROM product_list WHERE
	Year_Manufacture NOT IN
	(2011,2012,2013,2014,2015,2016,2017,2018,2019,2
	020);
	 END IF;
	 NULL;

Oracle语法	迁移后语法
	END; /

SEGMENT CREATION

DWS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 */;
```

STORAGE

DWS不支持BUFFER_POOL、MAXEXTENTS等存储参数。如果comment_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comment_storage_parameter设为true，存储参数会被注释掉。

STORE

DWS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DWS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')))*/;

5.7.1.2 临时表

DWS不支持GLOBAL TEMPORARY TABLE。它会将GLOBAL TEMPORARY TABLE迁移为LOCAL TEMPORARY TABLE。

同样，DWS也不支持ON COMMIT DELETE ROWS。它会将其迁移为ON COMMIT PRESERVE ROWS。

以下示例为临时表迁移前后的语法：

临时表迁移前

图 5-6 表中包含 GLOBAL TEMPORARY TABLE 和 ON COMMIT DELETE ROWS

```
CREATE
    GLOBAL TEMPORARY TABLE
        schema1.temp_tbl1 (
            col1 VARCHAR2 (400)
            ,col2 DATE NOT NULL
        )
            ON COMMIT DELETE ROWS
;
```

```
CREATE
    GLOBAL TEMPORARY TABLE
        schema1.temp_tbl1
        (
            col1 VARCHAR2 (400),
            col2 DATE NOT NULL
        )
          ON COMMIT DELETE ROWS
;
```

临时表迁移后

图 5-7 迁移为 LOCAL TEMPORARY TABLE 和 ON COMMIT PRESERVE ROWS

```
CREATE
    LOCAL TEMPORARY TABLE
        schema1_temp_tbl1 (
            col1 VARCHAR2 (400)
            ,col2 DATE NOT NULL
        )
            ON COMMIT PRESERVE ROWS
;
```

```
CREATE
    LOCAL TEMPORARY TABLE
        schema1_temp_tbl1
        (
            col1 VARCHAR2 (400),
            col2 DATE NOT NULL
        )
          ON COMMIT PRESERVE ROWS
;
```

5.7.1.3 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 ;
```

5.7.1.4 索引

在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 5-8 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 5-9 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 )) );
```

5.7.1.5 视图

视图是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。

以下为视图迁移前后的语法示例：

表名称不带模式名

图 5-10 输入：视图中不带模式名的表 tab1、tab2

```
CREATE
    OR REPLACE VIEW schema1.v_view_name AS SELECT
                dict_code code
                ,dict_name name
          FROM
                tab1
          WHERE
                BEAN_CODE = 'LOA_PERSONAL_ACCOUNT#PRTYCODE'
                AND WORK_WT = (
                     SELECT
                               MAX( WORK_DT )
                          FROM
                               tab2
                          WHERE
                               BEAN_CODE = 'LOA_PERSONAL_ACCOUNT#PRTYCODE'
                )
                AND WORK_WT = (
                     SELECT
                               MAX( WORK_DT )
                          FROM
                               schema2.tab3
                          WHERE
                               BEAN_CODE = 'LOA_PERSONAL_ACCOUNT#PRTYCODE'
                )
;
```

表名称包含模式名

图 5-11 输出：用视图的模式名去修饰表

```
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'
                )
    )
;
```

5.7.1.6 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。DWS中，MAXVALUE最大可设为9223372036854775807。
- 在迁移序列之前，请复制sequence_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**执行自定义数据库脚本**。

序列

输入：CREATE SEQUENCE

```
CREATE SEQUENCE GROUP_DEF_SEQUENCE
 minvalue 1
 maxvalue 100000000000000000000
 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
        ,9999999999999999999999999999
        ,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DSC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，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sequence_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**执行自定义数据库脚本**。

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DSC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，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sequence_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**执行自定义数据库脚本**。

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
;
```

5.7.1.7 PURGE

在Oracle中，DROP TABLE语句用于将一张表放入回收站。而PURGE语句则不同，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，也可以是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。

通过Oracle语法迁移后，查询中将不含PURGE。

以下示例分别展示PURGE在迁移前后语法的变化：

PURGE 迁移前

图 5-12 输入：包含 PURGE 的语句

```
Execute immediate 'Drop table
table1 purge' ;

drop table test.emp purge ;
```

PURGE 迁移后

图 5-13 输出：语句中不包含 PURGE

```
Execute immediate 'Drop table table1' ;


drop table test.emp ;
```

5.7.1.8 数据库关键字

DSC支持DWS关键字，如NAME、LIMIT、OWNER、KEY和CAST。这些关键字必须放在双引号内。

DWS 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：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DWS关键字OWNER

```
SELECT
        owner
    FROM
        Test_Col;
```

输出

```
SELECT
        "OWNER"
    FROM
        Test_Col;
```

输入：DELETE，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：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DWS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DWS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)
	);

5.7.2 COMPRESS 短语

该功能通过对COMPRESS设置，迁移过程中默认注释掉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)
) ;
```

5.7.3 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) ;
```

5.7.4 自定义表空间

本节主要介绍自定义表空间的迁移语法。

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))*/
;
```

5.7.5 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，DWS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

DWS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*/
 )
```

5.7.6 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)
);
```

5.7.7 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*/
 );
```

5.7.8 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**

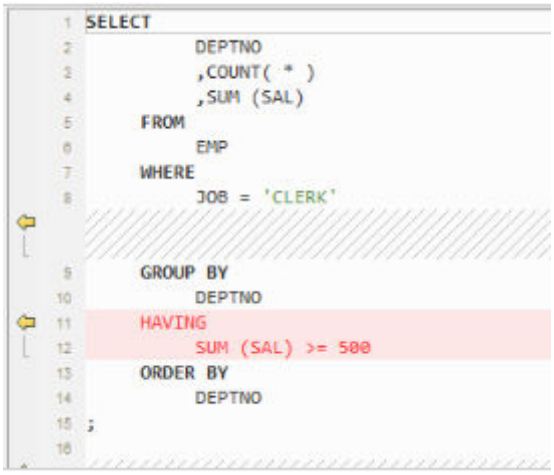
   HAVING子句必须出现在GROUP BY子句后面，而Oracle允许HAVING在GROUP BY子句之前或之后。在目标数据库中，HAVING子句被移至GROUP BY子句之后。

   图 5-14 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 5-15 输出：子句顺序



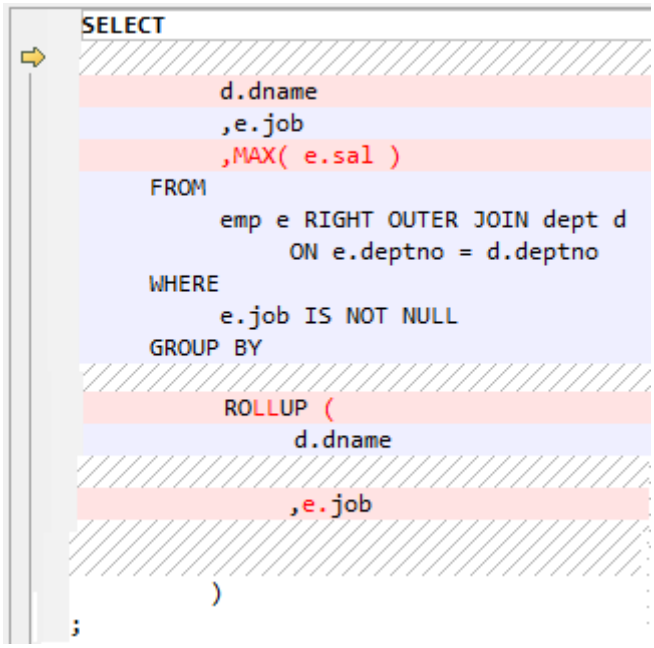
```
SELECT
        DEPTNO
        ,COUNT( * )
        ,SUM (SAL)
    FROM
        EMP
    WHERE
        JOB = 'CLERK'

    GROUP BY
        DEPTNO
    HAVING
        SUM (SAL) >= 500
    ORDER BY
        DEPTNO
;
```

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DWS中不可用，可通过UNION ALL操作符实现。

图 5-16 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 5-17 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。

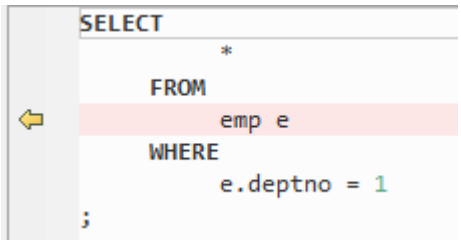
3. **括号中的表名**

   表名不需要在括号内指定，而Oracle允许使用括号。

图 5-18 输入：括号中的表名

```
SELECT
        *
    FROM
        (emp) e
    WHERE
        e.deptno = 1
;
```

图 5-19 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 5-20 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 5-21 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 5-22 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 5-23 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。

DSC使用多种方法将MERGE迁移到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 5-24 输入：MERGE（1）

```
MERGE INTO
     student a
          USING (
          SELECT
                    id
                    ,sname
                    ,score
               FROM
                    student_n
     ) b
          ON( a.id = b.id ) WHEN MATCHED
     THEN UPDATE

          SET
               a.sname = b.sname
               ,a.score = b.score DELETE


                    WHERE


                         a.score < 640

;
```

图 5-25 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 5-26 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 5-27 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;
```

5.7.9 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 5-28 输入：ROWID

```
SELECT
          empid
          ,ename
          ,ROWID
     FROM
          employees
;
```

