数据仓库服务

最佳实践

文档版本 35

发布日期 2024-04-25

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。

目 录

1 导入导出

1.1 导入数据最佳实践

从 OBS 并行导入数据

- 将导入数据拆分为多个文件

  导入大数据量的数据时通常需要较长的时间及耗费较多的计算资源。

  从OBS上导入数据时，如下方法可以提升导入性能：将数据文件存储到OBS前，尽可能均匀地将文件切分成多个，文件的数量为DN的整数倍更适合。
- 在导入前后验证数据文件

  从OBS导入数据时，首先将您的文件上传到OBS存储桶中，我们建议您列出存储桶的内容，然后验证该存储桶是否包含所有正确的文件并且仅包含这些文件。

  在完成导入操作后，请使用SELECT查询语句以验证所需文件是否已导入。
- OBS导入导出数据时，不支持中文路径。

使用 GDS 导入数据

- 数据倾斜会造成查询表性能下降。对于记录数超过千万条的表，建议在执行全量数据导入前，先导入部分数据，以进行数据倾斜检查和调整分布列，避免导入大量数据后发现数据倾斜，调整成本高。详细请参见**查看数据倾斜状态**。
- 为了优化导入速度，建议拆分文件，使用多GDS进行并行导入。另外，单个导入任务可以拆分成多个导入任务并发执行导入，多个导入任务使用同一GDS时可以使用**-t**参数打开GDS多线程并发执行导入。GDS建议挂载在不同物理盘以及不同网卡上，避免物理IO以及网络可能出现的瓶颈。
- 在GDS IO与网卡未达到物理瓶颈前，可以考虑在GaussDB(DWS)开启SMP进行加速。SMP开启之后会对对应的GDS产生成倍的压力。需要特别说明的是：SMP自适应衡量的标准是GaussDB(DWS)的CPU压力，而不是GDS所承受的压力。有关SMP的更多信息请参见**SMP手动调优建议**。
- GDS与GaussDB(DWS)通信要求物理网络畅通，并且尽量使用万兆网。千兆网无法承载高速的数据传输压力，极易出现断连。即使用千兆网时GaussDB(DWS)无法提供通信保障。满足万兆网的同时，数据磁盘组I/O性能大于GDS单核处理能力上限（约400MB/s）时，方可寻求单文件导入速率最大化。
- 并发导入场景，与单表导入相似，至少应保证I/O性能大于网络最大速率。

- GDS跟DN的数据比例建议在1:3至1:6之间。
- 为了优化列存分区表的批量插入效率，在批量插入过程中会对数据进行缓存后再批量写盘。通过GUC参数“**partition_mem_batch**”和“**partition_max_cache_size**”，可以设置缓存个数以及缓存区大小。这两个参数的值越小，列存分区表的批量插入越慢。当然，越大的缓存个数和缓存分区，会带来越多的内存消耗。

使用 INSERT 多行插入

如果不能使用COPY命令，而您需要进行SQL插入，可以根据情况使用多行插入。如果一次只添加一行或几行，则数据压缩效率低下。

多行插入是通过批量进行一系列插入而提高性能。下面的示例使用一条INSERT语句向一个三列表插入三行。这仍属于少量插入，只是用来说明多行插入的语法。创建表的步骤请参考**创建表**。

向表customer_t1中插入多行数据：

```
INSERT INTO customer_t1 VALUES
(6885, 'maps', 'Joes'),
(4321, 'tpcds', 'Lily'),
(9527, 'world', 'James');
```

有关更多详情和示例，请参阅**INSERT**。

使用 COPY 命令导入数据

COPY命令从本地或其它数据库的多个数据源并行导入数据。COPY导入大量数据的效率要比INSERT语句高很多，而且存储数据也更有效率。

有关如何使用COPY命令的更多信息，请参阅**使用COPY FROM STDIN导入数据** 。

使用 gsql 元命令导入数据

\copy命令在任何gsql客户端登录数据库成功后可以执行导入数据。与COPY命令相比较，\copy命令不是读取或写入指定文件的服务器，而是直接读取或写入文件。

这个操作不如SQL COPY命令有效，因为所有的数据必须通过客户端/服务器的连接来传递。对于大量的数据来说SQL命令可能会更好。

有关如何使用\copy命令的更多信息，请参阅**使用gsql元命令\COPY导入数据** 。

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。

1.2 GDS 实践指南

- 安装GDS前必需确认GDS所在服务器环境的系统参数是否和数据库集群的系统参数一致。
- GDS与GaussDB(DWS)通信要求物理网络畅通，尽量使用万兆网。因为千兆网无法承载高速的数据传输压力，极易出现断连，使用千兆网时GaussDB(DWS)无法提供通信保障。满足万兆网的同时，要求数据磁盘组I/O性能大于GDS单核处理能力上限（约400MB/s），才能保证单文件导入速率最大化。

- 提前做好服务部署规划，数据服务器上，建议一个Raid只布1~2个GDS。GDS跟DN的数据比例建议在1:3至1:6之间。一台加载机的GDS进程不宜部署太多，千兆网卡部署1个GDS进程即可，万兆网卡机器建议部署不大于4个进程。
- 提前对GDS导入导出的数据目录做好层次划分，避免一个数据目录包含过多的文件，并及时清理过期文件。
- 合理规划目标数据库的字符集，强烈建议使用UTF8作为数据库的字符集，不建议使用sql_ascii编码，因为极易引起混合编码问题。GDS导出时保证外表的字符集和客户端字符集一致即可，导入时保证客户端编码，数据文件内容编码和客户端一致。
- 如果存在无法变更数据库，客户端，外表字符集时，可以尝试使用**iconv**命令进行手动转换。

```
#注意 -f 表示源文件的字符集，-t为目标字符集
iconv -f utf8 -t gbk utf8.txt -o gbk.txt
```

- 关于GDS导入实践可参考**使用GDS导入数据**。
- GDS支持CSV、TEXT、FIXED三种格式，缺省为TEXT格式。不支持二进制格式，但是可以使用encode/decode函数处理二进制类型。例如：

  对二进制表导出：

```
--创建表。
CREATE TABLE blob_type_t1
(
    BT_COL BYTEA
) DISTRIBUTE BY REPLICATION;
-- 创建外表
CREATE FOREIGN TABLE f_blob_type_t1( BT_COL  text ) SERVER gsmpp_server OPTIONS (LOCATION
'gsfs://127.0.0.1:7789/', FORMAT 'text', DELIMITER E'\x08',  NULL '', EOL '0x0a' ) WRITE ONLY;
INSERT INTO blob_type_t1 VALUES(E'\\xDEADBEEF');
INSERT INTO blob_type_t1 VALUES(E'\\xDEADBEEF');
INSERT INTO blob_type_t1 VALUES(E'\\xDEADBEEF');
INSERT INTO blob_type_t1 VALUES(E'\\xDEADBEEF');
INSERT INTO f_blob_type_t1 select encode(BT_COL,'base64') from blob_type_t1;
```

  对二进制表导入：

```
--创建表。
CREATE TABLE blob_type_t2
(
    BT_COL BYTEA
) DISTRIBUTE BY REPLICATION;
-- 创建外表
CREATE FOREIGN TABLE f_blob_type_t2( BT_COL  text ) SERVER gsmpp_server OPTIONS (LOCATION
'gsfs://127.0.0.1:7789/f_blob_type_t1.dat.0', FORMAT 'text', DELIMITER E'\x08',  NULL '', EOL '0x0a' );
insert into  blob_type_t2 select decode(BT_COL,'base64') from f_blob_type_t2;
SELECT * FROM blob_type_t2;
   bt_col
------------
 \xdeadbeef
 \xdeadbeef
 \xdeadbeef
 \xdeadbeef
(4 rows)
```

- 对同一张外表重复导出会覆盖之前的文件，因此不要对同一个外表重复导出。
- 若不确定文件是否为标准的csv格式，推荐将quote参数设置为0x07，0x08或0x1b等不可见字符来进行GDS导入导出，避免文件格式问题导致任务失败。

```
CREATE FOREIGN TABLE foreign_HR_staffS_ft1
(
  MANAGER_ID     NUMBER(6),
  section_ID     NUMBER(4)
) SERVER gsmpp_server OPTIONS (location 'file:///input_data/*', format 'csv', mode 'private', quote
'0x07', delimiter ',') WITH err_HR_staffS_ft1;
```

- GDS支持并发导入导出，**gds -t**参数用于设置gds的工作线程池大小，控制并发场景下同时工作的工作线程数且不会加速单个sql任务。gds -t缺省值为8，上限值为200。在使用管道功能进行导入导出时，**-t**参数应不低于业务并发数。
- GDS外表参数delimiter是多字符时，建议TEXT格式下字符不要完全相同，例如不建议使用delimiter '---'。
- GDS多表并行导入同一个文件提升导入性能（仅支持text和csv文件）。

```
-- 创建目标表。
CREATE TABLE pipegds_widetb_1 (city integer, tel_num varchar(16), card_code  varchar(15),
phone_code vcreate table pipegds_widetb_3 (city integer, tel_num varchar(16), card_code varchar(15),
phone_code varchar(16), region_code varchar(6), station_id varchar(10), tmsi varchar(20), rec_date
integer(6), rec_time integer(6), rec_type numeric(2), switch_id  varchar(15), attach_city varchar(6),
opc varchar(20), dpc varchar(20));

-- 创建带有file_sequence字段的外表。
CREATE FOREIGN TABLE gds_pip_csv_r_1( like pipegds_widetb_1) SERVER gsmpp_server  OPTIONS
(LOCATION 'gsfs://127.0.0.1:8781/wide_tb.txt', FORMAT 'text', DELIMITER E'|+|', NULL '', file_sequence
'5-1');

CREATE FOREIGN TABLE gds_pip_csv_r_2( like pipegds_widetb_1) SERVER gsmpp_server  OPTIONS
(LOCATION 'gsfs://127.0.0.1:8781/wide_tb.txt', FORMAT 'text', DELIMITER E'|+|', NULL '', file_sequence
'5-2');

CREATE FOREIGN TABLE gds_pip_csv_r_3( like pipegds_widetb_1) SERVER gsmpp_server  OPTIONS
(LOCATION 'gsfs://127.0.0.1:8781/wide_tb.txt', FORMAT 'text', DELIMITER E'|+|', NULL '', file_sequence
'5-3');

CREATE FOREIGN TABLE gds_pip_csv_r_4( like pipegds_widetb_1) SERVER gsmpp_server  OPTIONS
(LOCATION 'gsfs://127.0.0.1:8781/wide_tb.txt', FORMAT 'text', DELIMITER E'|+|', NULL '', file_sequence
'5-4');

CREATE FOREIGN TABLE gds_pip_csv_r_5( like pipegds_widetb_1) SERVER gsmpp_server  OPTIONS
(LOCATION 'gsfs://127.0.0.1:8781/wide_tb.txt', FORMAT 'text', DELIMITER E'|+|', NULL '', file_sequence
'5-5');

--将wide_tb.txt并发导入到pipegds_widetb_1。
\parallel on
INSERT INTO pipegds_widetb_1 SELECT * FROM gds_pip_csv_r_1;
INSERT INTO pipegds_widetb_1 SELECT * FROM gds_pip_csv_r_2;
INSERT INTO pipegds_widetb_1 SELECT * FROM gds_pip_csv_r_3;
INSERT INTO pipegds_widetb_1 SELECT * FROM gds_pip_csv_r_4;
INSERT INTO pipegds_widetb_1 SELECT * FROM gds_pip_csv_r_5;
\parallel off
```

file_sequence参数详细内容，可参考**CREATE FOREIGN TABLE (GDS导入导出)**。

1.3 教程：从 OBS 导入数据到集群

教程指引

本教程通过演示将样例数据上传OBS，以及将OBS的数据导入GaussDB(DWS)的目标表中，让您快速掌握如何从OBS导入数据到GaussDB(DWS)集群的完整过程。

GaussDB(DWS)支持通过外表将OBS上TXT、CSV、ORC、PARQUET、CARBONDATA以及JSON格式的数据导入到集群进行查询。

本教程中以CSV格式为例，进行如下操作：

- 生成CSV格式的数据文件。
- 创建一个与GaussDB(DWS)集群在同一区域的OBS存储桶，然后将数据文件上传到该存储桶。

- 创建外表，用于引流OBS存储桶中的数据到GaussDB(DWS)集群。
- 启动GaussDB(DWS)并创建数据库表后，将OBS上的数据导入到表中。
- 根据错误表中的提示诊断加载错误并更正这些错误。

估计时间：30分钟

准备数据源文件

- 数据文件“product_info0.csv”

```
100,XHDK-A,2017-09-01,A,2017 Shirt Women,red,M,328,2017-09-04,715,good!
205,KDKE-B,2017-09-01,A,2017 T-shirt Women,pink,L,584,2017-09-05,40,very good!
300,JODL-X,2017-09-01,A,2017 T-shirt men,red,XL,15,2017-09-03,502,Bad.
310,QQPX-R,2017-09-02,B,2017 jacket women,red,L,411,2017-09-05,436,It's nice.
150,ABEF-C,2017-09-03,B,2017 Jeans Women,blue,M,123,2017-09-06,120,good.
```

- 数据文件“product_info1.csv”

```
200,BCQP-E,2017-09-04,B,2017 casual pants men,black,L,997,2017-09-10,301,good quality.
250,EABE-D,2017-09-10,A,2017 dress women,black,S,841,2017-09-15,299,This dress fits well.
108,CDXK-F,2017-09-11,A,2017 dress women,red,M,85,2017-09-14,22,It's really amazing to buy.
450,MMCE-H,2017-09-11,A,2017 jacket women,white,M,114,2017-09-14,22,very good.
260,OCDA-G,2017-09-12,B,2017 woolen coat women,red,L,2004,2017-09-15,826,Very comfortable.
```

- 数据文件“product_info2.csv”

```
980,"ZKDS-J",2017-09-13,"B","2017 Women's Cotton Clothing","red","M",112,,,
98,"FKQB-I",2017-09-15,"B","2017 new shoes men","red","M",4345,2017-09-18,5473
50,"DMQY-K",2017-09-21,"A","2017 pants men","red","37",28,2017-09-25,58,"good","good","good"
80,"GKLW-l",2017-09-22,"A","2017 Jeans Men","red","39",58,2017-09-25,72,"Very comfortable."
30,"HWEC-L",2017-09-23,"A","2017 shoes women","red","M",403,2017-09-26,607,"good!"
40,"IQPD-M",2017-09-24,"B","2017 new pants Women","red","M",35,2017-09-27,52,"very good."
50,"LPEC-N",2017-09-25,"B","2017 dress Women","red","M",29,2017-09-28,47,"not good at all."
60,"NQAB-O",2017-09-26,"B","2017 jacket women","red","S",69,2017-09-29,70,"It's beautiful."
70,"HWNB-P",2017-09-27,"B","2017 jacket women","red","L",30,2017-09-30,55,"I like it so much"
80,"JKHU-Q",2017-09-29,"C","2017 T-shirt","red","M",90,2017-10-02,82,"very good."
```

步骤1 新建文本文档并使用本地编辑工具（例如Visual Studio Code）打开后，将示例数据拷贝进文本文档中。

步骤2 选择“格式 > 以UTF-8无BOM格式编码”。

步骤3 选择“文件 > 另存为”。

步骤4 在弹出的对话框中输入文件名后，将文件后缀设为.csv，单击“保存”。

----结束

上传数据到 OBS

步骤1 将上面准备的3个CSV格式的数据源文件存储到OBS桶中。

1. 登录OBS管理控制台。
   单击“服务列表”，选择“对象存储服务”，打开OBS管理控制台页面。
2. 创建桶。

   如何创建OBS桶，具体请参见《对象存储服务》“快速入门”中的**创建桶**章节。
   例如，创建以下两个桶：“mybucket”和“mybucket02”。

须知

确保这两个桶与GaussDB(DWS)集群在同一个区域，本教程以"华北-北京四”区域为例。

3. 新建文件夹。

   具体请参见《对象存储服务控制台指南》中的**新建文件夹**章节。

   例如：

   - 在已创建的OBS桶“mybucket”中新建一个文件夹“input_data”。
   - 在已创建的OBS桶“mybucket02”中新建一个文件夹“input_data”。

4. 上传文件。

   具体请参见《对象存储服务控制台指南》的**上传对象**章节。

   例如：

   - 将以下数据文件上传到OBS桶“mybucket”的“input_data”目录中。

     ```
     product_info0.csv
     product_info1.csv
     ```

   - 将以下数据文件上传到OBS桶“mybucket02”的“input_data”目录中。

     ```
     product_info2.csv
     ```

步骤2 为导入用户设置OBS桶的读取权限。

在从OBS导入数据到集群时，执行导入操作的用户需要取得数据源文件所在OBS桶的读取权限。通过配置桶的ACL权限，可以将读取权限授予指定的用户账号。

具体请参见《对象存储服务控制台指南》中的**配置桶ACL**章节。

----结束

创建外表

步骤1 连接GaussDB(DWS)数据库。

步骤2 创建外表。

说明

- ACCESS_KEY和SECRET_ACCESS_KEY

  用户访问OBS的AK和SK，请根据实际替换。

  获取访问密钥，请登录管理控制台，将鼠标移至右上角的用户名，单击“我的凭证”，然后在左侧导航树单击“访问密钥”。在访问密钥页面，可以查看已有的访问密钥ID（即AK），如果要同时获取AK和SK，可以单击“新增访问密钥”创建并下载访问密钥。

- 认证用的AK和SK硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全。

```
DROP FOREIGN TABLE IF EXISTS product_info_ext;
CREATE FOREIGN TABLE product_info_ext
(
    product_price                integer       not null,
    product_id                   char(30)      not null,
    product_time                 date          ,
    product_level                char(10)      ,
    product_name                 varchar(200)  ,
    product_type1                varchar(20)   ,
    product_type2                char(10)      ,
    product_monthly_sales_cnt    integer       ,
    product_comment_time         date          ,
```

```
    product_comment_num          integer        ,
    product_comment_content      varchar(200)
)
SERVER gsmpp_server
OPTIONS(
LOCATION 'obs://mybucket/input_data/product_info | obs://mybucket02/input_data/product_info',
FORMAT 'CSV' ,
DELIMITER ',',
ENCODING 'utf8',
HEADER 'false',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
FILL_MISSING_FIELDS 'true',
IGNORE_EXTRA_DATA 'true'
)
READ ONLY
LOG INTO product_info_err
PER NODE REJECT LIMIT 'unlimited';
```

返回如下信息表示创建成功：

```
CREATE FOREIGN TABLE
```

----结束

执行数据导入

步骤1 在GaussDB(DWS)数据库中，创建一个名为product_info的表，用于存储从OBS导入的数据。

```
DROP TABLE IF EXISTS product_info;
CREATE TABLE product_info
(
    product_price               integer        not null,
    product_id                  char(30)       not null,
    product_time                date           ,
    product_level               char(10)       ,
    product_name                varchar(200)   ,
    product_type1               varchar(20)    ,
    product_type2               char(10)       ,
    product_monthly_sales_cnt   integer        ,
    product_comment_time        date           ,
    product_comment_num         integer        ,
    product_comment_content     varchar(200)
)
WITH (
orientation = column,
compression=middle
)
DISTRIBUTE BY hash (product_id);
```

步骤2 执行INSERT命令，通过外表product_info_ext将OBS上的数据导入到目标表product_info 中：

```
INSERT INTO product_info SELECT * FROM product_info_ext;
```

步骤3 执行SELECT命令查询目标表product_info，查看从OBS导入到GaussDB(DWS)中的数据。

```
SELECT * FROM product_info;
```