图 5-29 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 5-30 输入：ROWNUM

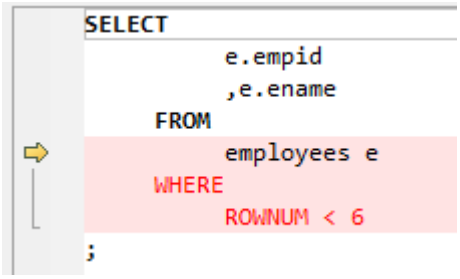
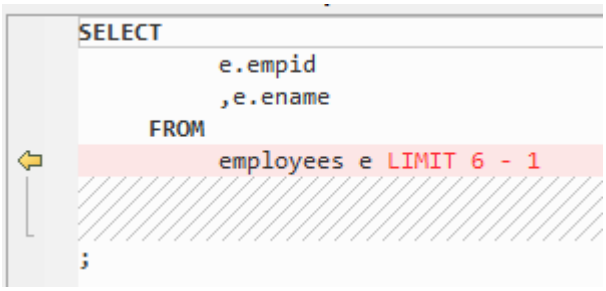


图 5-31 输出：ROWNUM



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，DWS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
       );
```

5.7.10 OUTER JOIN

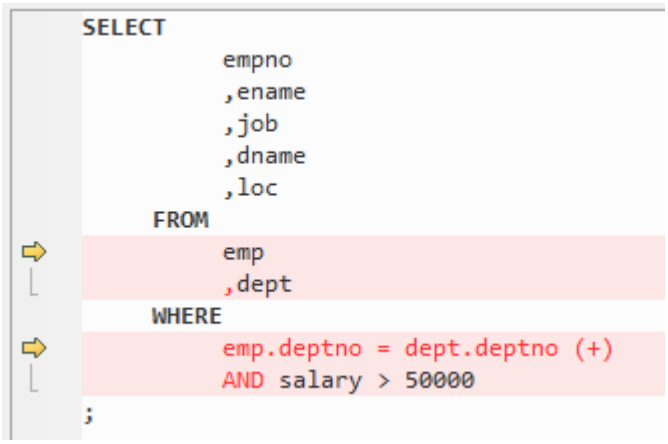
本节主要介绍Oracle OUTER JOIN的迁移语法。迁移语法决定了关键字/功能的迁移方式。

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。

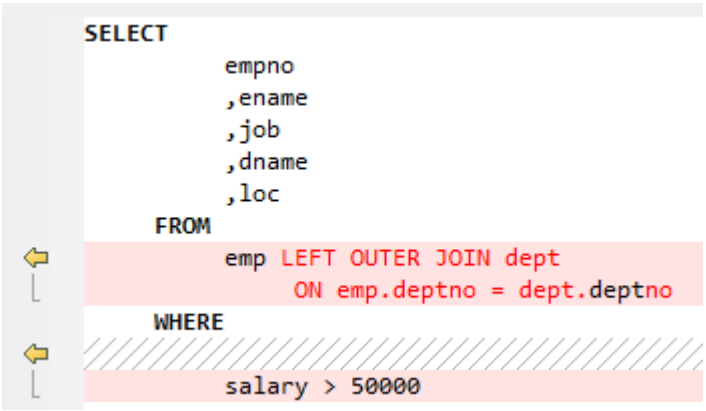
DWS不支持“+”操作符。该操作符的功能通过LEFT OUTER JOIN和RIGHT OUTER JOIN关键词实现。

图 5-32 输入：OUTER JOIN



```
SELECT
        empno
        ,ename
        ,job
        ,dname
        ,loc
    FROM
        emp
        ,dept
    WHERE
        emp.deptno = dept.deptno (+)
        AND salary > 50000
;
```

图 5-33 输出：OUTER JOIN



```
SELECT
        empno
        ,ename
        ,job
        ,dname
        ,loc
    FROM
        emp LEFT OUTER JOIN dept
            ON emp.deptno = dept.deptno
    WHERE

        salary > 50000
```

5.7.11 OUTER QUERY (+)

DWS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
;
```

5.7.12 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;
```

5.7.13 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**日期函数**~**正则表达式函数**章节。

5.7.13.1 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，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**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' */))
 );
```

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，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**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，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. 复制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**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 5-34 输入：SYSTIMESTAMP

```
SELECT
        SYSTIMESTAMP
    FROM
        tab1
;
```

图 5-35 输出：SYSTIMESTAMP

```
SELECT
        CURRENT_TIMESTAMP
    FROM
        tab1
;
```

5.7.13.2 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，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);
```

5.7.13.3 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

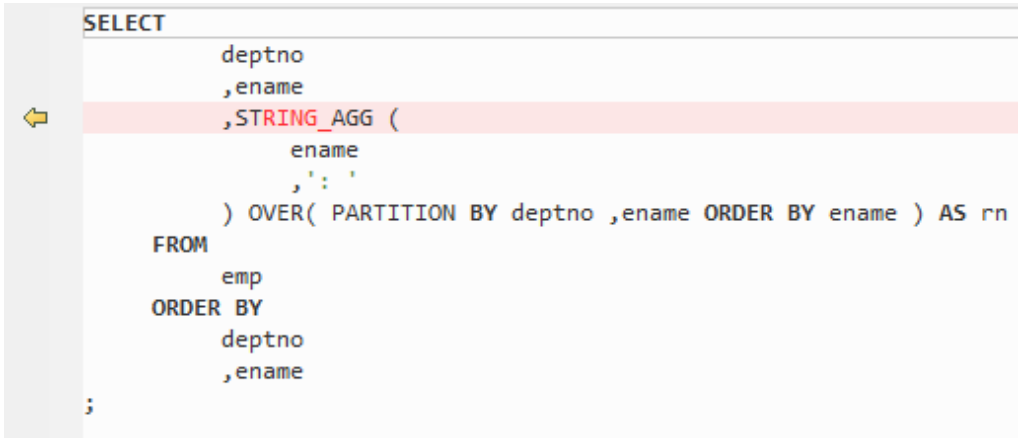
LISTAGG

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 5-36 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 5-37 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 5-38 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 5-39 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。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 5-40 输入：引号

```
SELECT
          q'[It's a string quote operator.]'
     FROM
          dual
;
```

图 5-41 输出：引号

```
SELECT
          $q$It's a string quote operator.$q$
     FROM
          dual
;
```

5.7.13.4 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
;
```

5.7.13.5 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"
```

5.7.14 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

DWS暂不支持set define off/on、spool off，经过DSC工具转换后，在目标数据库中将相关命令注释掉。

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

变量赋值

END

EXCEPTION处理

子事务处理

STRING

LONG

RESULT_CACHE

包含空格的关系运算符

替换变量

PARALLEL_ENABLE

TRUNCATE TABLE

ALTER SESSION

AUTONOMOUS

过程调用

EDITIONABLE

DWS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 5-42 输入：PL/SQL

```
BEGIN
 ...
 v_rowcount : = SQL%ROWCOUNT;
 ..
END;
```

图 5-43 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

DWS不支持EXCEPTION处理。要将脚本迁移，必须将exceptionHandler参数设置为True。

对于DSC此参数必须设置为默认值False。

图 5-44 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 5-45 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 5-46 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 5-47 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

DWS不支持Oracle PL/SQL数据类型STRING。使用VARCHAR来处理该数据类型。

图 5-48 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 5-49 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 5-50 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 5-51 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

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 5-52 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 5-53 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.quotation_id ,cur_1.item_code
 ,cur_1.refresh_date ,cur_1.split_date ,cur_1.error_msg ,cur_1.created_by ,cur_1.creation_date ,cur_1.last_updat
ed_by ,cur_1.last_update_date ) ;
END LOOP ;
---
WHEN OTHERS THEN
po_error_msg := 'Others Exception raise in ' || SAD.SAD_CALC_ITEM_PKG_TEST_OB#func_name ( ) || ',' ||
SQLERRM ;
END ;
```

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 ;
/
```

5.7.15 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

DWS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DWS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。

5.7.16 PL/SQL 包

本节主要介绍Oracle PL/SQL包（详情请参见**包**）和REF CURSOR（详情请参见**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**包**~**数据类型**章节。

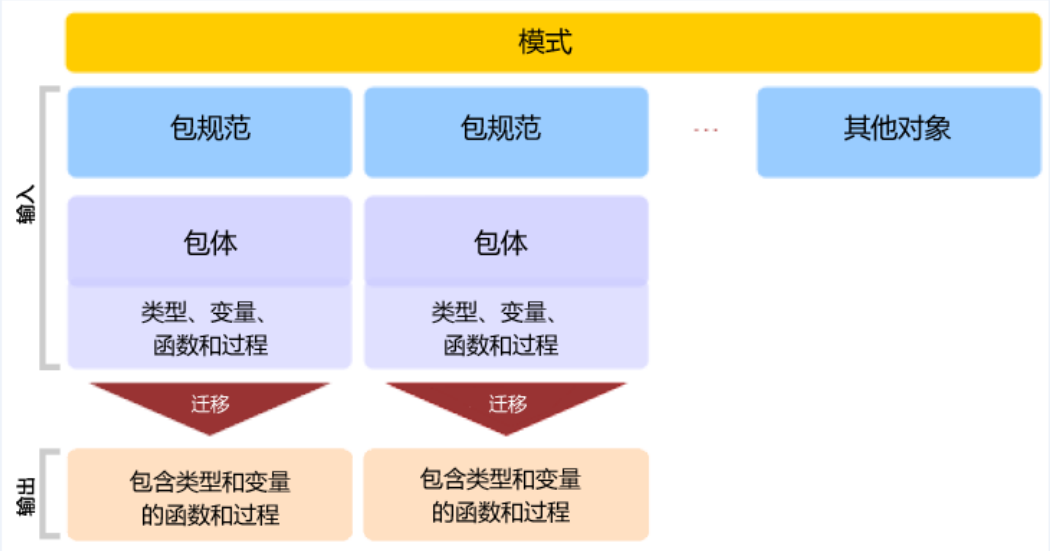
5.7.16.1 包

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 5-54 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DWS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 ;
/
```

5.7.16.2 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**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，DWS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_NUMBE
R_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_HEADER_WAITING_SPLI
T_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。