查询结果的结尾将显示以下信息：

```
(20 rows)
```

步骤4 对表product_info执行VACUUM FULL。

```
VACUUM FULL product_info;
```

步骤5 更新表product_info的统计信息。


```
ANALYZE product_info;
```

----结束

清除资源

步骤1 如果执行了导入数据后查询数据，请执行以下命令，删除目标表。

```
DROP TABLE product_info;
```

当结果显示为如下信息，则表示删除成功。

```
DROP TABLE
```

步骤2 执行以下命令，删除外表。

```
DROP FOREIGN TABLE product_info_ext;
```

当结果显示为如下信息，则表示删除成功。

```
DROP FOREIGN TABLE
```

----结束

1.4 教程：使用 GDS 从远端服务器导入数据

教程指引

本教程旨在演示使用GDS（General Data Service）工具将远端服务器上的数据导入GaussDB(DWS)中的办法，帮助您学习如何通过GDS进行数据导入的方法。

GaussDB(DWS)支持通过GDS外表将TXT、CSV和FIXED格式的数据导入到集群进行查询。

在本教程中，您将：

- 生成本教程需要使用的CSV格式的数据源文件。
- 将数据源文件上传到数据服务器。
- 创建外表，用于对接GDS和GaussDB(DWS)，将数据服务器上的数据导入到GaussDB(DWS)集群中。
- 启动GaussDB(DWS)并创建数据库表后，将数据导入到表中。
- 根据错误表中的提示诊断加载错误并更正这些错误。

准备 ECS 作为 GDS 服务器

购买Linux弹性云服务器的操作步骤，请参见《弹性云服务器快速入门》中的**自定义购买弹性云服务器**。购买后，请参见**登录Linux弹性云服务器**进行登录。

说明

- ECS操作系统必须是GDS工具包所支持的操作系统。
- ECS与DWS处于同一区域、同一虚拟私有云和子网。
- ECS安全组规则需放通DWS集群的访问，即安全组入规则：
  - 协议：TCP
  - 端口范围：5000
  - 源地址：选择“IP地址”，输入GaussDB(DWS) 集群地址，例如“192.168.0.10/32”。
- ECS内部如果启用了防火墙，需要保证防火墙打开了GDS服务的监听端口：
  ```
  iptables  -I INPUT -p tcp -m tcp --dport <gds_port> -j ACCEPT
  ```

下载 GDS 工具包

步骤1 登录GaussDB(DWS)管理控制台。

步骤2 在左侧导航栏中，单击“连接管理”。

步骤3 在“命令行客户端”的下拉列表中，选择对应版本的GDS客户端。

请根据集群版本和安装客户端的操作系统，选择对应版本。

说明

客户端CPU架构要和集群一致，如果集群是X86规格，则也应该选择X86客户端。

步骤4 单击“下载”。

----结束

准备数据源文件

- 数据文件“product_info0.csv”
  ```
  100,XHDK-A,2017-09-01,A,2017 Shirt Women,red,M,328,2017-09-04,715,good!
  205,KDKE-B,2017-09-01,A,2017 T-shirt Women,pink,L,584,2017-09-05,40,very good!
  300,JODL-X,2017-09-01,A,2017 T-shirt men,red,XL,15,2017-09-03,502,Bad.
  310,QQPX-R,2017-09-02,B,2017 jacket women,red,L,411,2017-09-05,436,It's nice.
  150,ABEF-C,2017-09-03,B,2017 Jeans Women,blue,M,123,2017-09-06,120,good.
  ```
- 数据文件“product_info1.csv”
  ```
  200,BCQP-E,2017-09-04,B,2017 casual pants men,black,L,997,2017-09-10,301,good quality.
  250,EABE-D,2017-09-10,A,2017 dress women,black,S,841,2017-09-15,299,This dress fits well.
  108,CDXK-F,2017-09-11,A,2017 dress women,red,M,85,2017-09-14,22,It's really amazing to buy.
  450,MMCE-H,2017-09-11,A,2017 jacket women,white,M,114,2017-09-14,22,very good.
  260,OCDA-G,2017-09-12,B,2017 woolen coat women,red,L,2004,2017-09-15,826,Very comfortable.
  ```
- 数据文件“product_info2.csv”
  ```
  980,"ZKDS-J",2017-09-13,"B","2017 Women's Cotton Clothing","red","M",112,,,
  98,"FKQB-I",2017-09-15,"B","2017 new shoes men","red","M",4345,2017-09-18,5473
  50,"DMQY-K",2017-09-21,"A","2017 pants men","red","37",28,2017-09-25,58,"good","good","good"
  80,"GKLW-l",2017-09-22,"A","2017 Jeans Men","red","39",58,2017-09-25,72,"Very comfortable."
  30,"HWEC-L",2017-09-23,"A","2017 shoes women","red","M",403,2017-09-26,607,"good!"
  40,"IQPD-M",2017-09-24,"B","2017 new pants Women","red","M",35,2017-09-27,52,"very good."
  50,"LPEC-N",2017-09-25,"B","2017 dress Women","red","M",29,2017-09-28,47,"not good at all."
  60,"NQAB-O",2017-09-26,"B","2017 jacket women","red","S",69,2017-09-29,70,"It's beautiful."
  70,"HWNB-P",2017-09-27,"B","2017 jacket women","red","L",30,2017-09-30,55,"I like it so much"
  80,"JKHU-Q",2017-09-29,"C","2017 T-shirt","red","M",90,2017-10-02,82,"very good."
  ```

步骤1 新建文本文档并使用本地编辑工具（例如Visual Studio Code）打开后，将示例数据拷贝进文本文档中。

步骤2 选择“格式 > 以UTF-8无BOM格式编码”。

步骤3 选择“文件 > 另存为”。

步骤4 在弹出的对话框中输入文件名后，将文件后缀设为.csv，单击“保存”。

步骤5 以root用户登录GDS服务器。

步骤6 创建数据文件存放目录“/input_data”。

```
mkdir -p /input_data
```

步骤7 使用MobaXterm将数据源文件上传至上一步所创建的目录中。

----结束

安装并启动 GDS

步骤1 以root用户登录GDS服务器，创建存放GDS工具包的目录/opt/bin/dws。

```
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redhat版本的工具包为例，将GDS工具包“dws_client_8.1.x_redhat_x64.zip”上传至上一步所创建的目录中。

步骤3 在工具包所在目录下，解压工具包。

```
cd /opt/bin/dws
unzip dws_client_8.1.x_redhat_x64.zip
```

步骤4 创建用户gds_user及其所属的用户组gdsgrp。此用户用于启动GDS，且需要拥有读取数据源文件目录的权限。

```
groupadd gdsgrp
useradd -g gdsgrp gds_user
```

步骤5 修改工具包以及数据源文件目录属主为创建的用户gds_user及其所属的用户组gdsgrp。

```
chown -R gds_user:gdsgrp /opt/bin/dws/gds
chown -R gds_user:gdsgrp /input_data
```

步骤6 切换到gds_user用户。

```
su - gds_user
```

若当前集群版本为8.0.x及之前低版本，请跳过**步骤7**，直接执行**步骤8**。

若当前集群版本为8.1.x及以上版本，则正常执行以下步骤。

步骤7 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8 启动GDS。

```
/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
```

命令中的部分信息请根据实际填写。

- **-d** *dir*：保存有待导入数据的数据文件所在目录。本教程中为“/input_data/”。
- **-p** *ip:port*：GDS监听IP和监听端口。默认值为：127.0.0.1，需要替换为能跟GaussDB(DWS)通信的万兆网IP。监听端口的取值范围：1024~65535。默认值为：8098。本教程配置为：192.168.0.90:5000。

- **-H** *address_string*：允许哪些主机连接和使用GDS服务。参数需为CIDR格式。此参数配置的目的是允许GaussDB(DWS)集群可以访问GDS服务进行数据导入。所以请保证所配置的网段包含GaussDB(DWS)集群各主机。
- **-l** *log_file*：存放GDS的日志文件路径及文件名。本教程为“/opt/bin/dws/gds/gds_log.txt”。
- **-D**：后台运行GDS。仅支持Linux操作系统下使用。

----结束

创建外表

步骤1 使用SQL客户端工具连接GaussDB(DWS)数据库。

步骤2 创建如下外表：

⚠ 注意

LOCATION：请替换成实际的GDS地址和端口。

```
DROP FOREIGN TABLE IF EXISTS product_info_ext;
CREATE FOREIGN TABLE product_info_ext
(
    product_price                integer        not null,
    product_id                   char(30)       not null,
    product_time                 date           ,
    product_level                char(10)       ,
    product_name                 varchar(200)   ,
    product_type1                varchar(20)    ,
    product_type2                char(10)       ,
    product_monthly_sales_cnt    integer        ,
    product_comment_time         date           ,
    product_comment_num          integer        ,
    product_comment_content      varchar(200)
)
SERVER gsmpp_server
OPTIONS(
LOCATION 'gsfs://192.168.0.90:5000/*',
FORMAT 'CSV' ,
DELIMITER ',',
ENCODING 'utf8',
HEADER 'false',
FILL_MISSING_FIELDS 'true',
IGNORE_EXTRA_DATA 'true'
)
READ ONLY
LOG INTO product_info_err
PER NODE REJECT LIMIT 'unlimited';
```

返回如下信息表示创建成功：

```
CREATE FOREIGN TABLE
```

----结束

导入数据

步骤1 使用如下语句在GaussDB(DWS)中创建目标表product_info，用于存储导入的数据。

```
DROP TABLE IF EXISTS product_info;
CREATE TABLE product_info
```



```
(
    product_price               integer        not null,
    product_id                  char(30)       not null,
    product_time                date           ,
    product_level               char(10)       ,
    product_name                varchar(200)   ,
    product_type1               varchar(20)    ,
    product_type2               char(10)       ,
    product_monthly_sales_cnt   integer        ,
    product_comment_time        date           ,
    product_comment_num         integer        ,
    product_comment_content     varchar(200)
)
WITH (
orientation = column,
compression=middle
)
DISTRIBUTE BY hash (product_id);
```

步骤2 将数据源文件中的数据通过外表“product_info_ext”导入到表“product_info”中。

```
INSERT INTO product_info SELECT * FROM product_info_ext ;
```

出现以下信息，说明数据导入成功。

```
INSERT 0 20
```

步骤3 执行SELECT命令查询目标表product_info，查看导入到GaussDB(DWS)中的数据。

```
SELECT count(*) FROM product_info;
```

查询结果显示结果如下，表示导入成功。

```
count
-------
    20
(1 row)
```

步骤4 对表product_info执行VACUUM FULL。

```
VACUUM FULL product_info
```

步骤5 更新表product_info的统计信息。

```
ANALYZE product_info;
```

----结束

停止 GDS

步骤1 以gds_user用户登录安装GDS的数据服务器。

步骤2 请使用以下方式停止GDS。

1. 执行如下命令，查询GDS进程号。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 /opt/bin/gds/gds_log.txt  -D
   gds_user 129003 118723  0 15:04 pts/0    00:00:00 grep gds
   ```

2. 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

清除资源

步骤1 执行以下命令，删除目标表product_info。

```
DROP TABLE product_info;
```

当结果显示为如下信息，则表示删除成功。

```
DROP TABLE
```

步骤2 执行以下命令，删除外表product_info_ext。

```
DROP FOREIGN TABLE product_info_ext;
```

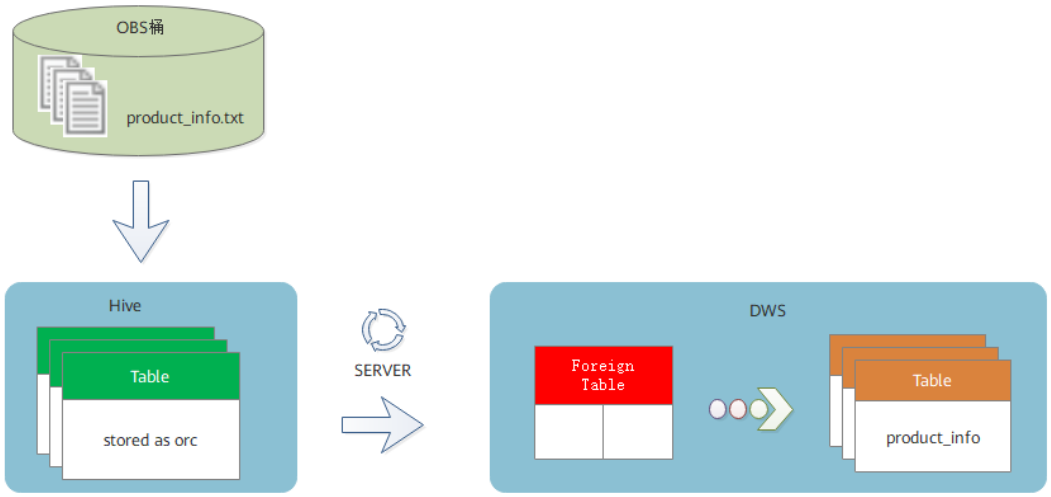
当结果显示为如下信息，则表示删除成功。

```
DROP FOREIGN TABLE
```

----结束

1.5 教程：查看或导入 MRS 的 hive 数据

本教程通过建立HDFS外表实现GaussDB(DWS)远端访问或读取MRS数据源。



准备环境

已创建DWS集群，需确保MRS和DWS集群在同一个区域、可用区、同一VPC子网内，确保集群网络互通。

基本流程

本实践预计时长：1小时，基本流程如下：

1. 创建MRS分析集群（选择Hive、Spark、Tez组件）。
2. 通过将本地txt数据文件上传至OBS桶，再通过OBS桶导入Hive，并由txt存储表导入ORC存储表。
3. 创建MRS数据源连接。
4. 创建外部服务器。
5. 创建外表。
6. 通过外表导入DWS本地表。

创建 MRS 分析集群

步骤1 登录华为云控制台，选择“大数据 > MapReduce服务”，单击“购买集群”，选择“自定义购买”，填写软件配置参数，单击“下一步”。

表 1-1 软件配置

参数项	取值
区域	华北-北京四
集群名称	mrs_01
版本类型	普通版
集群版本	**MRS 1.9.2**（主推） **说明** • 8.1.1.300及以上版本集群，MRS集群支持连接1.6.*、*1.7.**、1.8.*、*1.9.**、*2.0.**、3.0.*、3.1.*及以上版本（“*”代表的是数字）。 • 8.1.1.300以下版本集群，MRS集群支持连接1.6.*、*1.7.**、1.8.*、*1.9.**、*2.0.**版本（“*”代表的是数字）。
集群类型	分析集群
元数据	本地元数据

步骤2 填写硬件配置参数，单击“下一步”。

表 1-2 硬件配置

参数项	取值
计费模式	按需计费
可用区	可用区2
虚拟私有云	vpc-01
子网	subnet-01
安全组	自动创建
弹性公网IP	*10.x.x.x*
企业项目	default
Master节点	2
分析Core节点	3
分析Task节点	0

步骤3 填写高级配置参数，单击“立即购买”，等待约15分钟，集群创建成功。

表 1-3 高级配置

参数项	取值
标签	test01

参数项	取值
主机名前缀	可不填写，用作集群中ECS机器或BMS机器主机名的前缀。
弹性伸缩	保持默认即可
引导操作	保持默认即可，MRS 3.x版本暂时不支持该参数。
委托	保持默认即可
数据盘加密	默认关闭，保持默认即可。
告警	保持默认即可
规则名称	保持默认即可
主题名称	选择相应的主题
Kerberos认证	默认打开
用户名	admin
密码	设置密码，该密码用于登录集群管理页面。
确认密码	再次输入设置admin用户密码
登录方式	密码
用户名	root
密码	设置密码，该密码用于远程登录ECS机器。
确认密码	再次输入设置的root用户密码
通信安全授权	勾选“确认授权”

----结束

准备 MRS 的 ORC 表数据源

步骤1 本地PC新建一个product_info.txt，并拷贝以下数据，保存到本地。

```
100,XHDK-A-1293-#fJ3,2017-09-01,A,2017 Autumn New Shirt Women,red,M,328,2017-09-04,715,good
205,KDKE-B-9947-#kL5,2017-09-01,A,2017 Autumn New Knitwear Women,pink,L,584,2017-09-05,406,very good!
300,JODL-X-1937-#pV7,2017-09-01,A,2017 autumn new T-shirt men,red,XL,1245,2017-09-03,502,Bad.
310,QQPX-R-3956-#aD8,2017-09-02,B,2017 autumn new jacket women,red,L,411,2017-09-05,436,It's really super nice
150,ABEF-C-1820-#mC6,2017-09-03,B,2017 Autumn New Jeans Women,blue,M,1223,2017-09-06,1200,The seller's packaging is exquisite
200,BCQP-E-2365-#qE4,2017-09-04,B,2017 autumn new casual pants men,black,L,997,2017-09-10,301,The clothes are of good quality.
250,EABE-D-1476-#oB1,2017-09-10,A,2017 autumn new dress women,black,S,841,2017-09-15,299,Follow the store for a long time.
108,CDXK-F-1527-#pL2,2017-09-11,A,2017 autumn new dress women,red,M,85,2017-09-14,22,It's really amazing to buy
450,MMCE-H-4728-#nP9,2017-09-11,A,2017 autumn new jacket women,white,M,114,2017-09-14,22,Open the package and the clothes have no odor
260,OCDA-G-2817-#bD3,2017-09-12,B,2017 autumn new woolen coat women,red,L,2004,2017-09-15,826,Very favorite clothes
980,ZKDS-J-5490-#cW4,2017-09-13,B,2017 Autumn New Women's Cotton Clothing,red,M,112,2017-09-16,219,The clothes are small
```

```
98,FKQB-I-2564-#dA5,2017-09-15,B,2017 autumn new shoes men,green,M,4345,2017-09-18,5473,The clothes are thick and it's better this winter.
150,DMQY-K-6579-#eS6,2017-09-21,A,2017 autumn new underwear men,yellow,37,2840,2017-09-25,5831,This price is very cost effective
200,GKLW-l-2897-#wQ7,2017-09-22,A,2017 Autumn New Jeans Men,blue,39,5879,2017-09-25,7200,The clothes are very comfortable to wear
300,HWEC-L-2531-#xP8,2017-09-23,A,2017 autumn new shoes women,brown,M,403,2017-09-26,607,good
100,IQPD-M-3214-#yQ1,2017-09-24,B,2017 Autumn New Wide Leg Pants Women,black,M,3045,2017-09-27,5021,very good.
350,LPEC-N-4572-#zX2,2017-09-25,B,2017 Autumn New Underwear Women,red,M,239,2017-09-28,407,The seller's service is very good
110,NQAB-O-3768-#sM3,2017-09-26,B,2017 autumn new underwear women,red,S,6089,2017-09-29,7021,The color is very good
210,HWNB-P-7879-#tN4,2017-09-27,B,2017 autumn new underwear women,red,L,3201,2017-09-30,4059,I like it very much and the quality is good.
230,JKHU-Q-8865-#uO5,2017-09-29,C,2017 Autumn New Clothes with Chiffon Shirt,black,M,2056,2017-10-02,3842,very good
```

步骤2 登录OBS控制台，单击“创建桶”，填写以下参数，单击“立即创建”。

表 1-4 桶参数

参数项	取值
区域	华北-北京四
数据冗余存储策略	单AZ存储
桶名称	mrs-datasource
默认存储类别	标准存储
桶策略	私有
默认加密	关闭
归档数据直读	关闭
企业项目	default
标签	-

步骤3 等待桶创建好，单击桶名称，选择“对象 > 上传对象”，将product_info.txt上传至OBS桶。

步骤4 切换回MRS控制台，单击创建好的MRS集群名称，进入“概览”，单击“IAM用户同步”所在行的“同步”，等待约5分钟同步完成。

步骤5 单击“节点管理”，单击任意一台master节点，进入该节点页面，切换到“弹性公网IP”，单击“绑定弹性公网IP”，勾选已有弹性IP并单击“确定”，如果没有，请创建。记录此公网IP。