5.7.16.3 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 ;
/
```

5.7.16.4 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 ,DEFA
ULT_VALUE ,EXPRESSION_I )
    VALUES ( UPPER( current_schema
( ) ) ,UPPER( 'pkg_refcur' ) ,'B' ,UPPER( 'var_num' ) ,UPPER( 'NUMBER' ) ,false ,NULL ,false ) ;
END ;
/

CREATE
    OR REPLACE PROCEDURE pkg_refcur#p_get_employees ( v_id IN INTEGER ,po_results OUT
SYS_REFCURSOR ) is vemp_rc SYS_REFCURSOR ;
    Begin
        OPEN po_results for SELECT
                *
            FROM
                emp_o e
            WHERE
                e.id = v_id ;
                EXCEPTION WHEN OTHERS
            THEN RAISE ;
    END ;
    /
```

5.7.16.5 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
;
```

5.7.17 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;
```

5.7.18 授予执行权限

此功能授予用户特定包的特定权限。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' )::VARCHAR2(30);
    MIG_PV_VAL_DUMMY_G_FUNC_NAME VARCHAR2(100) :=
MIG_ORA_EXT.MIG_FN_GET_PKG_VARIABLE( 'SAD' ,'bas_dml_lookup_pkg' ,'g_func_name' )::VARCHAR2(100);
    l_func_name VARCHAR2 ( 100 ) ;

BEGIN
    l_func_name := MIG_PV_VAL_DUMMY_G_PKG_NAME || '.' || MIG_PV_VAL_DUMMY_G_FUNC_NAME ;
    MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE
( 'SAD' ,'bas_dml_lookup_pkg' ,'g_func_name' ,MIG_PV_VAL_DUMMY_G_FUNC_NAME ) ;
    MIG_ORA_EXT.MIG_FN_SET_PKG_VARIABLE
( 'SAD' ,'bas_dml_lookup_pkg' ,'g_pkg_name' ,MIG_PV_VAL_DUMMY_G_PKG_NAME ) ;

    RETURN l_func_name ;
END ;
/

CREATE OR REPLACE PROCEDURE SAD.bas_dml_lookup_pkg#data_change_logs
  ( pi_table_name IN VARCHAR2
    , pi_table_key_columns IN VARCHAR2
    , po_error_msg OUT VARCHAR2 )
PACKAGE
IS
BEGIN
        ...
END ;
/

GRANT EXECUTE ON FUNCTION SAD.bas_dml_lookup_pkg#bas_dml_lookup_pkg#func_name() TO eip_sad;
GRANT EXECUTE ON FUNCTION SAD.bas_dml_lookup_pkg#data_change_logs(VARCHAR2, VARCHAR2) TO
eip_sad;
```

5.7.19 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.
```

5.7.20 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_SERVICE_PKG.SQL, BAS_LOOKUP_MISC_PKG.SQL

输入：

```
CREATE OR REPLACE PACKAGE BODY "SAD"."BAS_DML_SERVICE_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_SERVICE_PKG;
```

输出：

```
CREATE
OR REPLACE PROCEDURE SAD.BAS_DML_SERVICE_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_SERVICE_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_SERVICE_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_SERVICE_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 ;
/
```

5.7.21 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)*/;
```

5.8 Netezza 语法迁移

5.8.1 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 (AG_NO national character varying(50) not null, AG_CATEG_CD national character varying(12) not null, AMT_TYPE_CD national character varying(12) not null, DATA_START_DT date not null, CCY_CD national character varying(3) not null, DATA_END_DT date) DISTRIBUTE ON (AG_NO, AG_CATEG_CD, AMT_TYPE_CD) ORGANIZE ON (AG_CATEG_CD, AMT_TYPE_CD, DATA_END_DT) ;	CREATE TABLE N_AG_AMT_H (AG_NO national character varying(50) not null, AG_CATEG_CD national character varying(12) not null, AMT_TYPE_CD national character varying(12) not null, DATA_START_DT date not null, CCY_CD national character varying(3) not null, DATA_END_DT date) WITH(ORIENTATION=COLUMN) DISTRIBUTE BY HASH (AG_NO, AG_CATEG_CD, AMT_TYPE_CD) /* ORGANIZE ON (AG_CATEG_CD, AMT_TYPE_CD, DATA_END_DT) */ ;

ORGANIZE ON

ORGANIZE ON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) */ ;

5.8.2 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
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;
	/

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;
 	
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;
	/

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;
	/

5.8.3 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_ST
ART,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_ST
	ART,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; O_RETURN INTEGER; BEGIN O_RETURN := 0; EXECUTE IMMEDIATE 'INSERT INTO TMPO_HXYW_LNSACCTINFO_H1 (ACCTNO, ACCTYPE, SUBCTRLCODE, CCY, NAME) SELECT ACCTNO, ACCTYPE, SUBCTRLCODE, CCY, NAME FROM O_HXYW_LNSACCTINFO T WHERE NOT EXISTS (SELECT 1 FROM O_HXYW_LNSACCT T1 WHERE T1.DATA_START_DT<='''||V_PAR_DAY||''' AND T.MD5_VAL=T1.MD5_VAL)'; O_RETURN := ROW_COUNT; RETURN O_RETURN; END; END_PROC;	CREATE OR REPLACE FUNCTION "EDW"."SP_NTZ_ROWCOUNT" (CHARACTER VARYING(8)) RETURN INTEGER AS V_PAR_DAY ALIAS for $1; O_RETURN INTEGER; BEGIN O_RETURN := 0; EXECUTE IMMEDIATE 'INSERT INTO TMPO_HXYW_LNSACCTINFO_H1 (ACCTNO, ACCTYPE, SUBCTRLCODE, CCY, NAME) SELECT ACCTNO, ACCTYPE, SUBCTRLCODE, CCY, NAME FROM O_HXYW_LNSACCTINFO T WHERE NOT EXISTS (SELECT 1 FROM O_HXYW_LNSACCT T1 WHERE T1.DATA_START_DT<='''||V_PAR_DAY||''' AND T.MD5_VAL=T1.MD5_VAL)'; O_RETURN := SQL%ROWCOUNT; RETURN O_RETURN; END; /

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_schema = '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_schema => 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;
END_PROC;
	CREATE OR REPLACE FUNCTION
	sp_ntz_char_with_any
	(NATIONAL CHARACTER VARYING(10)
	, CHARACTER VARYING)
	RETURN CHARACTER VARYING
	AS
	I_LOAD_DT ALIAS FOR $1 ;
	/* ETL Date */
	V_TASK_ID ALIAS FOR $2 ;
	BEGIN
	RETURN I_LOAD_DT || ',' || V_TASK_ID;
	END;
	/

Numeric (ANY)

Netezza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
ERROR_INFO NVARCHAR(2000) := '';
 	
p_l_begindate ALIAS FOR $1;
p_l_enddate ALIAS FOR $2;
BEGIN
ERROR_INFO := CRHSP_CRH_ETL_EXCHDATE(p_l_begindate,p_l_enddate);
RETURN ERROR_INFO;
 	
EXCEPTION
WHEN TRANSACTION_ABORTED THEN
ROLLBACK;
BEGIN
ERROR_INFO := SQLERRM||' sp_o_transaction_aborted:';
RETURN ERROR_INFO;
END;
 	
WHEN OTHERS THEN
BEGIN
ERROR_INFO := SQLERRM||' sp_o_transaction_aborted:';
RETURN ERROR_INFO;
END;
END;
END_PROC;
	CREATE OR REPLACE FUNCTION sp_ntz_transaction_aborted
	(NUMERIC
	, NUMERIC)
	RETURN NATIONAL CHARACTER VARYING
	AS
	ERROR_INFO NCHAR VARYING(2000) := '';
	
	p_l_begindate ALIAS FOR $1;
	p_l_enddate ALIAS FOR $2;
	BEGIN
	ERROR_INFO := CRHSP_CRH_ETL_EXCHDATE(p_l_begindate,p_l_enddate);
	RETURN ERROR_INFO;
	
	EXCEPTION
	WHEN INVALID_TRANSACTION_TERMINATION THEN
	ROLLBACK;
	BEGIN
	ERROR_INFO := SQLERRM||' sp_o_transaction_aborted:';
	RETURN ERROR_INFO;
	END;
	
	WHEN OTHERS THEN
	BEGIN
	ERROR_INFO := SQLERRM||' sp_o_transaction_aborted:';
	RETURN ERROR_INFO;
	END;
	END;
	/

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) :=
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklm
nopqrstuvwxyz0123456789+/';
v_I int := 0;
v_J int := 0;
v_K int := 0;
v_N int := 0;
v_out Numeric(38,0) := 0;
I_LOAD_DT ALIAS FOR $1;
 	
BEGIN
v_N:=Length(v_B64);
FOR v_I In Reverse 1..Length(IN_base64)
LOOP
v_J:=Instr(v_B64,Substr(IN_base64,v_I,1))-1;
If v_J <0 Then
RETURN -1;
End If;
V_Out:=V_Out+v_J*(v_N**v_K);
v_K:=v_K+1;
END LOOP;
RETURN V_Out;
END
END_PROC;
	CREATE OR REPLACE FUNCTION
	sp_ntz_end_wo_semicolon
	(NATIONAL CHARACTER VARYING(10))
	RETURN CHARACTER VARYING
	AS
	v_B64 Varchar(64) :=
	'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklm
	nopqrstuvwxyz0123456789+/';
	v_I int := 0;
	v_J int := 0;
	v_K int := 0;
	v_N int := 0;
	v_out Numeric(38,0) := 0;
	I_LOAD_DT ALIAS FOR $1;
	
	BEGIN
	v_N:=Length(v_B64);
	FOR v_I In Reverse 1..Length(IN_base64)
	LOOP
	v_J:=Instr(v_B64,Substr(IN_base64,v_I,1))-1;
	If v_J <0 Then
	RETURN -1;
	End If;
	V_Out:=V_Out+v_J*(v_N**v_K);
	v_K:=v_K+1;
	END LOOP;
	RETURN V_Out;
	END;
	/

LOOP

Netezza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;
	/

DWS 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 INSTR(VIEWSQL1,'v_p_etldate')
> 0
LOOP
VIEWSQL1 :=
SUBSTR(VIEWSQL1,1,INSTR(VIEWSQL1,'v_p_etldat
e') - 1)||''''||I_LOAD_DT||''''||
SUBSTR(VIEWSQL1,INSTR(VIEWSQL1,'v_p_etldate')
+ 11);
END LOOP;
 	
EXECUTE IMMEDIATE VIEWSQL1;
END LOOP;
END;
 	
RETURN 0;
END;
END_PROC;
	CREATE OR REPLACE FUNCTION
	sp_ntz_declare_inside_begin
	(NATIONAL CHARACTER VARYING(10))
	RETURN INTEGER
	AS
	I_LOAD_DT ALIAS FOR $1;
	BEGIN
	DECLARE
	MYCUR RECORD;
	VIEWSQL1 NCHAR
	VARYING(4000);
	BEGIN
	FOR MYCUR IN (SELECT VIEWNAME,VIEWSQL
	FROM
	T_DDW_AUTO_F5_VIEW_DEFINE
	WHERE OWNER = 'ODS_PROD')
	LOOP
	VIEWSQL1 := MYCUR.VIEWSQL;
	WHILE INSTR(VIEWSQL1,'v_p_etldate')
	> 0
	LOOP
	VIEWSQL1 :=
	SUBSTR(VIEWSQL1,1,INSTR(VIEWSQL1,'v_p_etldat
	e') - 1)||''''||I_LOAD_DT||''''||
	SUBSTR(VIEWSQL1,INSTR(VIEWSQL1,'v_p_etldate')
	+ 11);
	END LOOP;
	
	EXECUTE IMMEDIATE VIEWSQL1;
	END LOOP;
	END;
	
	RETURN 0;
	END;
	/

EXECUTE AS CALLER

Netezza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;
END;
END_PROC;
	CREATE OR REPLACE FUNCTION sp_ntz_exec_as_caller
	(CHARACTER VARYING(512))
	RETURN INTEGER
	SECURITY INVOKER
	AS
	SQL ALIAS FOR $1;
	BEGIN
	EXECUTE IMMEDIATE SQL;
	RETURN 0;
	END;
	/
	---------- 	
	CREATE OR REPLACE FUNCTION sp_ntz_exec_as_owner
	(CHARACTER VARYING(512))
	RETURN INTEGER
	SECURITY DEFINER
	AS
	SQL ALIAS FOR $1;
	BEGIN
	EXECUTE IMMEDIATE SQL;
	RETURN 0;
	END;
	/

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;
	/

5.8.4 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 FROM tab123;	SELECT A.ETL_DATE, A.BRANCH_CODE, A.CUST_NO , NVL (B.RES_STOCK,0) AS RES_STOCK , NVL (B.ZY_VOL ,0) AS ZY_VOL , NVL (B.ZJ_VOL,0) AS ZJ_VOL FROM tab123;

NVL

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;	SELECT 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;
	/

5.8.5 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

5.8.6 DML（Netezza）

DWS 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;