步骤6 下载客户端。

1. 回到MRS集群页面，单击集群名称进入“概览”，单击“前往Manager”，如果提示绑定公网IP，请先绑定公网IP。
2. 如果弹出访问MRS Manager对话框，单击“确定”，页面会跳转到MRS登录页面。输入MRS Manager的用户名admin和密码，密码为创建MRS集群时输入的admin密码。
3. 选择“集群 > *待操作集群的名称* > 概览 > 更多 > 下载客户端”。界面显示“下载集群客户端”对话框。

说明

历史版本的客户端获取方法：请选择“服务管理 > 下载客户端”，“客户端类型”选择“仅配置文件”。

步骤7 确认主master节点。

1. 使用SSH工具以root用户登录以上节点，切换到omm用户。

   su - omm

2. 执行以下命令查询主master节点，回显信息中“HAActive”参数值为“active”的节点为主master节点。

   sh ${BIGDATA_HOME}/om-0.0.1/sbin/status-oms.sh

步骤8 使用root用户登录主master节点，并更新主管理节点的客户端配置。

cd /opt/client

sh refreshConfig.sh /opt/client *客户端配置文件压缩包完整路径*

本例命令为：

sh refreshConfig.sh /opt/client /tmp/MRS-client/MRS_Services_Client.tar

步骤9 切换到omm用户，并进入Hive客户端所在目录。

su - omm

cd /opt/client

步骤10 在Hive上创建存储类型为TEXTFILE的表product_info。

1. 在/opt/client路径下，导入环境变量。

   source bigdata_env

2. 登录Hive客户端。

   beeline

3. 依次执行以下SQL语句创建demo数据库及表product_info。

```
CREATE DATABASE demo;
USE demo;
DROP TABLE product_info;
```

```
CREATE TABLE product_info
(
    product_price               int           ,
    product_id                  char(30)      ,
    product_time                date          ,
    product_level               char(10)      ,
    product_name                varchar(200)  ,
    product_type1               varchar(20)   ,
    product_type2               char(10)      ,
    product_monthly_sales_cnt   int           ,
    product_comment_time        date          ,
    product_comment_num         int           ,
    product_comment_content     varchar(200)
)
row format delimited fields terminated by ','
stored as TEXTFILE;
```

步骤11 将product_info.txt数据文件导入Hive。

1. 切回到MRS集群，单击“文件管理”，单击“导入数据”。
2. OBS路径：选择上面创建好的OBS桶名，找到product_info.txt文件，单击“是”。
3. HDFS路径：选择/user/hive/warehouse/demo.db/product_info/，单击“是”。
4. 单击“确定”，等待导入成功，此时product_info的表数据已导入成功。

步骤12 创建ORC表，并将数据导入ORC表。

1. 执行以下SQL语句创建ORC表。

```
DROP TABLE product_info_orc;

CREATE TABLE product_info_orc
(
    product_price               int           ,
    product_id                  char(30)      ,
    product_time                date          ,
    product_level               char(10)      ,
    product_name                varchar(200)  ,
    product_type1               varchar(20)   ,
    product_type2               char(10)      ,
    product_monthly_sales_cnt   int           ,
    product_comment_time        date          ,
    product_comment_num         int           ,
    product_comment_content     varchar(200)
)
row format delimited fields terminated by ','
stored as orc;
```

2. 将product_info表的数据插入到Hive ORC表product_info_orc中。

```
insert into product_info_orc select * from product_info;
```

3. 查询ORC表数据导入成功。

```
select * from product_info_orc;
```

----结束

创建 MRS 数据源连接

步骤1 登录DWS管理控制台，单击已创建好的DWS集群，**确保DWS集群与MRS在同一个区域、可用分区，并且在同一VPC子网下。**

步骤2 切换到“MRS数据源”，单击“创建MRS数据源连接”。

步骤3 选择前序步骤创建名为的“mrs_01”数据源，用户名：admin，密码：用户自定义，单击“确定”，创建成功。

----结束

创建外部服务器

步骤1 使用Data Studio连接已创建好的DWS集群。

步骤2 新建一个具有创建数据库权限的用户dbuser：

```
CREATE USER dbuser WITH CREATEDB PASSWORD 'password';
```

步骤3 切换为新建的dbuser用户：

```
SET ROLE dbuser PASSWORD 'password';
```

步骤4 创建新的mydatabase数据库：

```
CREATE DATABASE mydatabase;
```

步骤5 执行以下步骤切换为连接新建的mydatabase数据库。

1. 在Data Studio客户端的**“对象浏览器”**窗口，右键单击数据库连接名称，在弹出菜单中单击“刷新”，刷新后就可以看到新建的数据库。
2. 右键单击“mydatabase”数据库名称，在弹出菜单中单击“打开连接”。
3. 右键单击“mydatabase”数据库名称，在弹出菜单中单击“打开新的终端”，即可打开连接到指定数据库的SQL命令窗口，后面的步骤，请全部在该命令窗口中执行。

步骤6 为dbuser用户授予创建外部服务器的权限，8.1.1及以后版本，还需要授予使用public模式的权限：

```
GRANT ALL ON FOREIGN DATA WRAPPER hdfs_fdw TO dbuser;
GRANT ALL ON SCHEMA public TO dbuser;  //8.1.1及以后版本，普通用户对public模式无权限，需要赋权，8.1.1之前版本不需要执行。
```

其中FOREIGN DATA WRAPPER的名字只能是hdfs_fdw，dbuser为创建SERVER的用户名。

步骤7 执行以下命令赋予用户使用外表的权限。

```
ALTER USER dbuser USEFT;
```

步骤8 切换回Postgres系统数据库，查询创建MRS数据源后系统自动创建的外部服务器。

```
SELECT * FROM pg_foreign_server;
```

返回结果如：

```
                srvname                 | srvowner | srvfdw | srvtype | srvversion | srvacl
|                                       srvoptions
------------------------------------------------+----------+--------+---------+------------+--------
+-----------------------------------------------------------------------------------------------------------------------
 gsmpp_server                                   |       10 |  13673 |         |            |
 gsmpp_errorinfo_server                         |       10 |  13678 |         |            |
 hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca |    16476 |  13685 |         |            |
{"address=192.168.1.245:9820,192.168.1.218:9820",hdfscfgpath=/MRS/8f79ada0-
d998-4026-9020-80d6de2692ca,type=hdfs}
(3 rows)
```

步骤9 切换到mydatabase数据库，并切换到dbuser用户。

```
SET ROLE dbuser PASSWORD 'password';
```

步骤10 创建外部服务器。

SERVER名字、地址、配置路径保持与**步骤8**一致即可。

```
CREATE SERVER hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca FOREIGN DATA WRAPPER
HDFS_FDW
OPTIONS
(
address '192.168.1.245:9820,192.168.1.218:9820',    //MRS管理面的Master主备节点的内网IP，可与DWS通讯。
hdfscfgpath '/MRS/8f79ada0-d998-4026-9020-80d6de2692ca',
type 'hdfs'
);
```

步骤11 查看外部服务器。

```
SELECT * FROM pg_foreign_server WHERE
srvname='hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca';
```

返回结果如下所示，表示已经创建成功：

```
                srvname                 | srvowner | srvfdw | srvtype | srvversion | srvacl
|                                       srvoptions
------------------------------------------------+----------+--------+---------+------------+--------
+-----------------------------------------------------------------------------------------------------------------------
 hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca |    16476 |  13685 |         |            |
{"address=192.168.1.245:9820,192.168.1.218:9820",hdfscfgpath=/MRS/8f79ada0-
d998-4026-9020-80d6de2692ca,type=hdfs}
(1 row)
```

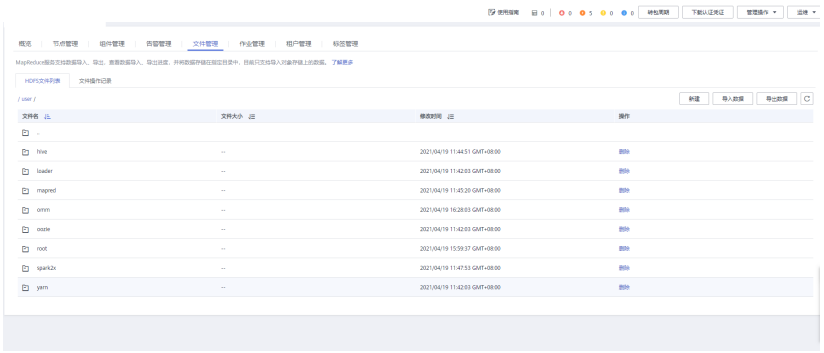
----结束

创建外表

步骤1 获取Hive的product_info_orc的文件路径。

1. 登录MRS管理控制台。
2. 选择“集群列表 > 现有集群”，单击要查看的集群名称，进入集群基本信息页面。
3. 单击“文件管理”，选择“HDFS文件列表”。
4. 进入您要导入到GaussDB(DWS)集群的数据的存储目录，并记录其路径。

图 1-1 在 MRS 上查看数据存储路径



步骤2 创建外表。 SERVER名字填写**步骤10**创建的外部服务器名称，foldername填写**步骤1**查到的路径。

```
DROP FOREIGN TABLE IF EXISTS foreign_product_info;

CREATE FOREIGN TABLE foreign_product_info
(
    product_price                integer        ,
    product_id                   char(30)       ,
    product_time                 date           ,
    product_level                char(10)       ,
    product_name                 varchar(200)   ,
    product_type1                varchar(20)    ,
    product_type2                char(10)       ,
    product_monthly_sales_cnt    integer        ,
    product_comment_time         date           ,
    product_comment_num          integer        ,
    product_comment_content      varchar(200)
) SERVER hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca
OPTIONS (
format 'orc',
encoding 'utf8',
foldername '/user/hive/warehouse/demo.db/product_info_orc/'
)
DISTRIBUTE BY ROUNDROBIN;
```

----结束

执行数据导入

步骤1 创建本地目标表。

```
DROP TABLE IF EXISTS product_info;
CREATE TABLE product_info
(
    product_price                integer        ,
    product_id                   char(30)       ,
    product_time                 date           ,
    product_level                char(10)       ,
    product_name                 varchar(200)   ,
    product_type1                varchar(20)    ,
    product_type2                char(10)       ,
    product_monthly_sales_cnt    integer        ,
    product_comment_time         date           ,
    product_comment_num          integer        ,
    product_comment_content      varchar(200)
)
with (
orientation = column,
compression=middle
)
DISTRIBUTE BY HASH (product_id);
```

步骤2 从外表导入目标表。

```
INSERT INTO product_info SELECT * FROM foreign_product_info;
```

步骤3 查询导入结果。

```
SELECT * FROM product_info;
```

----结束

1.6 教程：导入远端 DWS 数据源

大数据融合分析场景下，支持同一区域内的多套GaussDB(DWS)集群之间的数据互通互访，本实践将演示通过Foreign Table方式从远端DWS导入数据到本地端DWS。

本实践演示过程为：以gsql作为数据库客户端，gsql安装在ECS，通过gsql连接DWS，再通过外表方式导入远端DWS的数据。

操作流程

本实践预计时长40分钟，基本流程如下：

1. **准备工作**
2. **创建ECS**
3. **创建集群并下载工具包**
4. **使用GDS导入数据源**
5. **通过外表导入远端DWS数据**

准备工作

已注册华为账号并开通华为云，具体请参见**注册华为账号并开通华为云**，账号不能处于欠费或冻结状态。

创建 ECS

参见**自定义购买弹性云服务器**购买。购买后，参见**登录Linux弹性云服务器**进行登录。

须知

创建ECS过程中，注意选择与后续的DWS集群在同一个区域、可用区（本实践以“华北-北京四”、“可用区2”为例）和同一个VPC子网下，ECS的操作系统选择与下面的gsql客户端/GDS工具的操作系统一致（本例以CentOS 7.6为例），并选择以密码方式登录。

创建集群并下载工具包

步骤1 登录华为云管理控制台。

步骤2 在“服务列表”中，选择“大数据 > 数据仓库服务”，单击右上角“创建数据仓库集群”。

步骤3 参见**表1-5**进行参数配置。

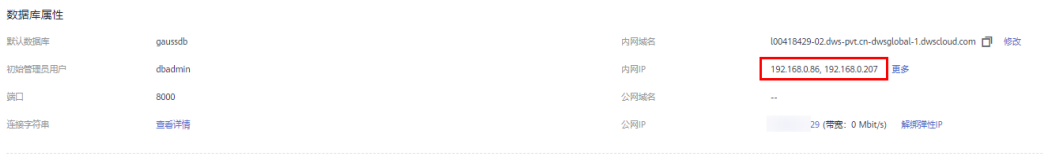
表 1-5 软件配置

参数名称	配置方式
区域	选择“华北-北京四”。 **说明** • 本指导以“华北-北京四”为例进行介绍，如果您需要选择其他区域进行操作，请确保所有操作均在同一区域进行。 • 请确保DWS跟ECS在同一个区域、可用区和同一个VPC子网下。
可用分区	可用区2
产品类型	标准数仓
计算类型	弹性云服务器
存储类型	SSD云盘
CPU架构	X86
节点规格	dws2.m6.4xlarge.8（16 vCPU | 128GB | 2000GB SSD） **说明** 如规格售罄，可选择其他可用区或规格。
热数据存储	100GB / 节点
节点数量	3
集群名称	dws-demo01
管理员用户	dbadmin
管理员密码	password，用户自定义
确认密码	password
数据库端口	8000
虚拟私有云	vpc-default
子网	subnet-default(192.168.0.0/24) **须知** 请确保与ECS在同一个VPC。
安全组	自动创建安全组
公网访问	现在购买
宽带	1Mbit/s
高级配置	默认配置

步骤4 信息核对无误，单击“立即购买”，单击“提交”。

步骤5 等待约10分钟，待集群创建成功后，单击集群名称进入“基本信息”，在“网络”区域，单击安全组名称，确认安全组规则已添加，以IP为192.168.0.x的客户端网段为例（本例gsql所在ECS的内网IP为192.168.0.90），需要添加192.168.0.0/24，端口为8000的安全组规则。

步骤6 返回到集群“基本信息”界面，记录下“内网IP”。



步骤7 返回到DWS控制台首页，左侧导航选择“连接管理”，选择ECS的操作系统（以CentOS 7.6为例，则选择“Redhat x86_64”），单击“下载”将工具包保存到本地。（工具包中包含gsql客户端和GDS工具）。

步骤8 重复执行**步骤1**~**步骤6**，创建第二套DWS集群，名称设置为dws-demo02。

----结束

准备源数据

步骤1 在本地PC指定目录下，创建以下3个.csv格式的文件，数据样例如下。

- 数据文件“product_info0.csv”

```
100,XHDK-A,2017-09-01,A,2017 Shirt Women,red,M,328,2017-09-04,715,good!
205,KDKE-B,2017-09-01,A,2017 T-shirt Women,pink,L,584,2017-09-05,40,very good!
300,JODL-X,2017-09-01,A,2017 T-shirt men,red,XL,15,2017-09-03,502,Bad.
310,QQPX-R,2017-09-02,B,2017 jacket women,red,L,411,2017-09-05,436,It's nice.
150,ABEF-C,2017-09-03,B,2017 Jeans Women,blue,M,123,2017-09-06,120,good.
```

- 数据文件“product_info1.csv”

```
200,BCQP-E,2017-09-04,B,2017 casual pants men,black,L,997,2017-09-10,301,good quality.
250,EABE-D,2017-09-10,A,2017 dress women,black,S,841,2017-09-15,299,This dress fits well.
108,CDXK-F,2017-09-11,A,2017 dress women,red,M,85,2017-09-14,22,It's really amazing to buy.
450,MMCE-H,2017-09-11,A,2017 jacket women,white,M,114,2017-09-14,22,very good.
260,OCDA-G,2017-09-12,B,2017 woolen coat women,red,L,2004,2017-09-15,826,Very comfortable.
```

- 数据文件“product_info2.csv”

```
980,"ZKDS-J",2017-09-13,"B","2017 Women's Cotton Clothing","red","M",112,,,
98,"FKQB-I",2017-09-15,"B","2017 new shoes men","red","M",4345,2017-09-18,5473
50,"DMQY-K",2017-09-21,"A","2017 pants men","red","37",28,2017-09-25,58,"good","good","good"
80,"GKLW-l",2017-09-22,"A","2017 Jeans Men","red","39",58,2017-09-25,72,"Very comfortable."
30,"HWEC-L",2017-09-23,"A","2017 shoes women","red","M",403,2017-09-26,607,"good!"
40,"IQPD-M",2017-09-24,"B","2017 new pants Women","red","M",35,2017-09-27,52,"very good."
50,"LPEC-N",2017-09-25,"B","2017 dress Women","red","M",29,2017-09-28,47,"not good at all."
60,"NQAB-O",2017-09-26,"B","2017 jacket women","red","S",69,2017-09-29,70,"It's beautiful."
70,"HWNB-P",2017-09-27,"B","2017 jacket women","red","L",30,2017-09-30,55,"I like it so much"
80,"JKHU-Q",2017-09-29,"C","2017 T-shirt","red","M",90,2017-10-02,82,"very good."
```

步骤2 使用root账户登录已创建好的ECS，执行以下命令创建数据源文件目录。

```
mkdir -p /input_data
```


步骤3 使用文件传输工具，将以上数据文档上传到ECS的/input_data目录下。

----结束

使用 GDS 导入数据源

步骤1 使用root账户登录ECS，使用文件传输工具将**步骤7**下载好的工具包上传到/opt目录下。

步骤2 在/opt目录下解压工具包。

cd /opt

unzip dws_client_8.1.x_redhat_x64.zip

步骤3 创建GDS用户，并修改数据源目录和GDS目录的属主。

groupadd gdsgrp

useradd -g gdsgrp gds_user

chown -R gds_user:gdsgrp /opt/gds

chown -R gds_user:gdsgrp /input_data

步骤4 切换到gds_user用户。

su - gds_user

步骤5 导入GDS环境变量。

说明

仅8.1.x及以上版本需要执行，老版本请跳过。

cd /opt/gds/bin

source gds_env

步骤6 启动GDS。

/opt/gds/bin/gds -d /input_data/ -p *192.168.0.90:5000* **-H** 192.168.0.0/24 **-l /opt/gds/gds_log.txt -D**

- -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。配置为GDS所在的ECS的内网IP，可与DWS通讯，本例为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。本例设置为DWS的内网IP所在的网段即可。
- -l log_file：存放GDS的日志文件路径及文件名。本教程为“/opt/gds/gds_log.txt”。
- -D：后台运行GDS。

步骤7 使用gsql连接第一套DWS集群。

1. 执行exit切换root用户，进入ECS的/opt目录，导入gsql的环境变量。

   exit

   cd /opt

   source gsql_env.sh

2. 进入/opt/bin目录，使用gsql连接第一套DWS集群。

   cd /opt/bin

   gsql -d gaussdb -h 192.168.0.8 -p 8000 -U dbadmin -W *password* -r

   - -d: 连接的数据库名，本例为默认数据库gaussdb。
   - -h：连接的DWS内网IP，即**步骤6**查询到的内网IP，本例为192.168.0.8
   - -p：DWS端口，固定为8000。
   - -U：数据库管理员用户，默认为dbadmin。
   - -W：管理员用户的密码，为**步骤3**创建集群时设置的密码，本例**password**为用户创建集群设置的密码。

步骤8 创建普通用户leo，并赋予创建外表的权限。

```
CREATE USER leo WITH PASSWORD 'password';
ALTER USER leo USEFT;
```