5.8.7 Unique 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, prod_name national character varying(32) not null, prod_desc clob) DISTRIBUTE ON (prod_no) ORGANIZE ON (prod_no, prod_name) ; ---------- CREATE TABLE prod (prod_no number(6) not null PRIMARY KEY, prod_name national character varying(32) not null, prod_desc clob) DISTRIBUTE ON (prod_no) ORGANIZE ON (prod_no, prod_name) ; ---------- CREATE TABLE prod (prod_no number(6) not null, prod_name national character varying(32) not null, prod_desc clob, constraint uq_prod UNIQUE (prod_no)) DISTRIBUTE ON (prod_no) ORGANIZE ON (prod_no, prod_name) ; ---------- CREATE TABLE prod (prod_no number(6) not null, prod_name national character varying(32) not null, prod_desc clob) DISTRIBUTE ON (prod_no) ORGANIZE ON (prod_no, prod_name) ; ALTER TABLE prod ADD constraint uq_prod UNIQUE (prod_no);	CREATE TABLE prod (prod_no number(6) not null /* unique */, prod_name national character varying(32) not null, prod_desc clob) WITH(ORIENTATION=COLUMN) DISTRIBUTE BY HASH (prod_no) /* ORGANIZE ON (prod_no, prod_name) */ ; ---------- CREATE TABLE prod (prod_no number(6) not null /* CONSTRAINT UQ_prod unique */, prod_name national character varying(32) not null, prod_desc clob) WITH(ORIENTATION=COLUMN) DISTRIBUTE BY HASH (prod_no) /* ORGANIZE ON (prod_no, prod_name) */ ; ---------- CREATE TABLE prod (prod_no number(6) not null /* PRIMARY KEY */, prod_name national character varying(32) not null, prod_desc clob) WITH(ORIENTATION=COLUMN) DISTRIBUTE BY HASH (prod_no) /* ORGANIZE ON (prod_no, prod_name) */ ; ---------- CREATE TABLE prod (prod_no number(6) not null, prod_name national character varying(32) not null, prod_desc clob /*, constraint uq_prod UNIQUE (prod_no) */) WITH(ORIENTATION=COLUMN) DISTRIBUTE BY HASH (prod_no) /* ORGANIZE ON (prod_no, prod_name)*/ ; ---------- CREATE TABLE prod (prod_no number(6) not null, prod_name national character varying(32) not null, prod_desc clob) DISTRIBUTE BY HASH (prod_no) /*ORGANIZE ON (prod_no, prod_name)*/ ; /* ALTER TABLE prod

Netezza语法	迁移后语法
	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”。

5.9 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。

5.10 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 5-39 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。

问题现象	原因及解决方案
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。
Disk is almost full. Please clear the space and re-run the tool.	**原因：**磁盘空间不足。 **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。
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**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。
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。
java.lang.OutOfMemoryError: GC overhead limit exceeded.	**原因：**在迁移操作期间，如果内存使用超过设置的值，DSC将有该错误提示并退出。 **解决方案：**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。

问题现象	原因及解决方案
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 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 **********

错误码

表 5-40 错误码

错误码	错误信息
Teradata	
DSC_ERR_003_001	Query/statement is not supported since the Teradata view "dbc.indices" is supported only for the indextype P and Q.

错误码	错误信息
DSC_ERR_003_002	Error in Bteq processing. Something went wrong while processing the BTEQ commands.
DSC_ERR_003_003	Query/statement is not supported in ddl DSC. Please check the same and refer user manual for the supported feature list.
DSC_ERR_003_004	Unsupported format decimal format like ZZZ99Z, ZZZ.ZZ9.
DSC_ERR_003_005	The tool does not support the "IN/NOT IN to EXISTS/NOT EXISTS conversion" for the query in which its outer query refers multiple tables and the column(s) specified with IN / NOT IN operator do not have table reference.
DSC_ERR_003_006	The tool does not support the query in which its outer query refers multiple tables and the column(s) specified with IN / NOT IN operator do not have table reference.
DSC_ERR_003_007	Primary Index without column is not supported.
DSC_ERR_003_008	TeradataQuerySplitter config file contains list is not supported.
DSC_ERR_003_009	Gauss does not support WITH CHECK OPTION in CREATE VIEW. Please enable the config_param tdMigrateVIEWCHECKOPTION to comment the WITH CHECK OPTION syntax in the statement.
DSC_ERR_003_010	Gauss does not have an equivalent syntax for CHARACTER SET & CASE SPECIFIC option in column-level. Please enable the config_param tdMigrateCharsetCase to comment the CHARACTER SET & CASE SPECIFIC option syntax in the statement.
DSC_ERR_003_011	Gauss does not have equivalent syntax for LOCK option in CREATE VIEW and INSERT statement. You can rewrite this statement or set the configuration parameter tdMigrateLOCKOption to TRUE to comment the LOCK syntax in this statement.

错误码	错误信息
DSC_ERR_003_012	Invalid width (Number of rows) parameter in MDIFF function.
DSC_ERR_003_013	First 2 parameters should be present in MDIFF function.
DSC_ERR_003_014	Query/statement is not supported as ORDER BY clause is not present in TOP WITH TIES. Please check the same and refer user manual.
DSC_ERR_003_015	Column mismatch for the TITLE conversion.
DSC_ERR_003_016	Query/statement is not supported as same Table alias is addressed in both inner and outer query. Please check the same and refer user manual for the supported feature list.
DSC_ERR_003_017	Sub query list does not have columns.
DSC_ERR_003_018	Number of expressions specified in the outer query does not match with inner query.
DSC_ERR_003_019	Error while loading the .RUN FILE from given location.
DSC_ERR_003_020	Unable to delete the file, file not found.
DSC_ERR_003_021	Unable to delete the file, failed with IOEXception.
DSC_ERR_003_022	Please specify the value for environment_file_path parameter in features-teradata.properties.
Application	
DSC_ERR_004_001	Application has timed out, exceeded the hours specified in the config file. Please configure the Timeout parameter in the application.properties to higher value.
DSC_ERR_004_002	Error while loading the property files from config directory.
DSC_ERR_004_003	Error while loading the property files from config directory, directory is not readable.
DSC_ERR_004_004	Error while loading the property file.

错误码	错误信息
DSC_ERR_004_005	Unable to load the JSON file.
DSC_ERR_004_006	DSC tool does not support this Conversion type provided.
DSC_ERR_004_007	Error occurred while framing output replacement query.
DSC_ERR_004_008	Invalid index value while parsing the script.
DSC_ERR_004_009	Error in conversion process, unable to convert the script.
DSC_ERR_004_010	No SQL files found in the input directory with the extension specified in the fileExtension property in application.properties.
DSC_ERR_004_011	The query length parameter (MaxSqlLen) value is not valid.
DSC_ERR_004_012	Since the input folder has write privileges to Group and/or Others, process is stopped due to security reason.
DSC_ERR_004_013	Since the output directory has write privileges to Group and/or Others, process is stopped due to security reason.
DSC_ERR_004_014	Disk is almost full. Please clear the space and re-run the tool.
DSC_ERR_004_015	DSC has been cancelled as configured by the user.
DSC_ERR_004_016	Error occurred while formatting the sql scripts.
DSC_ERR_004_017	Invalid index specified for fetching the element from list while formatting the scripts
DSC_ERR_004_018	Error occurred while converting from string to integer.
DSC_ERR_004_019	Input File is modified while DSC is in progress.
DSC_ERR_004_020	Process is null, unable to read encoding format.

错误码	错误信息
DSC_ERR_004_021	Target File does not have write permissions.
DSC_ERR_004_022	The target directory does not have write privileges to Group and/or Others, process is stopped due to security reason.
DSC_ERR_004_023	PL/SQL object contains incorrect DDL/ Query. Please check the script for the query position specified in the log.
DSC_ERR_004_024	PreQueryValidation failed due to bracket mismatch or invalid terminator.
DSC_ERR_004_025	Conversion task name is not valid.
DSC_ERR_004_026	Database entered by the user is not supported by the DSC tool.
DSC_ERR_004_027	Gauss db password should not be empty.
DSC_ERR_004_028	Gauss db password should not be empty.
DSC_ERR_004_029	Target db entered in the Gaussdb.properties is not valid.
DSC_ERR_004_030	User name entered in the Gaussdb.properties is empty.
DSC_ERR_004_031	Port entered in the Gaussdb.properties is not valid.
DSC_ERR_004_032	IP entered in the Gaussdb.properties is not valid.
DSC_ERR_004_033	Database name entered in the Gaussdb.properties is empty.
DSC_ERR_004_034	DSC Application failed to start.
DSC_ERR_004_035	Since the environment variable path has write privileges to Group and/or Others, process is stopped due to security reason.
DSC_ERR_004_036	Error while loading environment parameter File.

错误码	错误信息
DSC_ERR_004_037	Invalid input (empty/space/string value) for the parameter NoOfThreads in application.properties. Hence taking the default processes.
DSC_ERR_004_038	Input for the parameter NoOfThreads in application.properties is less than 1. Hence taking the default processes.
DSC_ERR_004_039	Error in processing the DDL query.
DSC_ERR_004_040	Error in processing the PL/SQL query.
DSC_ERR_004_041	Error in post processing the query.
DSC_ERR_004_042	Invalid application timeout value, default to 4 hours.
DSC_ERR_004_043	Error in writing the output file.
DSC_ERR_004_044	Error in reading the input file.
DSC_ERR_004_045	No valid files found in the input directory for migration.
DSC_ERR_004_046	Query is not converted as it contains unsupported keyword.
DSC_ERR_004_047	Error while reading the property.
DSC_ERR_004_048	PreQueryValidation failed due to query exceeds maximum length (MaxSqlLen config parameter).
DSC_ERR_004_049	Thread count entered in the Gaussdb.properties is not valid.
Wrapper	
DSC_ERR_005_003	Reading file Failed with error: File not found Exception.
DSC_ERR_005_004	Reading file Failed with error: IOException.
DSC_ERR_005_005	Root privileged users are not allowed to execute the DSC tool.
DSC_ERR_005_006	Error while getting the id of os user used to execute the DSC tool.
DSC_ERR_005_007	Arguments specified is not valid, please check the user manual for the command line arguments.

错误码	错误信息
DSC_ERR_005_008	File name is not specified for reading the encoding type.
DSC_ERR_005_009	Invalid argument specified for the encoding parameter.
DSC_ERR_005_010	Source database is not set. Please enter a valid source db and refer the user manual for syntax.
DSC_ERR_005_011	Commandline database specified for source to target is not supported by the DSC tool.
DSC_ERR_005_012	Error in loading config file with IOException.
DSC_ERR_005_013	Initial JVM memory is greater than maximum JVM memory.
DSC_ERR_005_014	Invalid value specified for configValue.
DSC_ERR_005_015	Invalid source database specified for source-db option.
DSC_ERR_005_016	Invalid target database specified for target-db option.
DSC_ERR_005_017	Invalid conversion type specified for dsc-type option.
DSC_ERR_005_018	Invalid application language specified for application-lang option.
DSC_ERR_005_019	Conversion-type should be DDL for application-lang type as perl.
DSC_ERR_005_020	Source-db should be teradata for application-lang type as perl.
DSC_ERR_005_021	Please use "-VN [V1R7 | V1R8_330]" or "--version-number [V1R7 | V1R8_330]" to specify the kernel version which can be either V1R7 or V1R8_330.
DSC_ERR_005_022	Input directory does not exist.
DSC_ERR_005_023	Getting path for input directory failed with IOException.
DSC_ERR_005_024	Getting path for output directory failed with IOException.
DSC_ERR_005_025	Setting file permission for output directory failed with IOException.

错误码	错误信息
DSC_ERR_005_026	Creating output directory failed.
DSC_ERR_005_027	Setting file permissions for log directory/file failed with FileException.
DSC_ERR_005_028	Error while connecting to GaussDB, Failed with error.
DSC_ERR_005_029	Error occurred due to file permission while creating or executing the file.
DSC_ERR_005_030	No arguments specified in the commandline.
DSC_ERR_005_031	Error occurred in creating output directory.

5.11 术语表

下表包含缩略语、术语及其说明。

术语	描述
C	
公用表表达式(CTE)	公用表表达式是一个在查询中定义的临时命名结果集，仅可用于更大的查询范围。
D	
数据库(DB)	数据库是一组相关信息的集合，通常是为了使通用的检索变得简单和高效而组织起来的。 数据库属性： • 数据库名称。 • Endian文件格式（BIG_ENDIAN大端或LITTLE_ENDIAN小端）。 • 关系。 • 不存在无关系的数据库。
数据库管理员(DBA)	数据库管理员是负责组织中数据库的安装、配置、升级、管理、监控和维护的人员。 该角色包括开发和设计数据库策略、监控和优化数据库性能和容量，以及规划未来的扩展需求。数据库管理员亦可计划、协调和实施安全措施，以保障资料库的安全。

术语	描述
E	
编码	在信息处理中，编码是一种规则系统，它把字母、单词、声音、图像或手势等信息转换成另一种规则。有时，它以缩短或秘密的形式通过通道进行通讯或存储在介质中。
I	
索引	数据库管理系统中的一种有序数据结构，可加速表内数据的查询和更新。
M	
迁移	迁移是指将源数据库（如Teradata）中的脚本、查询、模式、数据等迁移到目标数据库（如GaussDB (DWS)）。
元数据	元数据是关于数据的数据。元数据定义了数据的属性，用于指定数据的存储位置、历史数据、检索资源数据、记录信息等。
O	
操作系统(OS)	操作系统是管理计算机中所有其他程序的程序，这些程序最初通过引导程序加载到计算机中。
Q	
查询	查询是向数据库发出的信息请求。查询执行SQL语句，并返回该语句定义的结果集。
S	
结构化查询语言(SQL)	一种编程语言，广泛用于访问、更新、管理和查询关系数据库中的数据。
模式	模式是数据库管理系统支持的正式语言中描述的结构。它是指数据的组织，描述数据库是如何构建的。（在关系数据库中，它描述了如何将数据库划分为表。）
T	
Teradata	Teradata是一种关系数据库管理系统。它可用于同时运行多个复杂查询。它支持使用SQL的即席查询，并广泛用于管理大型仓储操作。

术语	描述
表	紧密相关的列的集合。表由包含相同列的不同值的行组成。
V	
视图	视图限制对表的特定行或列的访问。视图可以从一个或多个表中创建，并且由用于创建视图的查询决定。

6 DataCheck

6.1 DataCheck 简介

6.1.1 概述

当用户选择切换到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、sql脚本迁移以及迁移之后数据校对工作，其中，数据校对是保障数据库迁移完备的重要环节。

DataCheck是一款运行在Linux或Windows操作系统上的命令行工具，致力于向用户提供简单、快速、可靠的数据校对服务，通过连接源端数据库以及目标dws数据库，将源端数据库和目标端DWS数据库中的表数据进行校对，保证用户数据迁移前后的一致性。

DataCheck需要连接数据库，可在离线模式下实现数据校对，校对结果会依次写入Excel表格中，并用日志记录操作过程中发生的错误，便于快速定位问题。

数据校验支持的源端数据库

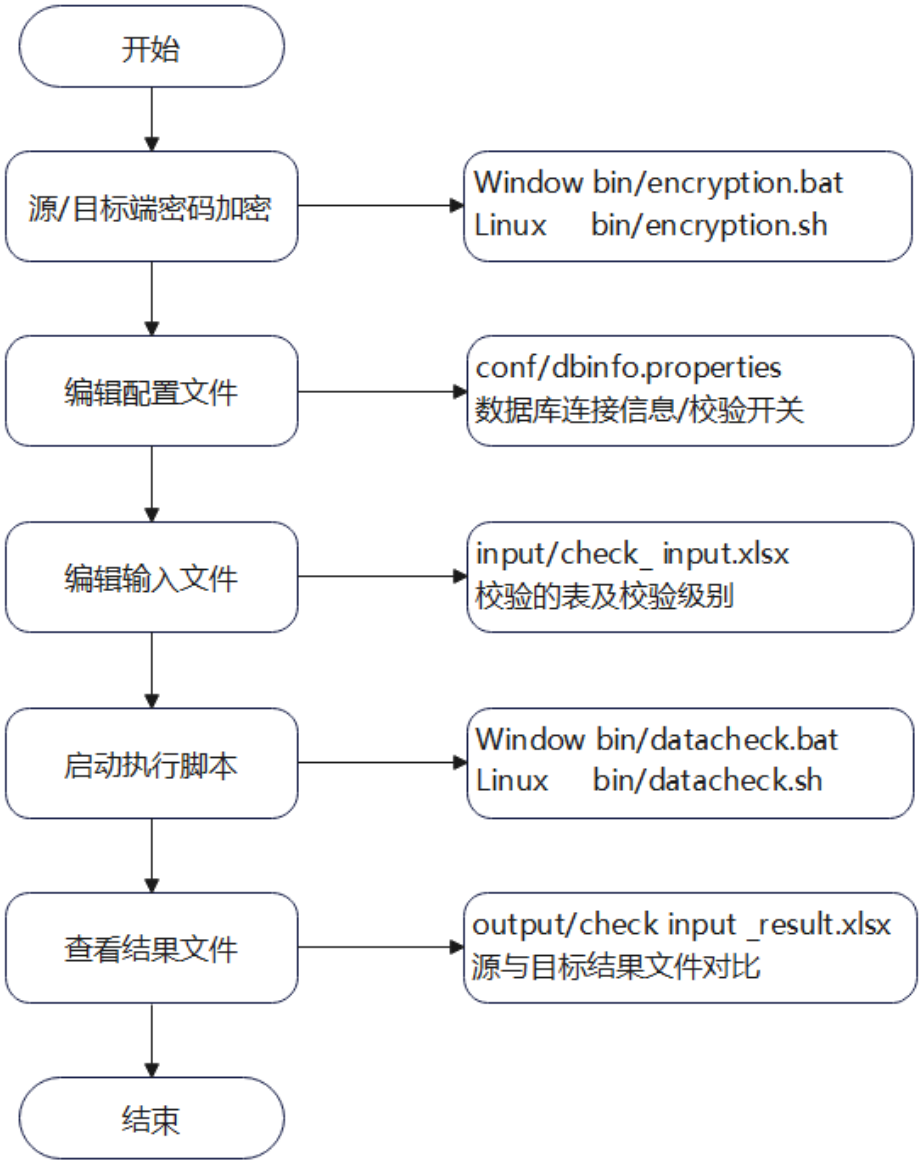
- MySQL(包括 AnalyticDB for MySQL)
- PostgreSQL
- DWS

数据校验流程

DataCheck流程如下：

1. 下载DataCheck的工具包到Linux或Windows服务器并解压。
2. 执行加密命令对源端/目标端数据库登录密码进行加密。
3. 配置dbinfo.properties文件，包含源数据库和目标数据库的相关连接信息以及函数开关信息。
4. 编辑check_input.xlsx文件，输入schema、源数据库表名和dws表名以及校验级别等参数。
5. 执行DataCheck启动命令进行数据校验，校验结果保存在check_input_result.xlsx中。

图 6-1 DataCheck 流程图



6.1.2 运行环境

支持的数据库

DataCheck支持的源数据库如所示。

表 6-1 支持的源数据库

数据库名称	数据库版本
MySQL	8.0
PostgreSQL	42.6.0
DWS	8.1.0及以上集群版本

DataCheck支持的目标数据库如所示。

表 6-2 支持的目标数据库

数据库名称	数据库版本
DWS	8.1.0及以上集群版本

硬件要求

DataCheck对硬件的要求如表6-3所示。

表 6-3 DataCheck 硬件环境要求

硬件	配置
CPU	AMD或Intel Pentium （最小频率：500 MHz）
最小内存	1 GB
磁盘空间	1 GB

软件要求

操作系统要求

DataCheck兼容的操作系统如表6-4所示。

表 6-4 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及以上	SUSE Linux Enterprise Server 11 SP1及以上
	RHEL	RedHat6.4及以上
	CentOS	CentOS6.4及以上
	Windows	7.0，10，11

其他软件要求

DataCheck对其他软件版本的要求如表6-5所示。

表 6-5 其他软件要求

软件	用途
JDK 1.8或JRE1.8	运行DataCheck工具。

6.2 DataCheck 基本功能

统计值校验

- 支持源端为DWS，MySQL，PostgreSQL，BigQuery等数据库与目标端为DWS数据库的数据校验。
- 支持通用类型字段校验：数值、时间、字符类型。
- 支持校验级别设置：包括high、middle、low三种。
- 支持指定schema、表名、列名进行校验。
- 支持指定记录的校验范围，默认为校验所有记录。
- 校验方式涉及COUNT(*)、MAX、MIN、SUM以及抽样明细校验等方式。
- 输出校验结果和相关校验明细说明。

表 6-6 数据校验级别说明

校验级别	校验说明	校验相关语法
低	数据数量校验	条数校验: COUNT(*)
中	• 数据数量校验 • 数值类型校验	• 条数校验: COUNT(*) • 数值校验: MAX, MIN, SUM
高	• 数据数量校验 • 数值类型校验 • 日期类型校验 • 字符类型校验	• 条数校验: COUNT(*) • 数值校验: MAX, MIN, SUM • 日期校验: MAX, MIN • 字符校验：order by limit 1000，读出数据并校验内容是否相同。

元数据校验

- 支持源端为DWS，MySQL，PostgreSQL，BigQuery等数据库与目标端为DWS数据库的表定义校验。
- 支持四大校验类型：字符、整数、小数、时间（包含日期）。
- 列名相同且类型一致，则校验通过。

数据精确对比

- 支持源端为DWS，MySQL，PostgreSQL，BigQuery等数据库与目标端为DWS数据库的数据精确对比。

- 根据主键或指定的列标识的唯一记录，从两端表中查询数据逐列比对，输出比对结果，包括源端或DWS端多出的记录和记录不同的列值。

⚠ 注意

- 数据精确比对，比较耗执行机资源，同时也占用两端数据库的负载，应在业务空闲时执行。
- 精确比对需要指定分批查询数据的条数（默认1000条）和出现差异的结果的数量（默认100条，达到此阈值时精确比对中止）。

6.3 下载并安装 DataCheck

前提条件

- 服务器：Linux或Windows服务器，支持64位操作系统。
- JRE或JDK：系统已安装JDK 1.8或JRE 1.8。
- 网络环境：安装、运行DataCheck工具的服务器，需要与待连接的数据库的网络是互通的。

下载 DataCheck 工具

下载DataCheck客户端软件，请联系技术支持工程师。

安装 DataCheck 工具

DataCheck是一款运行在Linux或Windows操作系统上的命令行工具，可免安装使用，下载软件包后，用户解压软件包即可使用。

Windows:

步骤1 解压DataCheck-*.zip包。

得到DataCheck-*文件夹。

说明

解压DataCheck-*时，可根据需要选择任意文件夹进行解压。

步骤2 进入DataCheck-*目录。

步骤3 找到并查看DataCheck目录中的文件。

解压出来的文件夹和文件说明如表6-7所示。

----结束

Linux操作系统:

步骤1 从DataCheck-*.zip包中提取文件。