步骤9 切换到leo用户，创建GDS外表。

说明

以下LOCATION参数请填写为**步骤6**的GDS的监听IP和端口，后面加上/*，例如：gsfs://**192.168.0.90:5000**/*

```
SET ROLE leo PASSWORD 'password';
DROP FOREIGN TABLE IF EXISTS product_info_ext;
CREATE FOREIGN TABLE product_info_ext
(
    product_price               integer       not null,
    product_id                  char(30)      not null,
    product_time                date          ,
    product_level               char(10)      ,
    product_name                varchar(200)  ,
    product_type1               varchar(20)   ,
    product_type2               char(10)      ,
    product_monthly_sales_cnt   integer       ,
    product_comment_time        date          ,
    product_comment_num         integer       ,
    product_comment_content     varchar(200)
)
SERVER gsmpp_server
OPTIONS(
LOCATION 'gsfs://192.168.0.90:5000/*',
FORMAT 'CSV' ,
DELIMITER ',',
ENCODING 'utf8',
HEADER 'false',
FILL_MISSING_FIELDS 'true',
IGNORE_EXTRA_DATA 'true'
)
READ ONLY
LOG INTO product_info_err
PER NODE REJECT LIMIT 'unlimited';
```

步骤10 创建本地表。

```
DROP TABLE IF EXISTS product_info;
CREATE TABLE product_info
(
    product_price               integer       not null,
    product_id                  char(30)      not null,
    product_time                date          ,
    product_level               char(10)      ,
    product_name                varchar(200)  ,
    product_type1               varchar(20)   ,
    product_type2               char(10)      ,
    product_monthly_sales_cnt   integer       ,
```

```
    product_comment_time        date          ,
    product_comment_num         integer       ,
    product_comment_content     varchar(200)
)
WITH (
orientation = column,
compression=middle
)
DISTRIBUTE BY hash (product_id);
```

步骤11 从GDS外表导入数据，并查询，数据导入成功。

```
INSERT INTO product_info SELECT * FROM product_info_ext ;
SELECT count(*) FROM product_info;
```

----结束

通过外表导入远端 DWS 数据

步骤1 参见**步骤7**在ECS上连接第二套集群，其中连接地址改为第二套集群的地址，本例为192.168.0.86。

步骤2 创建普通用户jim，并赋予创建外表和server的权限。FOREIGN DATA WRAPPER固定为gc_fdw

```
CREATE USER jim WITH PASSWORD 'password';
ALTER USER jim USEFT;
GRANT ALL ON FOREIGN DATA WRAPPER gc_fdw TO jim;
```

步骤3 切换到jim用户，创建server。

```
SET ROLE jim PASSWORD 'password';
CREATE SERVER server_remote FOREIGN DATA WRAPPER gc_fdw OPTIONS
  (address '192.168.0.8:8000,192.168.0.158:8000' ,
 dbname 'gaussdb',
 username 'leo',
 password 'password'
);
```

- address：第一套集群的两个内网IP和端口，参见**步骤6**获取，本例为192.168.0.8:8000,192.168.0.158:8000。
- dbname：连接的第一套集群的数据库名，本例为gaussdb。
- username：连接的第一套集群的用户名，本例为leo。
- password：用户名密码。

步骤4 创建外表。

须知

外表的字段和约束，必须与待访问表的字段和约束保持一致。

```
CREATE FOREIGN TABLE region
(
    product_price               integer      ,
    product_id                  char(30)     ,
    product_time                date         ,
    product_level               char(10)     ,
    product_name                varchar(200)   ,
    product_type1               varchar(20)    ,
    product_type2               char(10)     ,
    product_monthly_sales_cnt   integer      ,
    product_comment_time        date         ,
    product_comment_num         integer      ,
```

```
    product_comment_content      varchar(200)
)
SERVER
    server_remote
OPTIONS
(
    schema_name 'leo',
    table_name 'product_info',
    encoding 'utf8'
);
```

- SERVER：上一步创建的server的名称，本例为server_remote。
- schema_name：待访问的第一套集群的schema名称，本例为leo。
- table_name：待访问的第一套集群的表名，参见**步骤10**获取，本例为product_info。
- encoding：保持与第一套集群的数据库编码一致，参见**步骤9**获取，本例为utf8。

步骤5 查看创建的server和外表。

```
\des+ server_remote
\d+ region
```

步骤6 创建本地表。

须知

表的字段和约束，必须与待访问表的字段和约束保持一致。

```
CREATE TABLE local_region
(
    product_price               integer        not null,
    product_id                  char(30)       not null,
    product_time                date           ,
    product_level               char(10)       ,
    product_name                varchar(200)   ,
    product_type1               varchar(20)    ,
    product_type2               char(10)       ,
    product_monthly_sales_cnt   integer        ,
    product_comment_time        date           ,
    product_comment_num         integer        ,
    product_comment_content     varchar(200)
)

WITH (
orientation = column,
compression=middle
)
DISTRIBUTE BY hash (product_id);
```

步骤7 通过外表导入数据到本地表。

```
INSERT INTO local_region SELECT * FROM region;
SELECT * FROM local_region;
```

步骤8 您也可以直接查询外表而无需将数据导入。

```
SELECT * FROM region;
```

----结束

1.7 教程：导出 ORC 数据到 MRS

GaussDB(DWS)数据库支持通过HDFS外表导出ORC格式数据至MRS，通过外表设置的导出模式、导出数据格式等信息来指定导出的数据文件，利用多DN并行的方式，将数

据从GaussDB(DWS)数据库导出到外部，存放在HDFS文件系统上，从而提高整体导出性能。

准备环境

已创建DWS集群，需确保MRS和DWS集群在同一个区域、可用区、同一VPC子网内，确保集群网络互通。

创建 MRS 分析集群

步骤1 登录华为云控制台，选择“大数据 > MapReduce服务”，单击“购买集群”，选择“自定义购买”，填写软件配置参数，单击“下一步”。

表 1-6 软件配置

参数项	取值样例
区域	华北-北京四
集群名称	mrs_01
集群版本	**MRS 1.9.2**（主推） **说明** • 8.1.1.300及以上版本集群，MRS集群支持连接1.6.*、*1.7.**、1.8.*、*1.9.**、*2.0.**、3.0.*、3.1.*及以上版本（“*”代表的是数字）。 • 8.1.1.300以下版本集群，MRS集群支持连接1.6.*、*1.7.**、1.8.*、*1.9.**、*2.0.**版本（“*”代表的是数字）。
集群类型	分析集群

步骤2 填写硬件配置参数，单击“下一步”。

表 1-7 硬件配置

参数项	取值样例
计费模式	按需计费
可用区	可用区2
虚拟私有云	vpc-01
子网	subnet-01
安全组	自动创建
弹性公网IP	*10.x.x.x*
企业项目	default
Master节点	2
分析Core节点	3
分析Task节点	0

步骤3 填写高级配置参数如下表，单击“立即购买”，等待约15分钟，集群创建成功。

表 1-8 高级配置

参数项	取值样例
标签	test01
主机名前缀	可不填写，用作集群中ECS机器或BMS机器主机名的前缀。
弹性伸缩	保持默认即可
引导操作	保持默认即可，MRS 3.x版本暂时不支持该参数。
委托	保持默认即可
数据盘加密	默认关闭，保持默认即可。
告警	保持默认即可
规则名称	保持默认即可
主题名称	选择相应的主题
Kerberos认证	默认打开
用户名	admin
密码	设置密码，该密码用于登录集群管理页面。
确认密码	再次输入设置admin用户密码
登录方式	密码
用户名	root
密码	设置密码，该密码用于远程登录ECS机器。
确认密码	再次输入设置的root用户密码
通信安全授权	勾选“确认授权”

----结束

创建 MRS 数据源连接

步骤1 登录DWS管理控制台，单击已创建好的DWS集群，**确保DWS集群与MRS在同一个区域、可用分区，并且在同一VPC子网下。**

步骤2 切换到“MRS数据源”，单击“创建MRS数据源连接”。

步骤3 选择前序步骤创建名为的“mrs_01”数据源，用户名：admin，密码：password，单击“确定”，创建成功。

----结束

创建外部服务器

步骤1 使用Data Studio连接已创建好的DWS集群。

步骤2 新建一个具有创建数据库权限的用户dbuser：

```
CREATE USER dbuser WITH CREATEDB PASSWORD 'password';
```

步骤3 切换为新建的dbuser用户：

```
SET ROLE dbuser PASSWORD 'password';
```

步骤4 创建新的mydatabase数据库：

```
CREATE DATABASE mydatabase;
```

步骤5 执行以下步骤切换为连接新建的mydatabase数据库。

1. 在Data Studio客户端的**“对象浏览器”**窗口，右键单击数据库连接名称，在弹出菜单中单击“刷新”，刷新后就可以看到新建的数据库。
2. 右键单击“mydatabase”数据库名称，在弹出菜单中单击“打开连接”。
3. 右键单击“mydatabase”数据库名称，在弹出菜单中单击“打开新的终端”，即可打开连接到指定数据库的SQL命令窗口，后面的步骤，请全部在该命令窗口中执行。

步骤6 为dbuser用户授予创建外部服务器的权限，8.1.1及以后版本，还需要授予使用public模式的权限：

```
GRANT ALL ON FOREIGN DATA WRAPPER hdfs_fdw TO dbuser;
GRANT ALL ON SCHEMA public TO dbuser;  //8.1.1及以后版本，普通用户对public模式无权限，需要赋权，8.1.1之前版本不需要执行。
```

其中FOREIGN DATA WRAPPER的名字只能是hdfs_fdw，dbuser为创建SERVER的用户名。

步骤7 执行以下命令赋予用户使用外表的权限。

```
ALTER USER dbuser USEFT;
```

步骤8 切换回Postgres系统数据库，查询创建MRS数据源后系统自动创建的外部服务器。

```
SELECT * FROM pg_foreign_server;
```

返回结果如：

```
                srvname                 | srvowner | srvfdw | srvtype | srvversion | srvacl
|                                      srvoptions
-----------------------------------------------------+----------+--------+---------+------------+--------
+-----------------------------------------------------------------------------------------------------------------------
 gsmpp_server                                       |       10 |  13673 |         |            |
 gsmpp_errorinfo_server                             |       10 |  13678 |         |            |
 hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca |    16476 |  13685 |         |            |
{"address=192.168.1.245:9820,192.168.1.218:9820",hdfscfgpath=/MRS/8f79ada0-
d998-4026-9020-80d6de2692ca,type=hdfs}
(3 rows)
```

步骤9 切换到mydatabase数据库，并切换到dbuser用户。

```
SET ROLE dbuser PASSWORD 'password';
```

步骤10 创建外部服务器。

SERVER名字、地址、配置路径保持与**步骤8**一致即可。

```
CREATE SERVER hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca FOREIGN DATA WRAPPER
HDFS_FDW
OPTIONS
(
address '192.168.1.245:9820,192.168.1.218:9820',  //MRS管理面的Master主备节点的内网IP，可与DWS通讯。
hdfscfgpath '/MRS/8f79ada0-d998-4026-9020-80d6de2692ca',
type 'hdfs'
);
```

步骤11 查看外部服务器。

```
SELECT * FROM pg_foreign_server WHERE
srvname='hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca';
```

返回结果如下所示，表示已经创建成功：

```
                srvname                 | srvowner | srvfdw | srvtype | srvversion | srvacl
|                                      srvoptions
-----------------------------------------------------+----------+--------+---------+------------+--------
+-----------------------------------------------------------------------------------------------------------------------
 hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca |    16476 |  13685 |         |            |
{"address=192.168.1.245:9820,192.168.1.218:9820",hdfscfgpath=/MRS/8f79ada0-
d998-4026-9020-80d6de2692ca,type=hdfs}
(1 row)
```

----结束

创建外表

建立不包含分区列的HDFS外表，表关联的外部服务器为hdfs_server，表对应的HDFS服务上的文件格式为‘orc’，HDFS上的数据存储路径为'/user/hive/warehouse/product_info_orc/'。

```
DROP FOREIGN TABLE IF EXISTS product_info_output_ext;
CREATE FOREIGN TABLE product_info_output_ext
(
    product_price               integer       ,
    product_id                  char(30)      ,
    product_time                date          ,
    product_level               char(10)      ,
    product_name                varchar(200)  ,
    product_type1               varchar(20)   ,
    product_type2               char(10)      ,
    product_monthly_sales_cnt   integer       ,
```

```
    product_comment_time         date          ,
    product_comment_num          integer       ,
    product_comment_content      varchar(200)
) SERVER hdfs_server_8f79ada0_d998_4026_9020_80d6de2692ca
OPTIONS (
format 'orc',
foldername  '/user/hive/warehouse/product_info_orc/',
   compression 'snappy',
    version '0.12'
) Write Only;
```

执行导出数据

创建普通表product_info_output。

```
DROP TABLE product_info_output;
CREATE TABLE product_info_output
(
    product_price               int           ,
    product_id                  char(30)      ,
    product_time                date          ,
    product_level               char(10)      ,
    product_name                varchar(200)  ,
    product_type1               varchar(20)   ,
    product_type2               char(10)      ,
    product_monthly_sales_cnt   int           ,
    product_comment_time        date          ,
    product_comment_num         int           ,
    product_comment_content     varchar(200)
)
with (orientation = column,compression=middle)
distribute by hash (product_name);
```

将表product_info_output的数据通过外表product_info_output_ext导出到数据文件中。

```
INSERT INTO product_info_output_ext SELECT * FROM product_info_output;
```

若出现以下类似信息，说明数据导出成功。

```
INSERT 0 10
```

查看导出结果

步骤1 回到MRS集群页面，单击集群名称进入集群详情界面。

步骤2 单击“文件管理 > HDFS文件列表”，在user/hive/warehouse/product_info_orc路径下查看导出的ORC格式文件。

说明

GaussDB(DWS)导出ORC数据的文件格式规则如下：

1. 导出至MRS（HDFS）：从DN节点导出数据时，以segment的格式存储在HDFS中，文件命名规则为“**mpp_数据库名_模式名_表名称_节点名称_n.orc**”。
2. 对于来自不同集群或不同数据库的数据，建议用户可以将数据导出到不同路径下。ORC格式文件大小最大为128M，Stripe大小最大为64M。
3. 导出完成后会生成_SUCCESS标记文件。

----结束

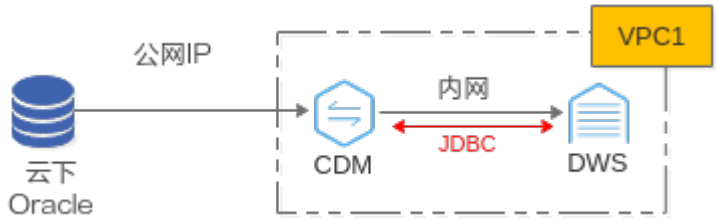
2 数据迁移实践

2.1 Oracle 迁移到 GaussDB(DWS)实践

2.1.1 迁移流程

本教程演示将Oracle业务相关的表数据迁移到GaussDB(DWS)的数据库的基本过程，迁移流程如图2-2和表2-1所示。

图 2-1 迁移场景图



须知

- 本实践以迁移Oracle中所属用户名**db_user01**下的表**APEX2_DYNAMIC_ADD_REMAIN_TEST**数据为例。
- 网络互通说明：本实践的Oracle数据库在云下，通过云数据迁移服务CDM连接Oracle和DWS。其中CDM通过公网IP与Oracle连通；CDM与DWS默认在同一个区域、虚拟私有云下，网络互通。**实际迁移过程请确保网络互通，本章节不详细介绍网络如何打通。**
- 本实践仅作为参考演示，实际迁移的复杂度可能受客户现网的网络环境、业务复杂度、节点规模、数据量等因素影响，项目实际迁移时建议在技术支持人员的指导下完成。

图 2-2 Oracle 迁移到 DWS 基本流程

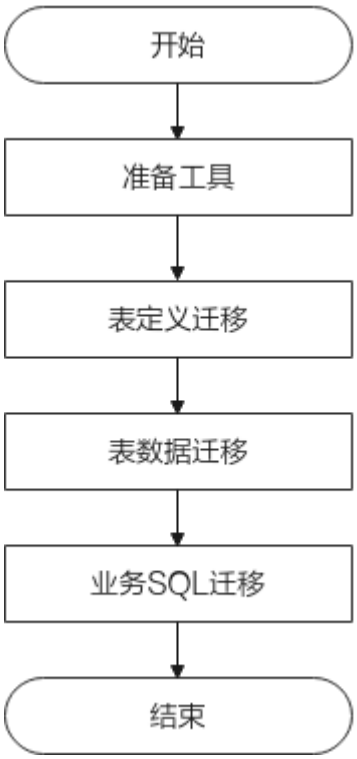


表 2-1 Oracle 迁移到 DWS 基本流程

流程	描述
准备工具	迁移前需准备的软件工具。
表定义迁移	使用PL/SQL Developer工具进行表定义迁移。
表全量数据迁移	使用华为云迁移服务CDM完成进行数据迁移。
业务SQL迁移	使用DSC语法迁移工具进行语法改写，使Oracle的业务SQL转换成适配DWS的SQL。

2.1.2 准备工具

迁移过程需准备的工具包括PL/SQL Developer、Instant Client、DSC，下载地址参见**表2-2**

表 2-2 准备工具

工具名	描述	下载地址
PL/SQL Developer	Oracle可视化开发工具	**PL/SQL Developer下载地址**
Oracle Instant Client	Oracle客户端	**Instant Client下载地址**