```
unzip DataCheck-*.zip
```

步骤2 进入DataCheck目录。

```
cd DataCheck-*
```

步骤3 查看DataCheck目录中的文件。

```
ls
conf   lib   bin   check_input.xlsx
```

----结束

表 6-7 DataCheck 目录

文件或文件夹		说明
DataCheck	bin	存放校验工具启动脚本。 Windows版本：datacheck.bat Linux版本：datacheck.sh
	conf	配置文件，进行源数据库和目的数据库的连接配置和日志打印设置。
	lib	保存校验工具运行所需的相关jar包。
	check_input.xlsx	1、待校验的表信息，包括Schema名、表名、列名等 2、记录用户的校验级别信息和校验规则。已支持3种级别校验，包括high、middle、low，默认为low。
	logs	压缩包中不包含该文件，校验工具执行后自动生成，记录工具运行过程日志。
	check_input_result.xlsx	压缩包中不包含该文件，校验工具执行后会在check_input.xlsx相同路径下生成校验结果文件。

6.4 配置 DataCheck

6.4.1 dbinfo.properties 配置

dbinfo.properties文件中包括一系列应用配置参数，用于连接源端数据库和目标dws数据库，该文件中的参数为通用参数。

设置方法如下。

步骤1 打开conf文件夹中的dbinfo.properties文件。

步骤2 根据实际需要修改dbinfo.properties文件中参数的值。

dbinfo.properties文件中的参数说明见表6-8。

说明

- 参数值不区分大小写。
- 除了列出的参数外，不得更改其他参数值。

步骤3 保存后退出。

----结束

表 6-8 dbinfo.properties 文件的配置参数

参数	说明	取值范围	默认值	样例
src.dbtype	源端数据库类型。	• mysql • pg • oracle • dws • hive • mrs_hive • hologres • greenplum • redshift • bigquery • elasticsearch • synapse	MySQL	src.dbtype=mysql
src.dbname	源端数据库名称。	NA	sys	src.dbname=sys
src.ip	源端数据库ip地址。	NA	NA	src.ip=100.xx.xx.47
src.port	源端数据库端口。	NA	3306	src.port=3306
src.username	源端数据库用户名。	NA	root	src.username=root
src.passwd	源端数据库密码。	NA	NA	src.passwd=123456
dws.dbtype	目标端dws数据库类型。	dws	dws	dws.dbtype=dws
dws.dbname	目标端dws数据库名称。	NA	gaussdb	dws.dbname=gaussdb
dws.ip	目标端dws数据库ip地址。	NA	NA	dws.ip=100.xx.xx.186
dws.port	目标端dws数据库端口。	NA	8000	dws.port=8000
dws.username	目标端dws数据库用户名。	NA	dbadmin	dws.username=dbadmin

参数	说明	取值范围	默认值	样例
dws.passwd	目标端dws数据库密码。	NA	NA	dws.passwd=123456
config.sum.switch	数值校验：求和函数开关。	• on • off	on	config.sum.switch=on
config.data.min.switch	数值校验：最小值函数开关。	• on • off	on	config.data.min.switch=on
config.data.max.switch	数值校验：最大值函数开关。	• on • off	on	config.data.max.switch=on
config.date.min.switch	日期校验：最小值函数开关。	• on • off	on	config.date.min.switch=on
config.date.max.switch	日期校验：最大值函数开关。	• on • off	on	config.date.max.switch=on
config.collate.switch	Collate规则计算的开关配置。 on：启动；off：关闭。	• on • off	on	config.collate.switch=on
config.collate.value	Collate规则的值：“C”、“zh_CN”、“en_US”。	• C • zh_CN • en_US	C	config.dws.collate=C
src.projectid	源端的项目ID，BigQuery源需要配置。	NA	NA	src.projectid=xxxxxxx
src.oauthtype	源端的oauth类型，BigQuery源需要配置。	• 0	0	src.oauthtype
src.oauthserviceacctemail	源端的oauth邮箱，BigQuery源需要配置。	NA	NA	src.oauthserviceacctemail=xxx@sina.com
src.oauthpvtkeypath	源端的oauth认证信息文件的路径，BigQuery源需要配置。	NA	NA	src.oauthpvtkeypath =/opt/temp/analytics-di-dev-a8dfdf.json
config.batch.size	精确比对时，每批查询的数据条数。	NA	1000	config.batch.size = 1000

参数	说明	取值范围	默认值	样例
precise.result.max	精确比对结果中出现差异的最大值，达到此阈值时会终止精确比对。	NA	100	precise.result.max = 100
difference	浮点数比较时允许的误差值。	NA	0.0001	difference = 0.0001
output.file.suffix	输出结果文件的后缀，支持csv/xlsx。	• xlsx • csv	xlsx	output.file.suffix = csv

6.4.2 check_input.xlsx 配置

check_input.xlsx文件中包括用户输入信息：schema、原表名、目标表名、指定列名（缺省为全部列校验）、校验范围、校验级别（缺省为low），排除列，可以按需进行配置。

设置方法如下。

步骤1 打开文件夹中的check_input.xlsx文件。

步骤2 根据实际需要修改check_input.xlsx文件中参数的值。

check_input.xlsx文件中的参数说明见**表6-9**。

说明

- 参数值不区分大小写。
- 除了列出的参数外，不得更改其他参数值。

步骤3 保存后退出。

----结束

表 6-9 check_input.xlsx 文件的配置参数

参数	说明	取值范围	默认值	样例
Source Database Name	指定需要校验的源表所属的database 名称，**可选项**，不填写表示使用dbinfo.properties中的src.dbname。	NA	NA	mydb
Source Schema Name	指定需要校验的源表所属的schema名称，不涉及则不填写。	NA	NA	myschema
Source Table Name	源端数据库需要校验的表名，**必填项**。	NA	NA	order_info

参数	说明	取值范围	默认值	样例
Target Database Name	目标端DWS表所属的database 名称，**可选项**，不填写表示使用dbinfo.properties中的dws.dbname。	NA	NA	dws_db
Target Schema Name	目标端DWS表所属的schema名称，**必填项**。	NA	NA	dws_sch
Target Table Name	目标端dws数据库需要校验的表名，**必填项**。	NA	NA	dws_info
Check Mode	校验模式：统计值校验，精准校验，元数据校验	• Statistics • Preciseness • Metadata	• Statistics	Statistics
Src Row Range(Where sql)	指定源数据表记录的校验范围，默认为校验所有记录。	NA	ALL	where begin_time > '2020-1-1' and begin_time < '2021-1-1'
DWS Row Range(Where sql)	指定目标（DWS）表记录的校验范围，默认为校验所有记录。	NA	ALL	where begin_time > '2020-1-1' and begin_time < '2021-1-1'
Column Range	可选，指定列校验，默认为所有支持类型字段，即：数值、时间、字符类型。	NA	NA	col1,col10,col19
Check Strategy	数据校验级别。	• low • middle • high	low	low
Sort Column	可选，精确校验时根据Sort Column字段进行排序，如果Sort Column为空，则根据主键进行排序，排序后逐行对比。	NA	NA	col1,col10,col19
Column Exclude	可选，排除指定列的校验。	NA	NA	col1,col10,col19

参数	说明	取值范围	默认值	样例
Columns Wthout Sum	可选，排除指定列的sum校验。	NA	NA	col1,col10,col19

6.5 使用 DataCheck

注意事项

- 启动DataCheck前，必须配置config文件夹中dbinfo.properties文件和check_input.xlsx文件。参数配置错误会导致DataCheck执行错误。
- 如果在同一台服务器上并发进行DataCheck（由同一个或不同DataCheck执行），不同的DataCheck任务必须使用不同的check_input.xlsx文件。
- 用户在执行完DataCheck后会生成logs文件夹，可以进入logs目录查看工具执行过程中的日志，方便定位问题。

基于 Linux 使用 DataCheck

步骤1 上传工具包到Linux服务器并解压：

```
[root@dws-testng-nodelete DataCheck]# ll
total 32
drwx------ 2 root root  4096 Nov 12 20:47 bin
drwxr-xr-x 2 root root  4096 Nov 12 20:34 conf
drwxr-xr-x 2 root root  4096 Nov 12 20:34 input
drwxr-xr-x 2 root root 20480 Nov 12 20:34 lib
```

步骤2 生成数据库登录密码密文：

进入bin目录：

```
[root@dws-testng-nodelete DataCheck]# cd bin
[root@dws-testng-nodelete bin]#
[root@dws-testng-nodelete bin]#
[root@dws-testng-nodelete bin]# ll
total 16
-rwxr-xr-x 1 root root 1254 Oct 19 14:58 datacheck.bat
-rwxr-xr-x 1 root root  410 Oct 19 14:58 datacheck.sh
-rwxr-xr-x 1 root root 1322 Oct 19 14:58 encryption.bat
-rwxr-xr-x 1 root root  385 Oct 19 14:58 encryption.sh
```

执行密文生成的脚本，密文会输出。对源端和目标端数据库登录密码分别执行此脚本生成密文。

sh encryption.sh [password]

```
[root@dws-testng-nodelete bin]# sh encryption.sh  12[redacted]
B06456948116E8E77222B6C4EE5F7CF7A122E357C[redacted]
[root@dws-testng-nodelete bin]#
```

步骤3 配置conf/dbinfo.properties文件：

进入Datacheck目录下，执行vi conf/dbinfo.properties

```
[root@dws-testng-nodelete DataCheck]# cd /opt/temp/DataCheck
[root@dws-testng-nodelete DataCheck]# vi conf/dbinfo.properties
[root@dws-testng-nodelete DataCheck]#
```

配置源端和目标端的数据库连接信息，配置文件中的密码，使用上一步生成的密文。

```
[Source Database Info]
## src.dbtype support: mysql/pg/oracle/dws_src/
src.dbtype = mysql
src.dbname = db_test
src.ip = localhost
src.port = 3306
src.username = admin
src.passwd = 745675379[redacted]

src.projectid =
src.oauthtype = 0
src.oauthserviceacctemail =
src.oauthpvtkeypath =

src.jar.path =
input.file.path =

[DWS Database Info]
dws.dbtype = dws
dws.dbname = db_dws
dws.ip = 100[redacted]
dws.port = 8000
dws.username = dbadmin[redacted]
dws.passwd = 424312BAB[redacted]
```

步骤4 编辑input/check_input.xlsx文件：

复制check_input.xlsx文件到windows服务器，使用Excel软件编辑，填写要校验的表信息，保存后，上传到Linux服务器覆盖原始的文件。

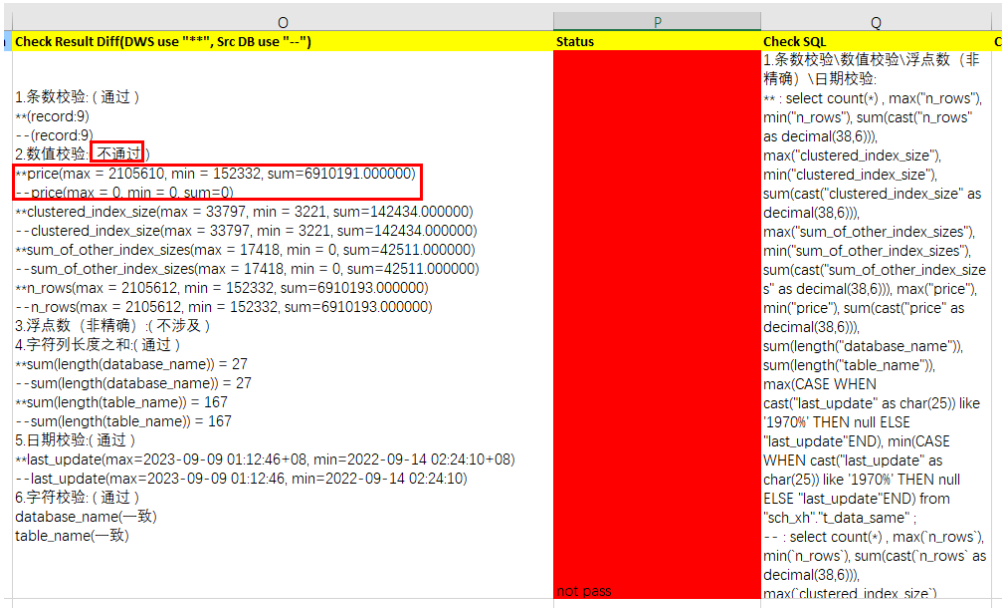
Source Database Name	Source Schema Name	Source Table Name	Target Database Name	Target Schema Name	Target Table Name	* Check Mode	Check Strategy
		t_data_20		sch_test	t_data_21	Statistics	high

步骤5 执行数据校验工具：

进入bin目录，执行启动脚本sh datacheck.sh

```
[root@dws-testng-nodelete DataCheck]# cd /opt/temp/DataCheck/bin/
[root@dws-testng-nodelete bin]# sh datacheck.sh
```

步骤6 查看校验结果 output/check_input_result.xlsx：



O	P	Q
Check Result Diff(DWS use "**", Src DB use "--")	Status	Check SQL
1.条数校验:(通过) **(record:9) --(record:9) 2.数值校验:(不通过) **price(max = 2105610, min = 152332, sum=6910191.000000) --price(max = 0, min = 0, sum=0) **clustered_index_size(max = 33797, min = 3221, sum=142434.000000) --clustered_index_size(max = 33797, min = 3221, sum=142434.000000) **sum_of_other_index_sizes(max = 17418, min = 0, sum=42511.000000) --sum_of_other_index_sizes(max = 17418, min = 0, sum=42511.000000) **n_rows(max = 2105612, min = 152332, sum=6910193.000000) --n_rows(max = 2105612, min = 152332, sum=6910193.000000) 3.浮点数（非精确）:(不涉及) 4.字符列长度之和:(通过) **sum(length(database_name)) = 27 --sum(length(database_name)) = 27 **sum(length(table_name)) = 167 --sum(length(table_name)) = 167 5.日期校验:(通过) **last_update(max=2023-09-09 01:12:46+08, min=2022-09-14 02:24:10+08) --last_update(max=2023-09-09 01:12:46, min=2022-09-14 02:24:10) 6.字符校验:(通过) database_name(一致) table_name(一致)	not pass	1.条数校验\数值校验\浮点数（非精确）\日期校验: ** : select count(*) , max("n_rows"), min("n_rows"), sum(cast("n_rows" as decimal(38,6))), max("clustered_index_size"), min("clustered_index_size"), sum(cast("clustered_index_size" as decimal(38,6))), max("sum_of_other_index_sizes"), min("sum_of_other_index_sizes"), sum(cast("sum_of_other_index_sizes" as decimal(38,6))), max("price"), min("price"), sum(cast("price" as decimal(38,6))), sum(length("database_name")), sum(length("table_name")), max(CASE WHEN cast("last_update" as char(25)) like '1970%' THEN null ELSE "last_update"END), min(CASE WHEN cast("last_update" as char(25)) like '1970%' THEN null ELSE "last_update"END) from "sch_xh"."t_data_same" ; -- : select count(*) , max(`n_rows`), min(`n_rows`), sum(cast(`n_rows` as decimal(38,6))), max(`clustered_index_size`)

步骤7 校验结果分析：

1. Status结果为No Pass代表校验未通过。
2. Check Result Diff列显示校验不通过的项，可在里面查看具体哪一列的校验不通过。
3. Check SQL中显示在数据库中执行的查询SQL。

----结束

基于 Windows 使用 DataCheck

步骤1 上传工具包到Windows服务器并解压：

bin	2024/11/13 10:31
conf	2024/11/13 9:42
input	2024/11/13 9:42
lib	2024/11/13 9:42

步骤2 生成数据库登录密码密文：

进入bin目录，启动CMD工具：

datacheck.bat	2024/10/19 14:58
datacheck.sh	2024/10/19 14:58
encryption.bat	2024/10/19 14:58
encryption.sh	2024/10/19 14:58

执行密文生成的脚本，密文会输出。对源端和目标端数据库登录密码分别执行此脚本生成密文。

encryption.bat [password]


```
D:\temp\DataCheck\bin>encryption.bat 1[redacted]
7EE79288E4BAFA94808D5540794CB1004491[redacted]

D:\temp\DataCheck\bin>encryption.bat a[redacted]
416A0EB307C290131A1C3D2AC77D36AC6D[redacted]
```

步骤3 配置conf/dbinfo.properties文件：

编辑conf目录下的dbinfo.properties文件，配置源端和目标端的数据库连接信息，配置文件中的密码，使用上一步生成的密文。

```
[Source Database Info]
## src.dbtype support: mysql/pg/oracle/dws_src
src.dbtype = mysql
src.dbname = mysql
src.ip = localhost
src.port = 3306
src.username = ***
src.passwd = *****

src.jar.path =
input.file.path =

[DWS Database Info]
## dws.dbtype support: dws
dws.dbtype = dws
dws.dbname = dbname
dws.ip = localhost
dws.port = 8000
dws.username = ***
dws.passwd = *****
```

步骤4 编辑input/check_input.xlsx文件并保存：

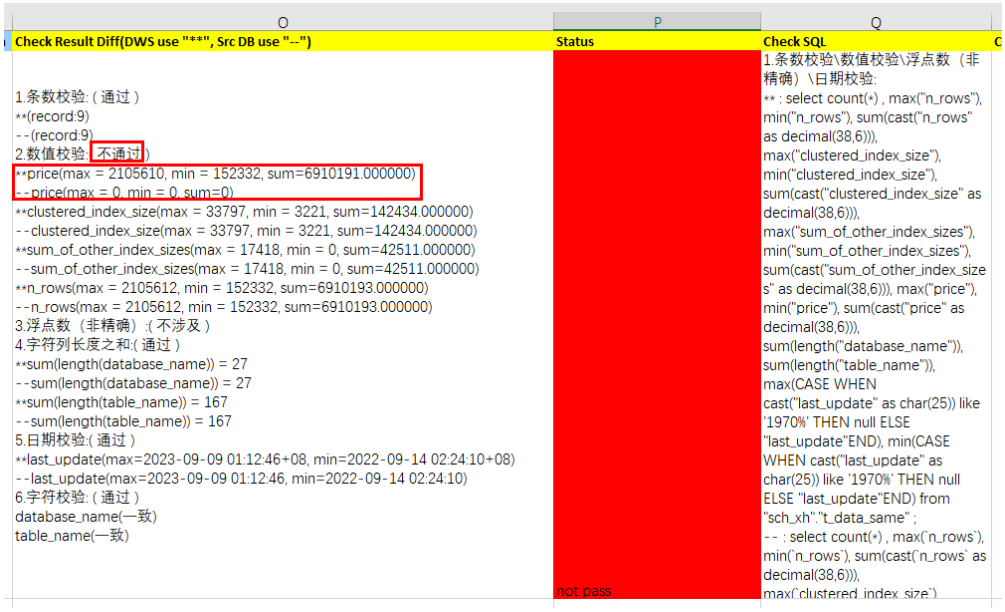
使用Excel软件编辑input/check_input.xlsx，填写要校验的表信息并保存。

Source Database Name	Source Schema Name	Source Table Name	Target Database Name	Target Schema Name	Target Table Name	* Check Mode	Check Strategy
		t_data_20		sch_test	t_data_21	Statistics	high

步骤5 执行数据校验工具 datacheck.bat：

```
D:\temp\DataCheck\bin>datacheck.bat
```

步骤6 查看校验结果 output/check_input_result.xlsx(校验结果分析同Linux场景)：



O	P	Q
Check Result Diff(DWS use "**", Src DB use "--")	Status	Check SQL
1.条数校验:(通过) **(record:9) --(record:9) 2.数值校验:(不通过) **price(max = 2105610, min = 152332, sum=6910191.000000) --price(max = 0, min = 0, sum=0) **clustered_index_size(max = 33797, min = 3221, sum=142434.000000) --clustered_index_size(max = 33797, min = 3221, sum=142434.000000) **sum_of_other_index_sizes(max = 17418, min = 0, sum=42511.000000) --sum_of_other_index_sizes(max = 17418, min = 0, sum=42511.000000) **n_rows(max = 2105612, min = 152332, sum=6910193.000000) --n_rows(max = 2105612, min = 152332, sum=6910193.000000) 3.浮点数（非精确）:(不涉及) 4.字符列长度之和:(通过) **sum(length(database_name)) = 27 --sum(length(database_name)) = 27 **sum(length(table_name)) = 167 --sum(length(table_name)) = 167 5.日期校验:(通过) **last_update(max=2023-09-09 01:12:46+08, min=2022-09-14 02:24:10+08) --last_update(max=2023-09-09 01:12:46, min=2022-09-14 02:24:10) 6.字符校验:(通过) database_name(一致) table_name(一致)	not pass	1.条数校验\数值校验\浮点数（非精确）\日期校验: ** : select count(*) , max("n_rows"), min("n_rows"), sum(cast("n_rows" as decimal(38,6))), max("clustered_index_size"), min("clustered_index_size"), sum(cast("clustered_index_size" as decimal(38,6))), max("sum_of_other_index_sizes"), min("sum_of_other_index_sizes"), sum(cast("sum_of_other_index_sizes" as decimal(38,6))), max("price"), min("price"), sum(cast("price" as decimal(38,6))), sum(length("database_name")), sum(length("table_name")), max(CASE WHEN cast("last_update" as char(25)) like '1970%' THEN null ELSE "last_update"END), min(CASE WHEN cast("last_update" as char(25)) like '1970%' THEN null ELSE "last_update"END) from "sch_xh"."t_data_same" ; -- : select count(*) , max(`n_rows`), min(`n_rows`), sum(cast(`n_rows` as decimal(38,6))), max(`clustered_index_size`)

步骤7 校验结果分析：

1. Status结果为No Pass代表校验未通过。
2. Check Result Diff列显示校验不通过的项，可在里面查看具体哪一列的校验不通过。
3. Check SQL中显示在数据库中执行的查询SQL。

----结束

7 服务端工具

7.1 gs_dump

背景信息

gs_dump是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，导出的表会被加锁，会导致阻塞读写。

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
纯文本格式	p	纯文本脚本文件包含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。	小型数据库，一般推荐纯文本格式。	使用gsql工具恢复数据库对象前，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_shl2.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_shl2.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_shl2.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_shl2.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

  该参数在8.2.0.100版本中已废弃，为兼容历史版本功能保留该函数。

  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。

- --cstore-fine-disaster（已废弃）

  选择此参数时，如果dump一张表级参数fine_disaster_table_role为“primary”的表，会得到一个表级参数fine_disaster_table_role为“standby”的表定义。

  该参数在8.2.1版本废弃。

- --only-publications

  指定该参数时，只dump当前数据库的所有发布（publication）的定义。该参数仅8.2.1及以上集群版本支持。

- --no-comment

在转储中不包含对象注释信息。该参数仅9.1.0.100及以上集群版本支持。

- --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，建议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，导出的表会被加锁，会导致阻塞读写。

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

  该参数在8.2.0.100版本中已废弃，为兼容历史版本功能保留该函数。

  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，建议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。需要注意，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选项一起用于导入某个指定的表。

  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

  该参数在8.2.0.100版本中已废弃，为兼容历史版本功能保留该函数。

  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，用于决定对象归属。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

- -d/--dbname 和 -f/--file 不能同时使用；
- -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；
- 使用--single-transaction时，-j/--jobs必须为单任务；
- --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-9.1.0-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-9.1.0-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，会递归读取location指定的目录层级下所有的同名文件，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
```