工具名	描述	下载地址
DSC	配套DWS的语法迁移工具	DSC下载地址

2.1.3 表定义迁移

2.1.3.1 本地安装 PLSQL 工具

操作步骤

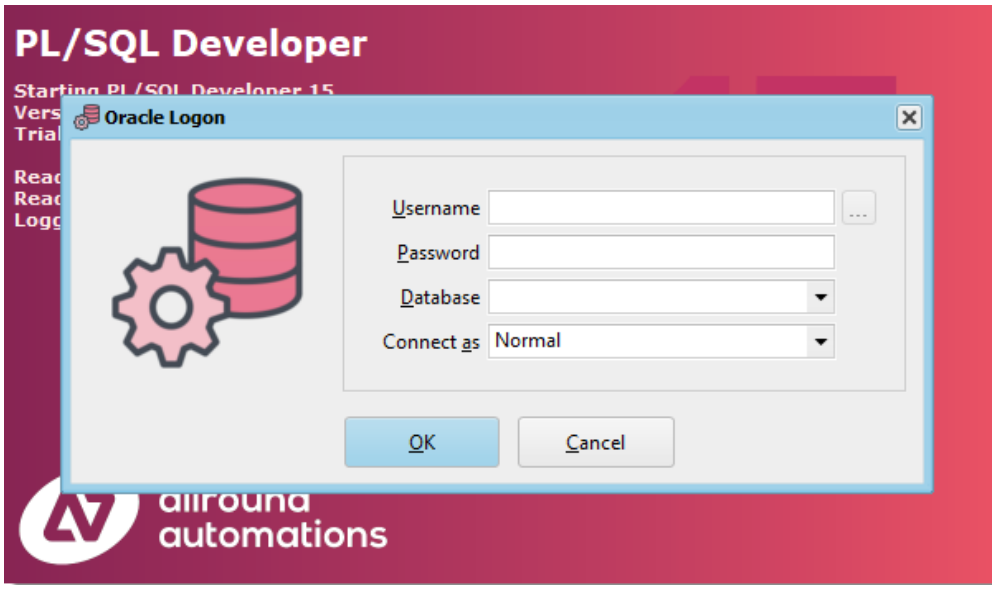
步骤1 解压PL/SQL Developer以及instance、DSC包。

步骤2 配置PL/SQL Developer的Oracle Home及OCL library。

说明

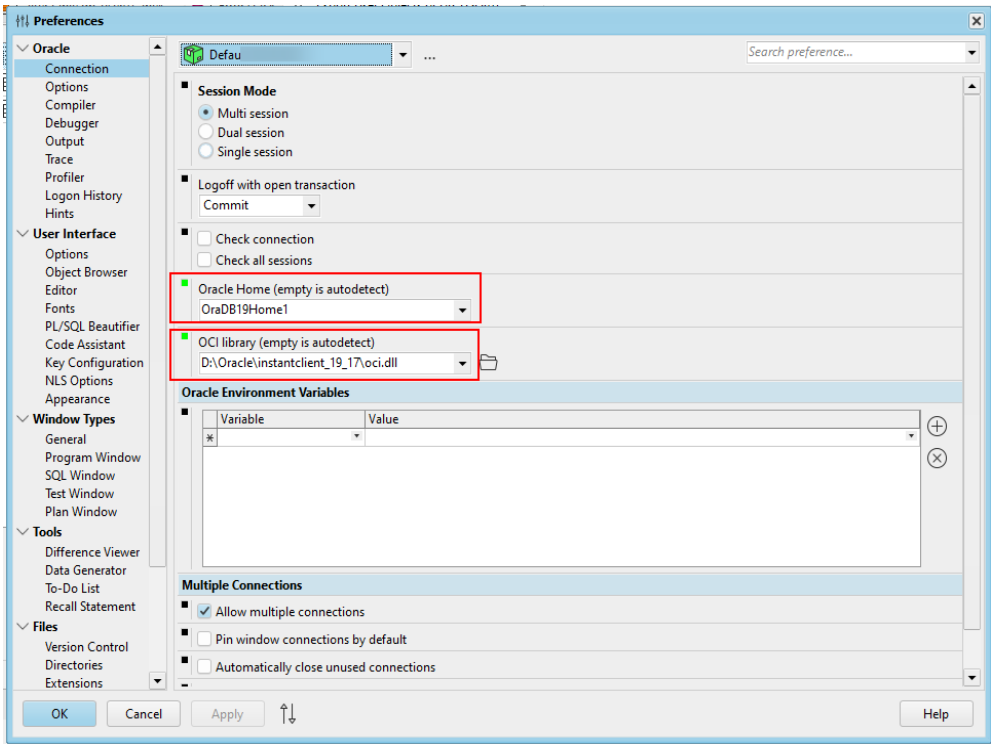
以下以试用版的PL/SQL Developer的界面为例，实际请以新界面为准。

1. 在输入密码的登录界面直接单击“取消”进入界面。

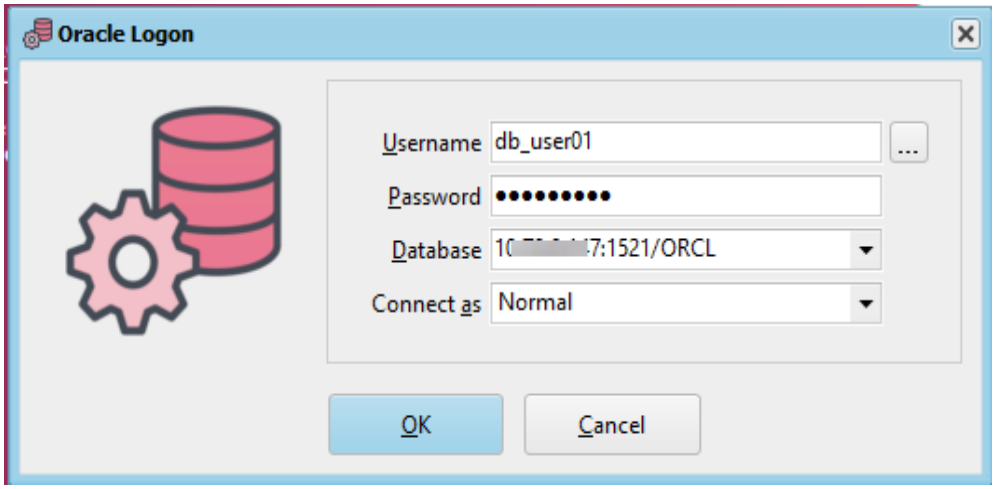


2. 选择“Configure > Preferences > Connection”，添加Oracle Home、OCl library配置。
3. 将步骤1解压好的instantclient文件目录复制到Oracle主目录中(例如D:\Oracle\instantclient_19_17)。

   将instantclient文件中的oci.dll文件目录复制到OCI库中（ 例如D:\Oracle\instantclient_19_17\oci.dll)。



步骤3 再回到PL/SQL Developer界面，依次输入用户名、密码，数据库地址，例如xx.xx.xx.xx:1521/ORCL。



步骤4 单击“确定”，能正常连接数据库，说明PL/SQL Developer安装完成。

----结束

2.1.3.2 导出表定义、语法转换迁移

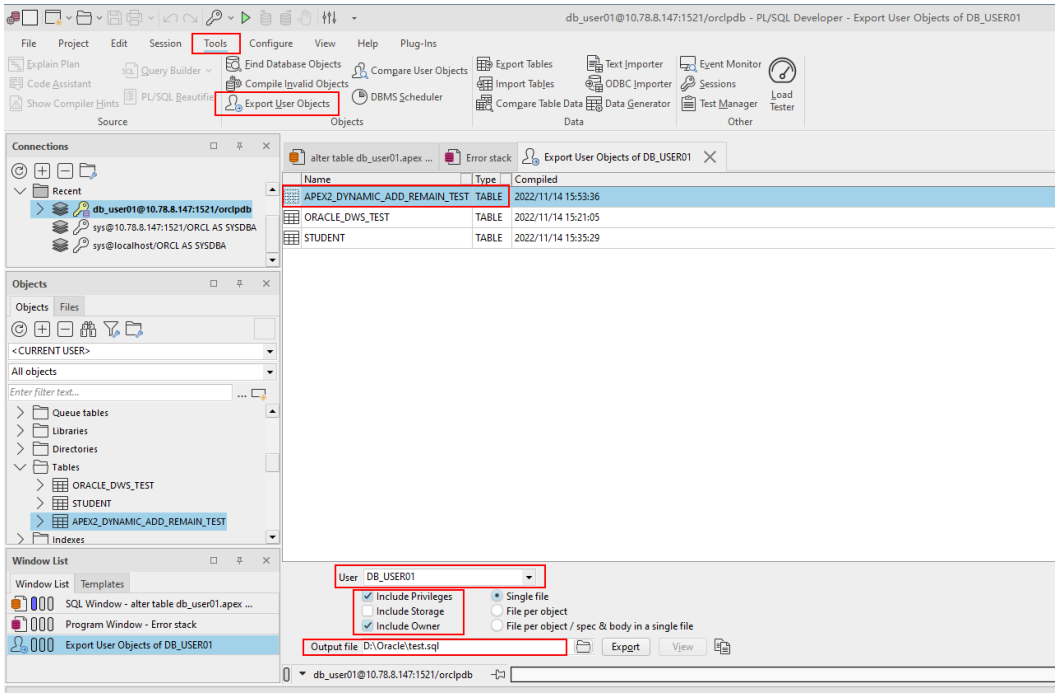
步骤1 使用拥有sysdba权限的账户（本例使用**db_user01**）登录PL/SQL Developer。

说明

以下以试用版的PL/SQL Developer的界面为例，实际请以新界面为准。

步骤2 在菜单栏选择“工具 > 导出用户对象...”。

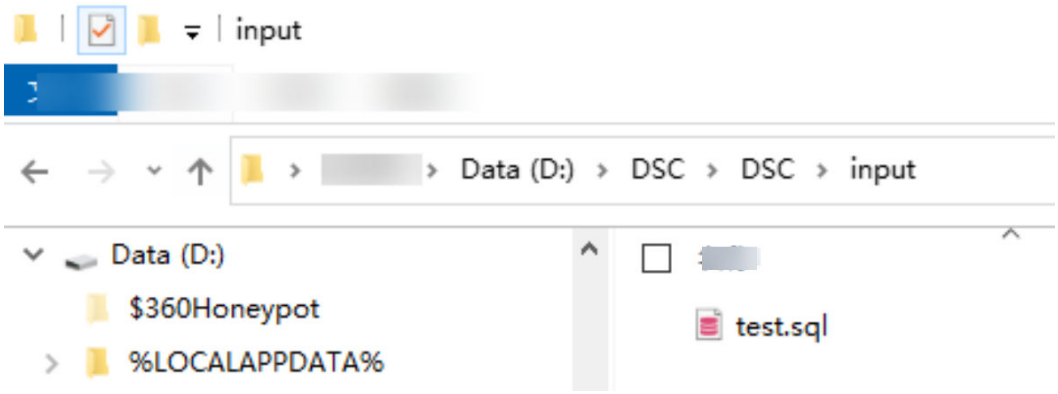
步骤3 选择当前登录用户**db_user01**，然后选择该用户下的表对象**APEX2_DYNAMIC_ADD_REMAIN_TEST**，不选择包括存储，并选择语法导出文件的路径(导出的sql文件命名为test)，单击“导出”。



导出的ddl文件如下显示。

```
D: > DSC > DSC > output > output > test.sql
1   prompt PL
2   /
3   SQL Developer Export USER Objects FOR USER DB_USER01@10.78.8.147 :1521 / ORCLPDB \echo Created by 1
4    /* SET define off; */
5   /*spool test.log*/
6   \echo
7    \echo Creating table APEX2_DYNAMIC_ADD_REMAIN_TEST
8    \echo ============================================
9    \echo
10   CREATE
11       UNLOGGED TABLE
12           DB_USER01.APEX2_DYNAMIC_ADD_REMAIN_TEST (
13               id INTEGER NOT NULL
14               ,TIME DATE
15               ,add_users NUMBER
16               ,remain_users NUMBER
17               ,PRIMARY KEY (ID)
18           ) ;
19  \echo Done
20   /*spool off*/
21  SET define
22      ON ;
```

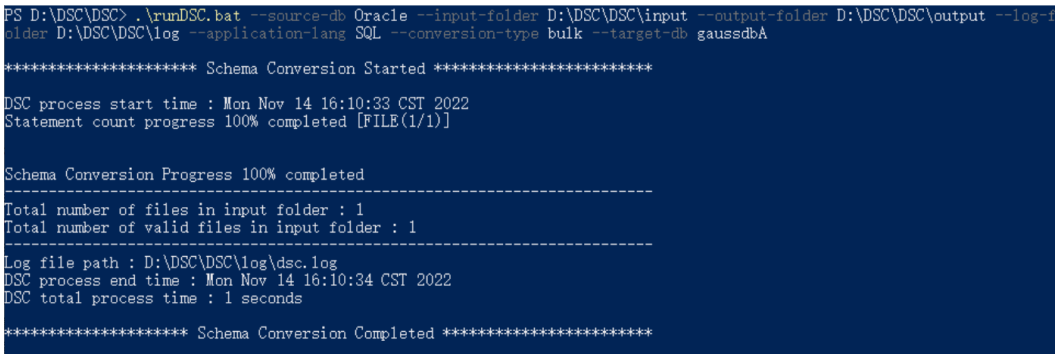
步骤4 将导出的DDL文件放在解压后的DSC文件夹的input目录下。

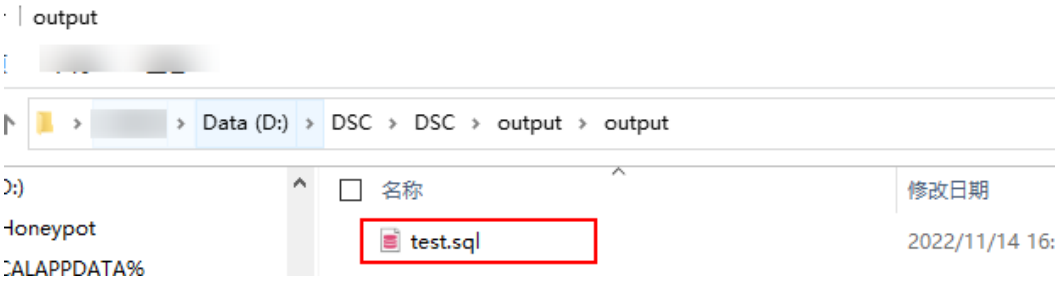


步骤5 在runDSC.bat同级目录下shift+鼠标右键，选择在此处打开power shell窗口，并执行转换。其中**D:\DSC\DSC\input**、**D:\DSC\DSC\output**、**D:\DSC\DSC\log**切换为实际DSC的路径。

```
.\runDSC.bat --source-db Oracle --input-folder D:\DSC\DSC\input --output-folder D:\DSC\DSC\output --log-folder D:\DSC\DSC\log --application-lang SQL --conversion-type bulk --target-db gaussdbA
```

步骤6 转换完成，在DSC的output路径下自动生成转换后的ddl文件。





步骤7 由于DWS的表定义结构与Oracle存在差异，需要手动修改转换后的表定义。

如下，将文件中的\echo整体注释掉（如果使用gsql工具导入表定义的话，不需要注释），同时手动修改指定表的分布列(distribute by hash (列名))。

- 修改前：

```
D: > DSC > DSC > output > output > test.sql
prompt PL
/
SQL Developer Export USER Objects FOR USER DB_USER01@10.78.8.147 :1521 / ORCLPDB \echo Created by l[illegible] on 2
 /* SET define off; */
/*spool test.log*/
\echo
 \echo Creating table APEX2_DYNAMIC_ADD_REMAIN_TEST
 \echo ========================================
 \echo
 CREATE
     UNLOGGED TABLE
          DB_USER01.APEX2_DYNAMIC_ADD_REMAIN_TEST (
               id INTEGER NOT NULL
               ,TIME DATE
               ,add_users NUMBER
               ,remain_users NUMBER
               ,PRIMARY KEY (ID)
          ) ;
\echo Done
 /*spool off*/
SET define
     ON ;
```

- 修改后：

```
prompt PL
/
SQL Developer Export USER Objects FOR USER DB_USER01@10.78.8.147 :1521 / ORCLPDB \echo Created by [illegible] on
 /* SET define off; */
/*spool test.log*/
 --\echo
 --\echo Creating table APEX2_DYNAMIC_ADD_REMAIN_TEST
  --\echo ========================================
  --\echo
 CREATE
     UNLOGGED TABLE
          DB_USER01.APEX2_DYNAMIC_ADD_REMAIN_TEST (
               id INTEGER NOT NULL
               ,TIME DATE
               ,add_users NUMBER
               ,remain_users NUMBER
               ,PRIMARY KEY (ID)
          ) DISTRIBUTE BY HASH (ID);
\echo Done
 /*spool off*/
SET define
     ON ;
```

说明

Hash分布表的分布列选取至关重要，需要满足以下原则：

1. 列值应比较离散，以便数据能够均匀分布到各个DN。例如，考虑选择表的主键为分布列，如在人员信息表中选择身份证号码为分布列。
2. 在满足第一条原则的情况下尽量不要选取存在常量filter的列。例如，表dwcjk相关的部分查询中出现dwcjk的列zqdh存在常量的约束(例如zqdh=’000001’)，那么就应当尽量不用zqdh做分布列。
3. 在满足前两条原则的情况，考虑选择查询中的连接条件为分布列，以便Join任务能够下推到DN中执行，且减少DN之间的通信数据量。

步骤8 完成DWS集群的创建，参见**创建集群**。

步骤9 连接DWS数据库，参见**使用Data Studio连接集群**，使用系统管理员dbadmin用户连接，首次默认先连接默认数据库gaussdb。

步骤10 创建新的目标数据库test，再切换到test数据库。

```
CREATE DATABASE test WITH ENCODING 'UTF-8' DBCOMPATIBILITY 'ORA' TEMPLATE template0;
```

步骤11 创建新的Schema切换到新的Schema，Schema名称保持与Oracle的用户名（本例为db_user01）一致。

```
CREATE SCHEMA db_user01;
SET CURRENT_SCHEMA = db_user01;
```


步骤12 复制**步骤7**中转换后的DDL语句到Data Studio中执行。

步骤13 在DWS集群的test库中的Schema xxx下能查到APEX2_DYNAMIC_ADD_REMAIN_TEST表即为表定义迁移完成。

```
SELECT COUNT(*) FORM db_user01.APEX2_DYNAMIC_ADD_REMAIN_TEST;
```

----结束

2.1.4 表全量数据迁移

2.1.4.1 配置 DWS 数据源连接

步骤1 参见**创建CDM集群**先完成CDM集群创建并**绑定弹性IP**。

须知

确保CDM集群与DWS集群在同一个区域、虚拟私有云下，以保证网络互通。

步骤2 在CDM管理控制台的“集群管理”页面，单击集群操作列的“作业管理”，选择“连接管理 > 新建连接”。

步骤3 连接器类型选择“数据仓库服务（DWS）”，单击“下一步”。

步骤4 配置DWS连接，点击“测试”通过后，单击“保存”。

表 2-3 DWS 连接信息

参数项	取值
名称	dws
数据库服务器	单击“选择”，从集群列表中选择要连接的DWS集群。 **说明** 系统会自动刷出同一个区域、同一个VPC下的DWS集群，如果没有，则需要手动填写网络已连通的DWS的访问IP。
端口	8000
数据库名称	test
用户名	dbadmin
密码	dbadmin用户密码
使用Agent	否

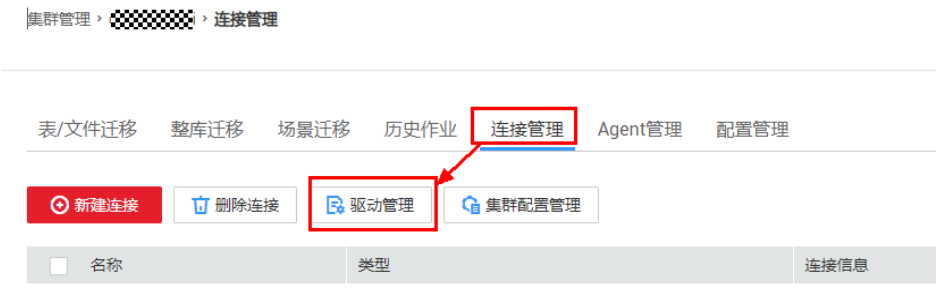
----结束

2.1.4.2 配置 Oracle 数据源连接

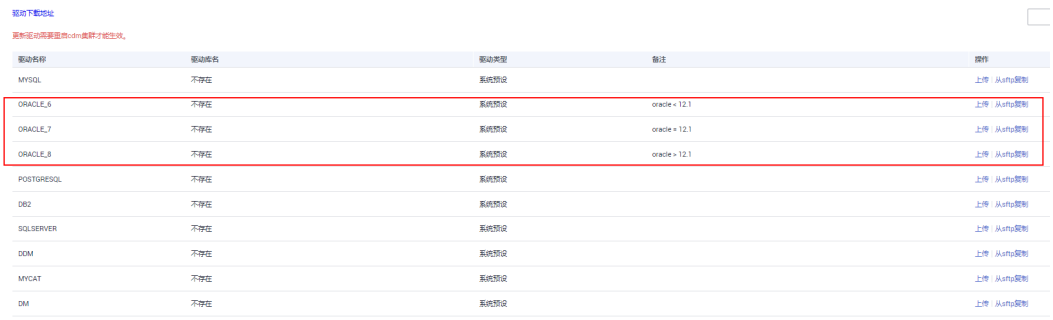
数据从Oracle迁移到GaussDB(DWS)，首先要配置Oracle数据源连接。

操作步骤

步骤1 在CDM管理控制台的“集群管理”页面，单击集群操作列的“作业管理”，选择“连接管理 > 驱动管理”。



步骤2 单击“ORACLE”右侧的“上传”，选择Oracle驱动包（如果本地没有驱动包，请参照**驱动包**下载），单击“上传文件”。



步骤3 在CDM管理控制台的“集群管理”页面，单击集群操作列的“作业管理”，选择“连接管理 > 新建连接”。

步骤4 连接器类型选择“Oracle”，单击“下一步”。

步骤5 配置Orcale连接，单击“测试”通过后，单击“保存”。

表 2-4 Oracle 连接信息

参数项	取值
名称	oracle
数据库服务器	192.168.1.100（示例，请填写Oracle实际的公网IP）
端口	1521
数据库连接类型	Service Name
数据库名称	orcl
用户名	db_user01
密码	-

参数项	取值
使用本地API	否
使用Agent	否
ORACLE版本	高于12.1

----结束

2.1.4.3 表迁移

操作步骤

步骤1 在CDM管理控制台的“集群管理”页面，单击集群操作列的“作业管理”，选择“表/文件迁移 > 新建作业”。

步骤2 选择源端以及目的端配置。

步骤3 配置源端作业参数，根据待迁移的数据库类型配置对应参数：

表 2-5 源端作业参数

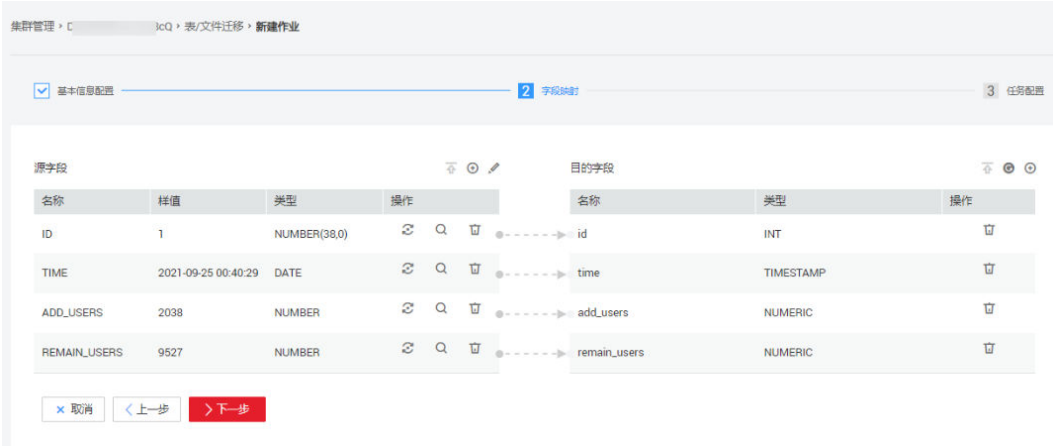
源端参数	取值样例
模式或表空间	db_user01
使用SQL语句	否
表名	APEX2_DYNAMIC_ADD_REMAIN_TEST
Where子句	-
分区字段是否允许空值	是

步骤4 配置目的端作业参数，根据待导入数据的云服务配置对应参数。

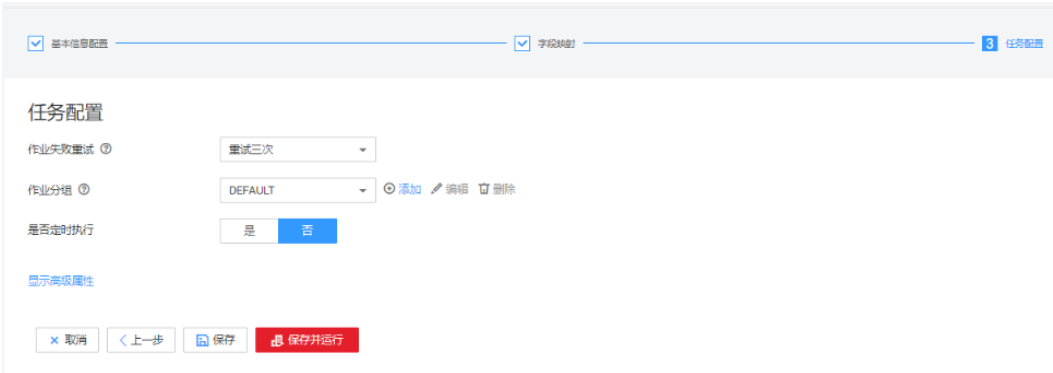
表 2-6 目的端作业参数

1. 参数名	取值样例
模式或表空间	db_user01
自动创表	不自动创建
表名	apex2_dynamic_add_remain_test
导入开始前	清除全部数据
导入模式	COPY
先导入阶段表	否
导入前准备语句	-
导入后完成语句	analyze db_user01. apex2_dynamic_add_remain_test;

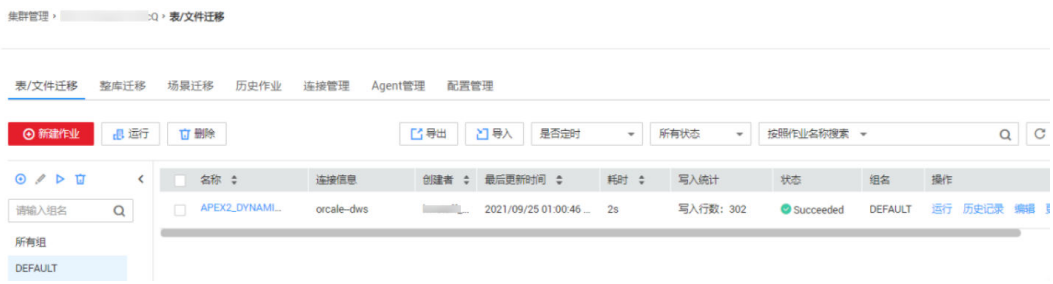
步骤5 源端表与目标端表字段对应关系。



步骤6 任务配置失败重试三次，保存并运行。



步骤7 任务执行完成，数据迁移结束。



----结束

2.1.4.4 验证

步骤1 在DWS新建的test数据库下，执行以下SQL语句查询表apex2_dynamic_add_remain_test的行数，如与源数据行数一致，说明数据一致。

```
SELECT COUNT(*) FROM  db_user01.apex2_dynamic_add_remain_test;
```

步骤2 执行以下语句检查数据倾斜率。

数据分布率在10%以内说明数据离散分布正常。数据迁移完成。

```
SELECT TABLE_SKEWNESS('db_user01.apex2_dynamic_add_remain_test');
```

	table_skewness
1	("dn_6001_6002 ",97,32.119%)
2	("dn_6003_6004 ",105,34.768%)
3	("dn_6005_6006 ",100,33.113%)

----结束

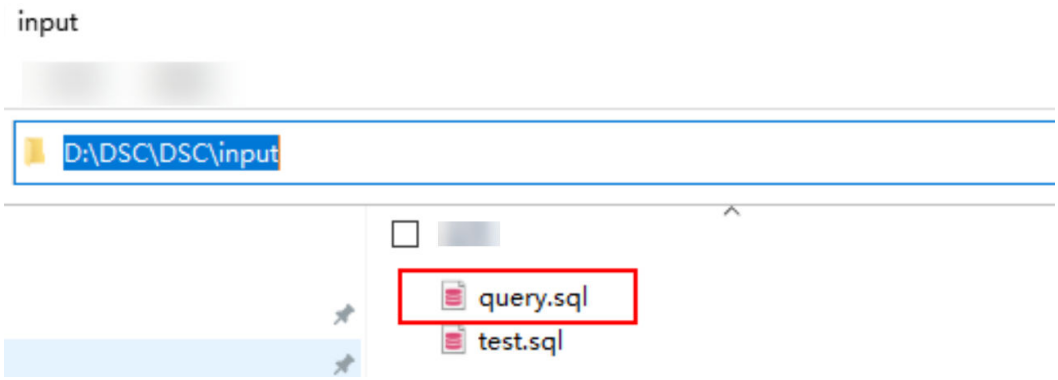
2.1.5 业务 SQL 迁移

2.1.5.1 业务语法转换迁移

步骤1 假设Oracle有原业务SQL如下，将其保存成query.sql格式文件。

```
-- 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
SELECT
id,
count(*),
sum(remain_users)
FROM LYC.APEX2_DYNAMIC_ADD_REMAIN_TEST
HAVING id <= 5
GROUP BY id;

-- UNIQUE关键字迁移为DISTINCT关键字
SELECT UNIQUE add_users FROM LYC.APEX2_DYNAMIC_ADD_REMAIN_TEST;

-- “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”
SELECT NVL2(add_users, 1, 2) FROM LYC.APEX2_DYNAMIC_ADD_REMAIN_TEST SHERE rownum <= 2;
```

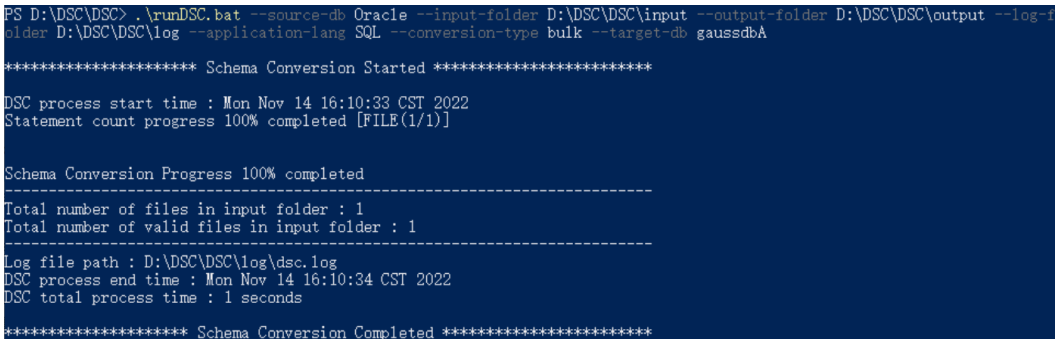
步骤2 将**步骤1**的query.sql文件放在解压后的DSC文件夹的input目录下。



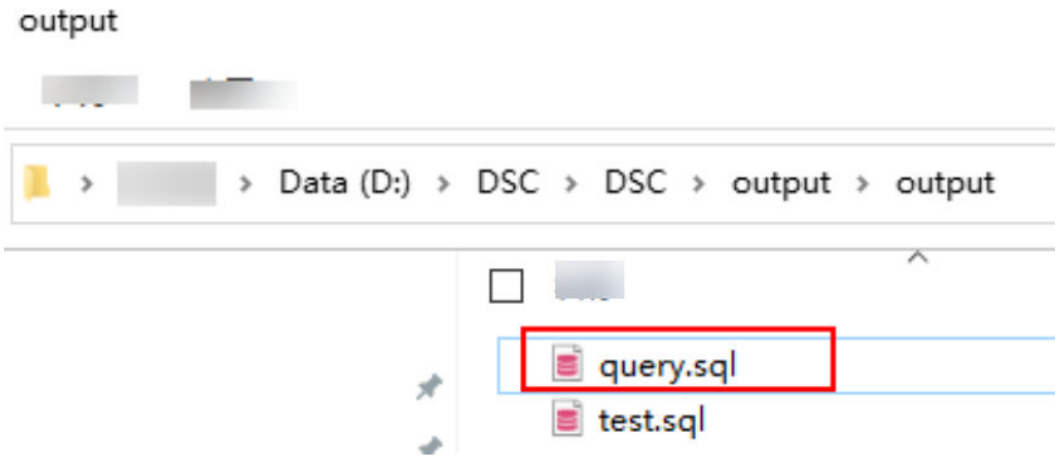
步骤3 在runDSC.bat当前目录下shift+鼠标右键，选择在此处打开power shell窗口，并执行转换。

其中**D:\DSC\DSC\input**、**D:\DSC\DSC\output**、**D:\DSC\DSC\log**请替换为实际的DSC路径。

```
.\runDSC.bat --source-db Oracle --input-folder D:\DSC\DSC\input --output-folder D:\DSC\DSC\output --log-folder D:\DSC\DSC\log --application-lang SQL --conversion-type bulk --target-db gaussdbA
```



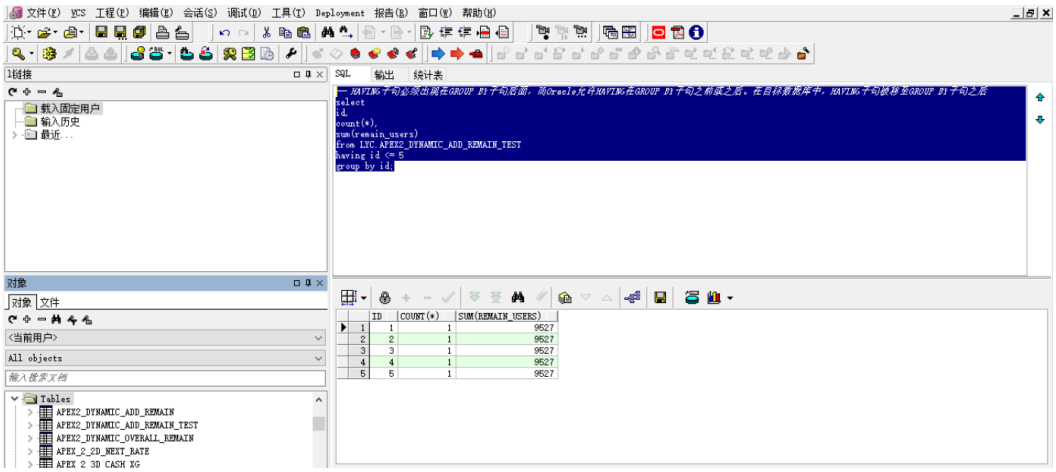
步骤4 转换完成后，在output路径下生成转换后的DML文件。

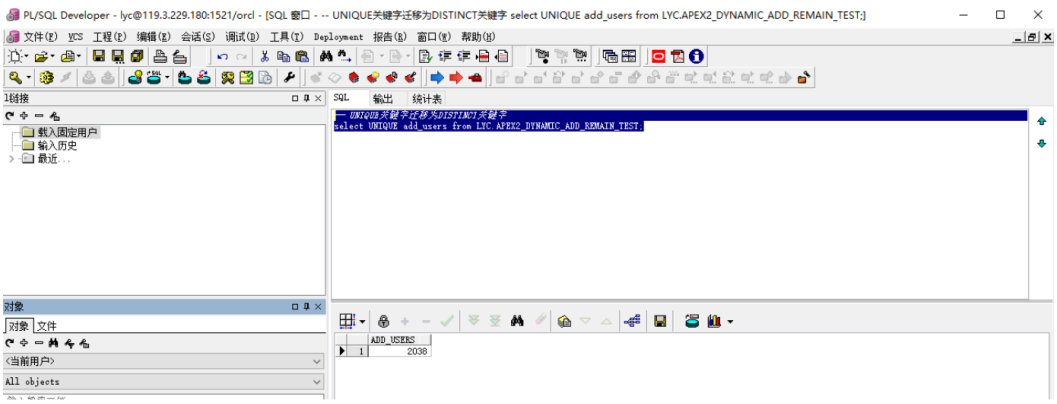


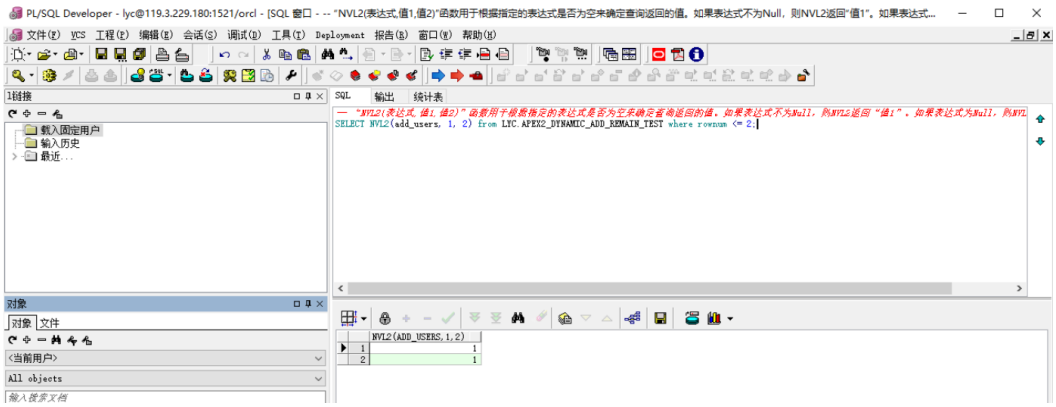
----结束

2.1.5.2 验证

步骤1 在Oracle中执行迁移前的业务SQL，执行结果如下图。



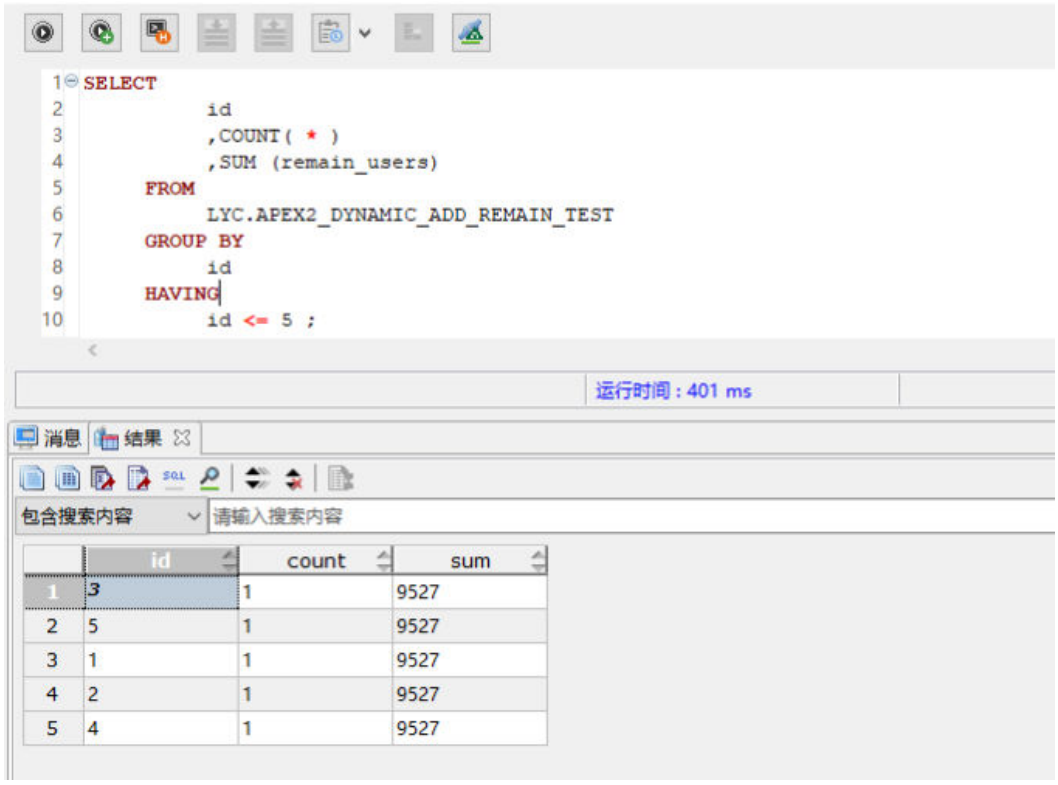


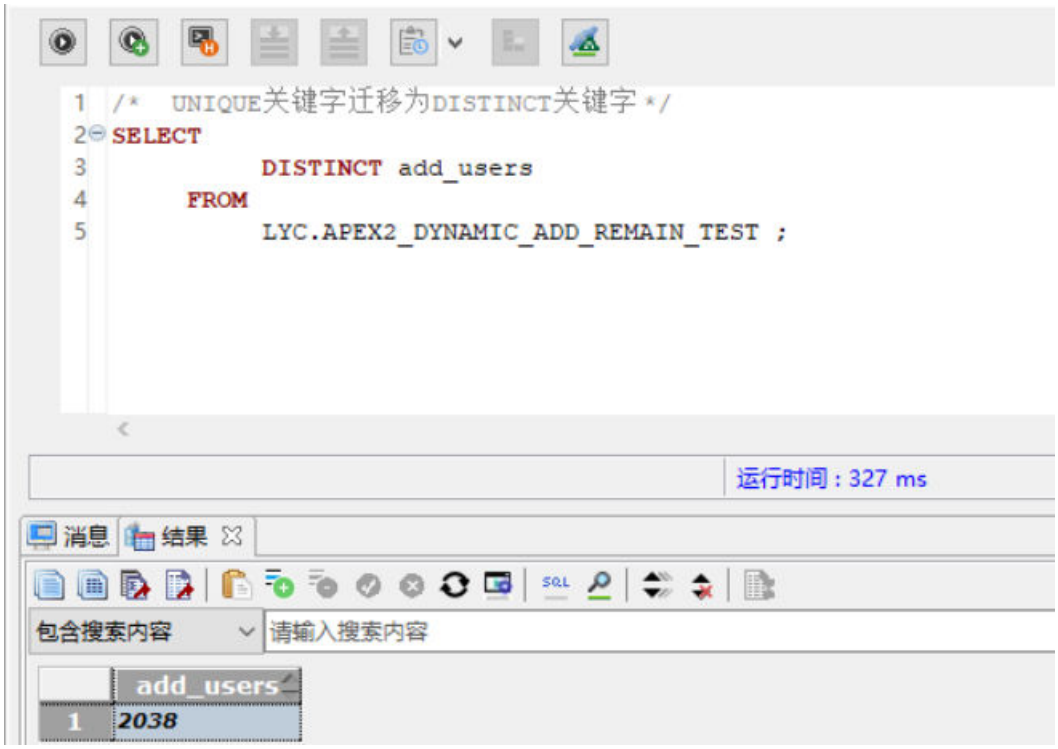


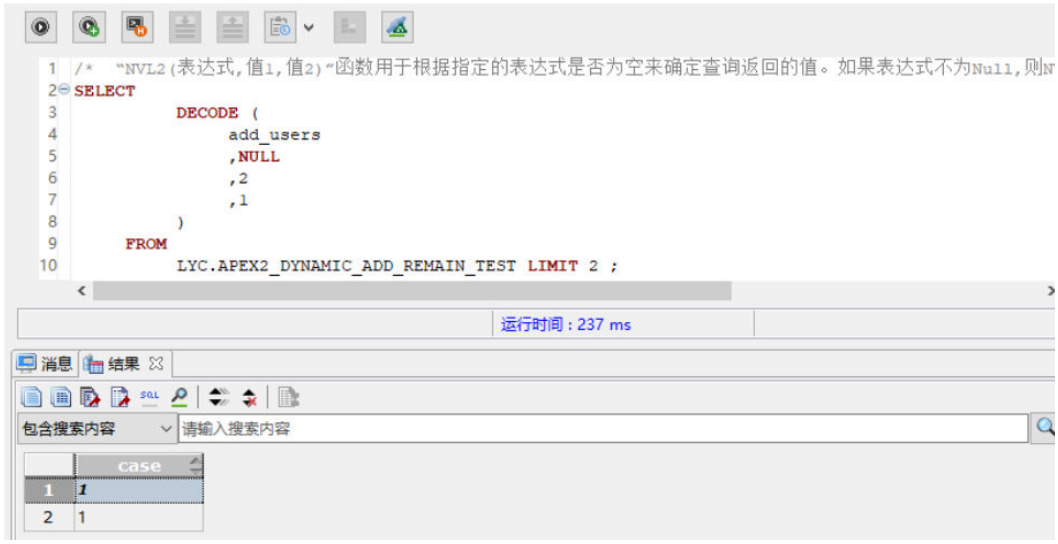
步骤2 在Data Studio中执行迁移后业务SQL。

```
/*  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 */
SELECT
        id
        ,COUNT( * )
        ,SUM (remain_users)
    FROM
        LYC.APEX2_DYNAMIC_ADD_REMAIN_TEST
    GROUP BY
        id
    HAVING
        id <= 5 ;
/*  UNIQUE关键字迁移为DISTINCT关键字 */
SELECT
        DISTINCT add_users
    FROM
        LYC.APEX2_DYNAMIC_ADD_REMAIN_TEST ;
/*  "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" */
SELECT
        DECODE (
            add_users
            ,NULL
            ,2
            ,1
        )
    FROM
        LYC.APEX2_DYNAMIC_ADD_REMAIN_TEST LIMIT 2 ;
```

步骤3 查看执行后的结果如下图。







步骤4 对比Oracle和DWS业务SQL的执行结果，结果一致，业务迁移完成。

----结束

2.2 MySQL 表数据实时同步到 GaussDB(DWS)实践

本实践演示通过华为云数据复制服务（Data Replication Service，简称DRS）完成MySQL数据实时同步到GaussDB(DWS)的基本过程。了解DRS服务，请参见**什么是数据复制服务**。

本实践预计时长60分钟，基本流程如下：

1. **准备工作**

2. **步骤一：准备MySQL源表数据**
3. **步骤二：创建DWS集群**
4. **步骤三：创建DRS同步任务**
5. **步骤四：验证数据同步**

场景描述

大数据分析场景下，MySQL作为OLTP数据库，接入GaussDB(DWS)数仓进行OLAP业务分析后，对于MySQL实时写入的数据也需要实时同步入仓，此时可通过DRS实现。

图 2-3 DRS 实时同步 MySQL

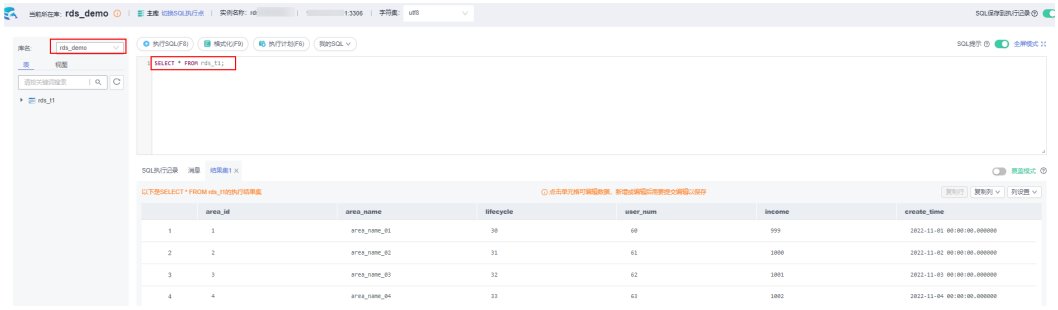
准备工作

- 已注册华为账号并开通华为云，具体请参见**注册华为账号并开通华为云**，且在使用GaussDB(DWS) 前检查账号状态，账号不能处于欠费或冻结状态。
- 已准备待迁移的MySQL源表数据，本实验使用华为云数据库RDS的MySQL数据库作为源数据举例，如源数据为MySQL线下版本，请确保网络连通。

步骤一：准备 MySQL 源表数据

步骤1 已购买云数据库RDS的MySQL引擎（本实践以MySQL 8.0.x为例），参见**购买RDS实例**。

步骤2 已创建源数据库rds_demo，字符集utf8mb4，且库中已存在表rds_t1和数据。



----结束

步骤二：创建 DWS 集群

步骤1 **创建集群**，为确保网络连通，GaussDB(DWS) 集群与RDS在同一个区域下。

步骤2 在GaussDB(DWS)控制台的集群管理页面，单击指定集群所在行操作列“登录”按钮。

> **说明**
>
> 本实践以8.1.3.x版本为例，8.1.2及以前版本不支持此登录方式，可以**使用Data Studio连接集群**。

步骤3 登录GaussDB(DWS)数据库后，创建待同步MySQL数据的数据库：rds_demo。

```
CREATE DATABASE rds_demo WITH ENCODING 'UTF-8' DBCOMPATIBILITY 'mysql' TEMPLATE template0;
```

步骤4 切换到rds_demo数据库，创建名称为rds_demo的Schema。

```
CREATE SCHEMA rds_demo;
```

步骤5 在rds_demo的Schema下创建表rds_t1。

```
CREATE TABLE rds_demo.rds_t1 (
  area_id varchar(256) NOT NULL,
  area_name varchar(256) DEFAULT NULL,
  lifecycle varchar(256) DEFAULT NULL,
  user_num int DEFAULT NULL,
  income  bigint DEFAULT NULL,
  create_time timestamp DEFAULT CURRENT_TIMESTAMP,
   PRIMARY KEY (area_id)
)distribute by hash(area_id);
 COMMENT on column rds_demo.rds_t1.area_id is '区域编码';
 COMMENT on column rds_demo.rds_t1.area_name is '区域名称';
 COMMENT on column rds_demo.rds_t1.lifecycle is '生命周期';
 COMMENT on column rds_demo.rds_t1.user_num is '各个生命周期用户数';
 COMMENT on column rds_demo.rds_t1.income is '区域总收入';
 COMMENT on column rds_demo.rds_t1.create_time is '创建时间';
```

步骤6 查询表数据，目前为空表。

```
SELECT * FROM rds_demo.rds_t1;
```

----结束

步骤三：创建 DRS 同步任务

步骤1 选择“服务列表 > 数据库 > 数据复制服务 DRS”，切换至DRS控制台。

步骤2 在左侧导航栏选择“实时同步管理”，单击右上角的“创建同步任务”。

步骤3 填写基本参数，参见表2-7。

表 2-7 基本参数

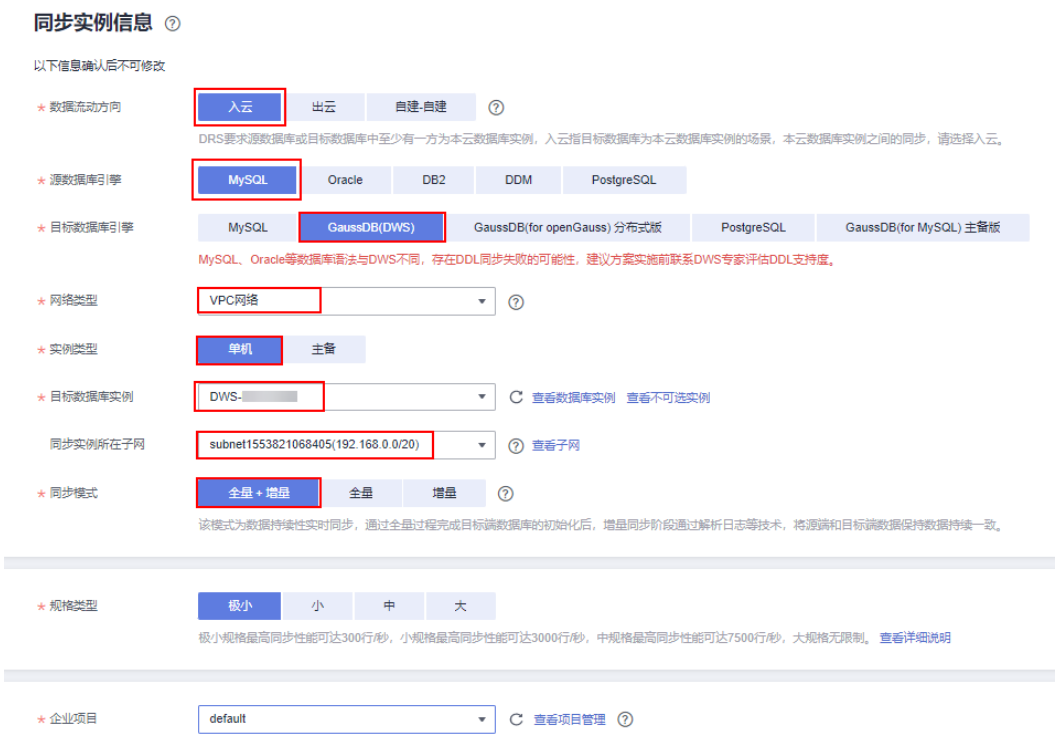
参数名称	取值
计费模式	按需计费
区域	华北-北京四，确保RDS与DWS所选区域为同一个区域。
项目	华北-北京四
任务名称	DRS-DWS
描述	-

步骤4 继续填写以下参数，参见表2-8。

表 2-8 同步实例信息

参数名称	取值
数据流动方向	入云
源数据引擎	MySQL
目标数据库引擎	GaussDB(DWS)
网络类型	本实践选择“**VPC网络**”，如果MySQL为线下版本，则需要选择“**公网网络**”。

参数名称	取值
实例类型	单机
目标数据库实例	选择**步骤二：创建DWS集群**的集群名称。
同步实例所在子网	选择DWS集群所在子网。本实践RDS、DWS都在同一个VPC和子网下。
同步模式	全量+增量。
规格类型	本实践选择“极小”，实际请按数据量和同步速率要求进行评估。



步骤5 单击“下一步”，确认无误后单击“我已阅读并知晓”。

等待实例同步，大约5~10分钟。

步骤6 实例同步成功后，填写源库信息，单击“测试连接”，显示测试连接成功即可。

表 2-9 源库信息

参数项	取值
数据库类型	RDS实例
数据库实例名称	选择创建的RDS实例。
数据库用户名	root
数据库密码	****

步骤7 填写目标库信息，单击“测试连接”，显示测试连接成功即可。

表 2-10 目标库信息

参数项	取值
数据库用户名	dbadmin
数据库密码	****

步骤8 单击“下一步”，确认无误后单击“同意，并继续”。

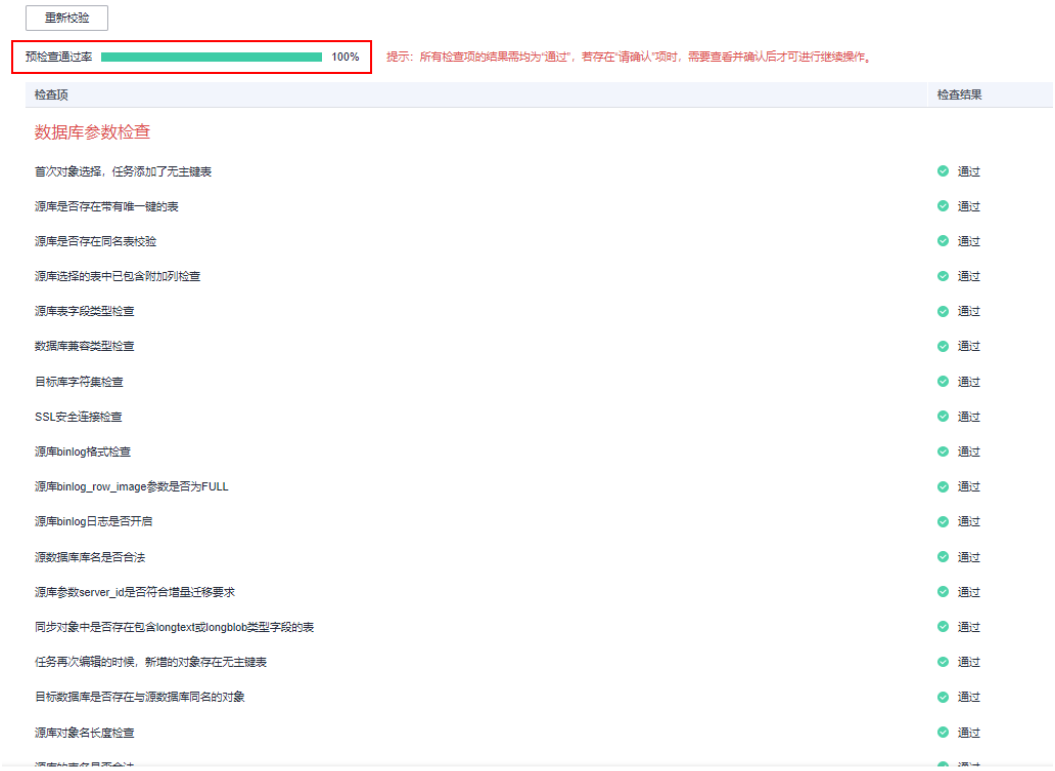
步骤9 设置同步策略，参见表2-11。

表 2-11 同步策略

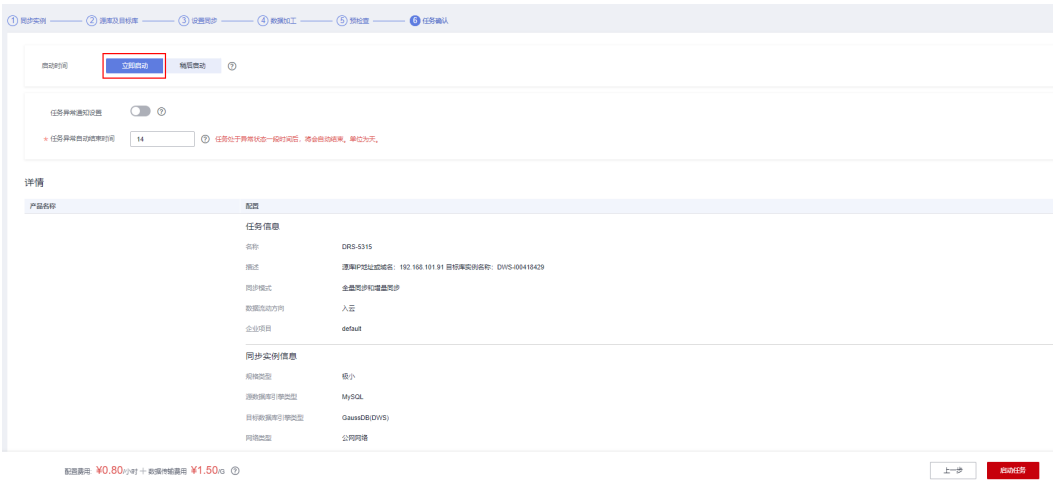
参数项	取值
流速模式	不限速
同步对象类型	同步数据
增量阶段冲突策略	覆盖
数据同步拓扑	一对一
增量支持DDL	默认值
同步对象	表级同步。 从源数据库中勾选要同步的表，本实践为rds_demo下的rds_t1。 并填写同步到DWS的数据库名：**rds_demo**

步骤10 单击“下一步”，确认无误，再单击“下一步”。

等待数据库参数检查成功，如果不成功，单击“重新校验”。



步骤11 单击“下一步”，启动时间选择“立即启动”，其他信息核对无误，单击右下角“启动任务”。



步骤12 弹出窗口确认无误后勾选“我已阅读启动前须知”，并单击“启动任务”。

回到DRS的“实时同步管理”页面，等待约5~10分钟，启动成功。

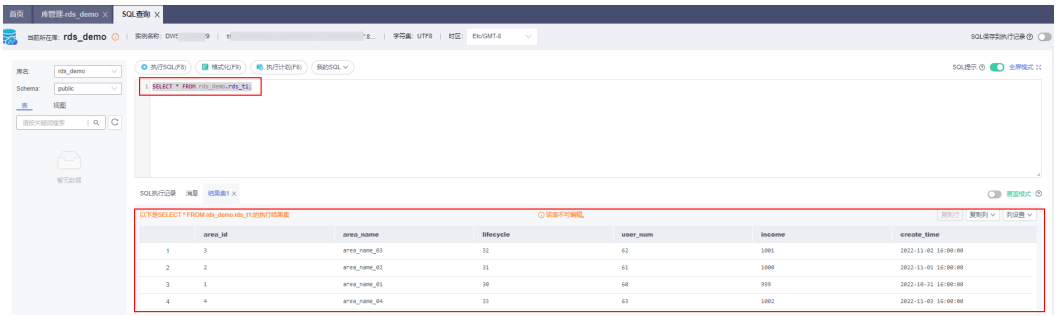
启动成功后，数据开始同步，等待约5分钟，继续执行**步骤四：验证数据同步**。

----结束

步骤四：验证数据同步

步骤1 回到GaussDB(DWS)管理控制台，重新登录DWS，执行以下语句再次查询表数据，显示结果如下图所示表示全量数据同步成功。

```
SELECT * FROM rds_demo.rds_t1;
```



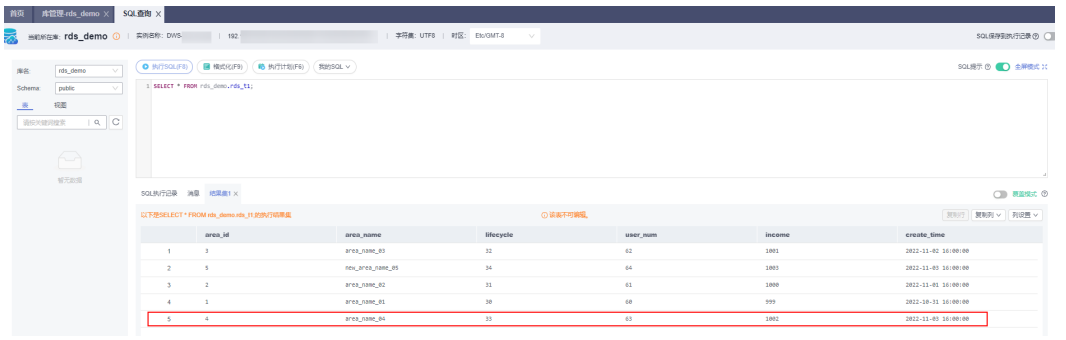
步骤2 切换到RDS管理控制台，登录RDS数据库，向表rds_t1插入新的数据。

```
INSERT INTO rds_t1 VALUES ('5','new_area_name_05',34,64,1003,'2022-11-04');
```

步骤3 切回DWS数据库，执行以下语句查询表数据。

查询结果中新增一条行数据，表示MySQL的更新数据已实时同步到GaussDB(DWS)。

```
SELECT * FROM rds_demo.rds_t1;
```



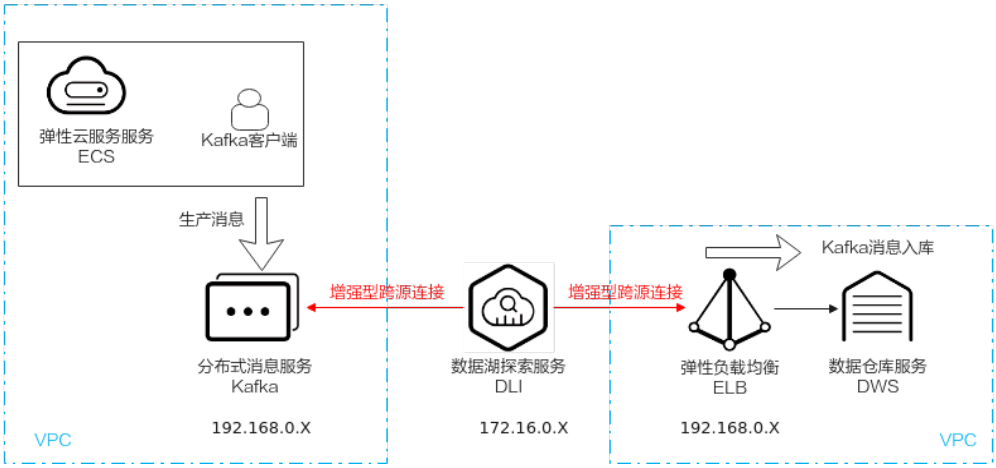
----结束

2.3 通过 DLI Flink 作业将 Kafka 数据实时写入 DWS

本实践演示通过**数据湖探索服务 DLI** Flink作业将**分布式消息服务 Kafka**的消费数据实时同步至GaussDB(DWS)数据仓库，实现Kafka实时入库到GaussDB(DWS)的过程。演示过程包括实时写入和更新已有数据的场景。

- 了解DLI请参见**数据湖产品介绍** 。
- 了解Kafka请参见**分布式消息服务Kafka产品介绍**。

图 2-4 Kafka 实时入库 DWS



本实践预计时长90分钟，实践用到的云服务包括**虚拟私有云 VPC及子网、弹性负载均衡 ELB、弹性云服务器 ECS、对象存储服务 OBS、分布式消息服务 Kafka、数据湖探索 DLI**和**数据仓库服务 GaussDB(DWS)**，基本流程如下：

1. **准备工作**
2. **步骤一：创建Kafka实例**
3. **步骤二：创建绑定ELB的DWS集群和目标表**
4. **步骤三：创建DLI队列**
5. **步骤四：分别创建Kafka和DWS的增强型跨源连接**
6. **步骤五：准备DWS对接Flink工具dws-connector-flink**

7. **步骤六：创建并编辑DLI Flink作业**
8. **步骤七：通过Kafka客户端生产和修改消息**

场景描述

假设数据源Kafka的样例数据是一个用户信息表，如**表2-12**所示， 包含 id，name，age三个字段。其中id是唯一且固定的字段，多个业务系统会公用，业务上一般不需要修改，仅修改姓名name，年龄age。

首先，通过Kafka生产以下三组数据，通过DLI Flink作业完成数据同步到数据仓库服务GaussDB(DWS)。接着，需要修改id为2和3的用户为新的jim和tom，再通过DLI Flink作业完成数据的更新并同步到GaussDB(DWS)。

表 2-12 样例数据

id	name	age
1	lily	16
2	lucy > jim	17
3	lilei > tom	15

约束限制

- 确保VPC、ECS、OBS、Kafka、DLI和DWS服务在同一个区域内，例如华北-北京四。
- 确保Kafka、DLI、DWS网络互通。本实践将Kafka和DWS创建在同一个区域和虚拟私有云下，同时在Kafka和DWS的安全组中放通了DLI的队列所在网段，确保网络互通。
- 为确保DLI到DWS的连接链路稳定，请创建完DWS集群后为集群绑定ELB服务。

准备工作

- 已注册华为账号并开通华为云，具体请参见**注册华为账号并开通华为云**，且在使用GaussDB(DWS) 前检查账号状态，账号不能处于欠费或冻结状态。
- 已创建虚拟私有云和子网，参见**创建虚拟私有云和子网**。

步骤一：创建 Kafka 实例

步骤1 登录华为云控制台，服务列表选择“应用中间件 > 分布式消息服务Kafka版”，进入Kafka管理控制台。

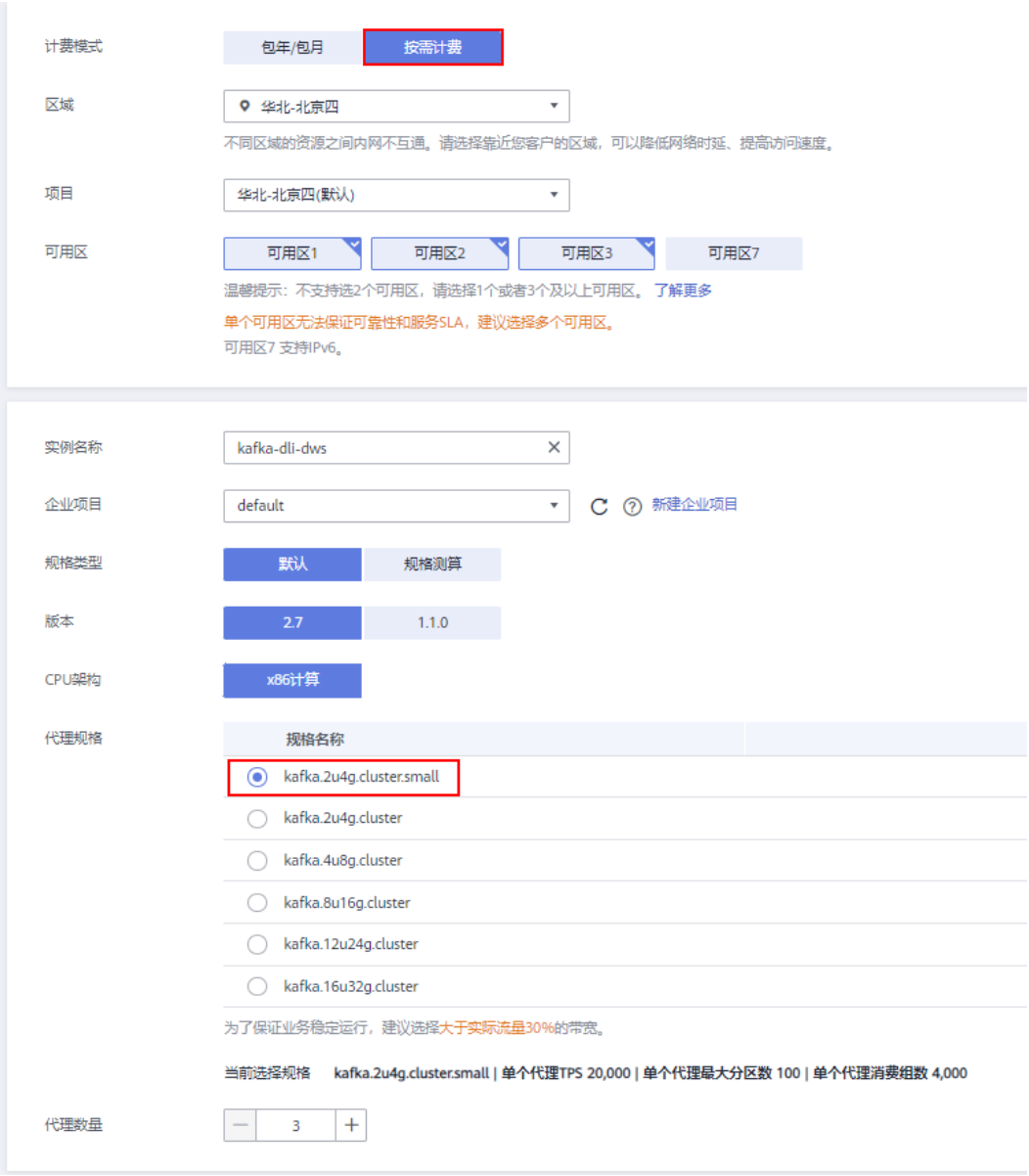
步骤2 左侧导航栏选择“Kafka专项版”，单击右上角的“购买kafka实例”。

步骤3 填写如下参数，其他参数项如表中未说明，默认即可：

表 2-13 kafka 实例参数

参数项	参数值
计费模式	按需计费
区域	华北-北京四
项目	默认
可用区	可用区1（如遇售罄，选择其他可用区）
实例名称	kafka-dli-dws
企业项目	default
规格类型	默认
版本	2.7
CPU架构	x86计算
代理规格	kafka.2u4g.cluster.small（实例仅为参考，选择最小规格即可）
代理数量	3
虚拟私有云	选择已创建的虚拟私有云，如果没有，则需要创建。
安全组	选择已创建的安全组，如果没有，则需要创建。
其他参数	保持默认

图 2-5 创建 Kafka 实例



步骤4 单击“立即创建”。等待创建成功。

步骤5 创建成功后，在Kafka实例列表中，单击创建好的Kafka实例名称，进入基本信息页面。

步骤6 左侧选择“Topic管理”，单击“创建Topic”。

Topic名称设置为“topic-demo”，其他可保持默认。

图 2-6 创建 Topic

步骤7 单击“确定”，在Topic列表中可以看到topic-demo已创建成功。

步骤8 左侧导航栏选择“消费组管理”，单击“创建消费组”。

步骤9 消费组名称输入“kafka01”，单击“确定”。

----结束

步骤二：创建绑定 ELB 的 DWS 集群和目标表

步骤1 **创建独享型弹性负载均衡服务ELB**，网络类型选择IPv4私网即可，区域、VPC选择与Kafka实例保持一致，本实践为“华北-北京四”。

步骤2 **创建集群**，为GaussDB(DWS)绑定弹性负载均衡 ELB，同时为确保网络连通，GaussDB(DWS)集群的区域、VPC选择与Kafka实例保持一致，本实践为“华北-北京四”，虚拟私有云与上面创建Kafka的虚拟私有云保持一致。

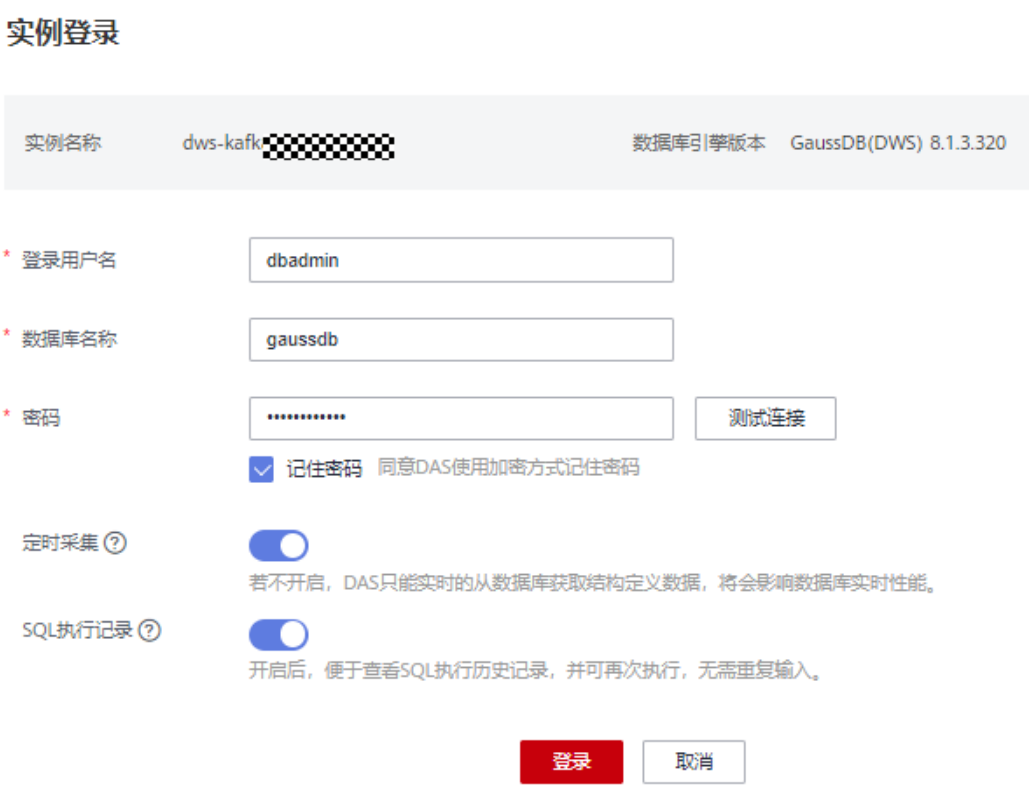
步骤3 在GaussDB(DWS)控制台的集群管理页面，单击指定集群所在行操作列的“登录”按钮。

说明

本实践以8.1.3.x版本为例，8.1.2及以前版本不支持此登录方式，可以**使用Data Studio连接集群**。

步骤4 登录用户名为dbadmin，数据库名称为gaussdb，密码为创建GaussDB(DWS)集群时设置的dbadmin用户密码，勾选“记住密码”，打开“定时采集”，“SQL执行记录”，单击“登录”。

图 2-7 登录 DWS



步骤5 单击“gaussdb”库名，再单击右上角的“SQL窗口”，进入SQL编辑器。

步骤6 复制如下SQL语句，在SQL窗口中，单击“执行SQL”，创建目标表user_dws。

```
CREATE TABLE user_dws (
id int,
name varchar(50),
age int,
PRIMARY KEY (id)
);
```

----结束

步骤三：创建 DLI 队列

步骤1 登录华为云控制台，服务列表选择“大数据 > 数据湖探索DLI”，进入DLI管理控制台。

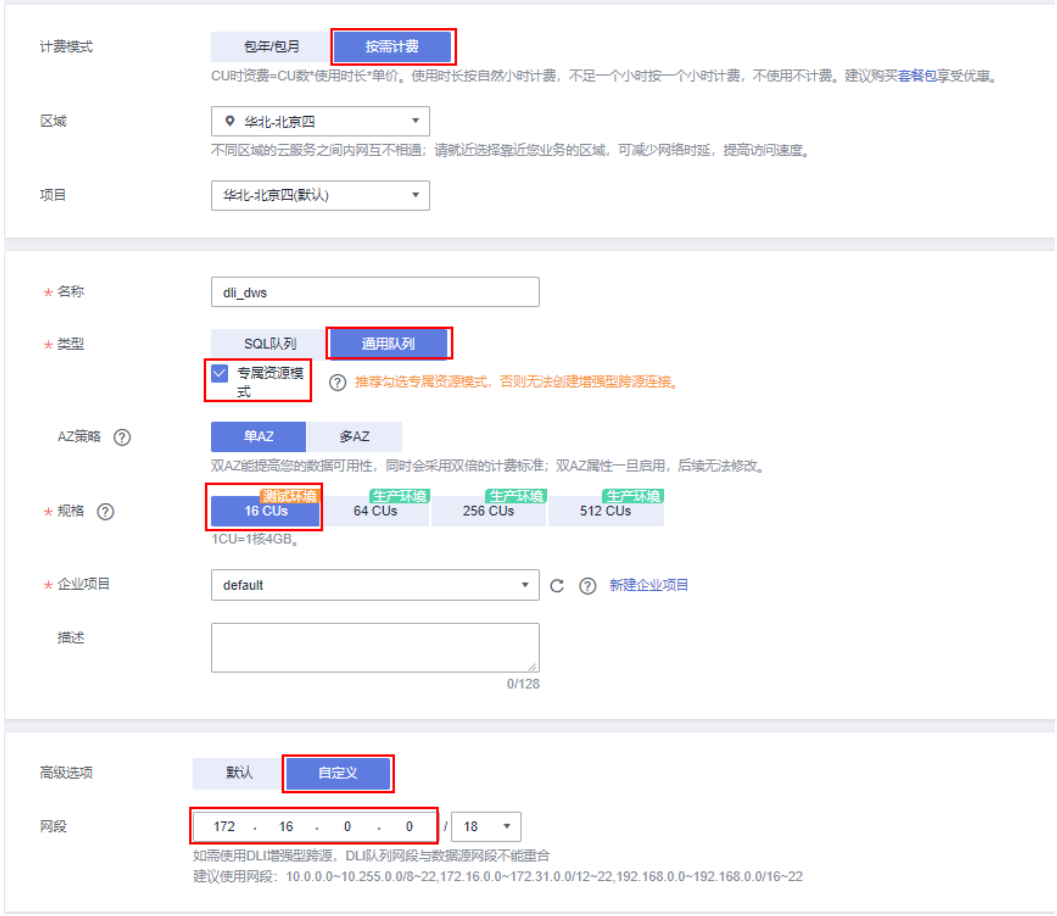
步骤2 左侧导航栏选择“资源管理 > 队列管理”，进入队列管理页面。

步骤3 单击右上角“购买队列”，填写如下参数，其他参数项如表中未说明，默认即可。

表 2-14 DLI 队列

参数项	参数值
计费模式	按需计费
区域	华北-北京四
项目	默认
名称	dli_dws
类型	通用队列，勾选“专属资源模式”。
AZ策略	单AZ
规格	16 CUs
企业项目	default
高级选项	自定义
网段	172.16.0.0/18，需选择与Kafka和DWS不在同一个网段。例如，如果Kafka和DWS在192.168.x.x网段，则DLI则选择172.16.x.x。

图 2-8 创建 DLI 队列



步骤4 单击“立即购买”。

----结束

步骤四：分别创建 Kafka 和 DWS 的增强型跨源连接

步骤1 放通Kafka的安全组，允许DLI队列所在的网段可以访问Kafka。

1. 回到Kafka控制台，单击Kafka实例名称进入基本信息。查看“连接信息”的“内网连接地址”，并记录下此地址，以备后续步骤使用。

图 2-9 kafka 内网连接地址

2. 单击网络的安全组名称。

图 2-10 kafka 安全组

3. 选择“入方向规则 > 添加规则”，如下图，添加DLI队列的网段地址，本实践为172.16.0.0/18，实际请与**步骤三：创建DLI队列**的时候填入的网段保持一致。

图 2-11 kafka 安全组添加规则

4. 单击“确定”。

步骤2 回到DLI管理控制台，单击左侧的“跨源管理”，选择“增强型跨源”，单击“创建”。

步骤3 填写如下参数，其他参数项如表中未说明，默认即可。

表 2-15 DLI 到 Kafka 的连接

参数项	参数值
连接名称	dli_kafka
弹性资源池	选择上面创建的DLI队列名称dli_dws。
虚拟私有云	选择Kafka所在的虚拟私有云。
子网	选择Kafka所在的子网。
其他参数	保持默认。

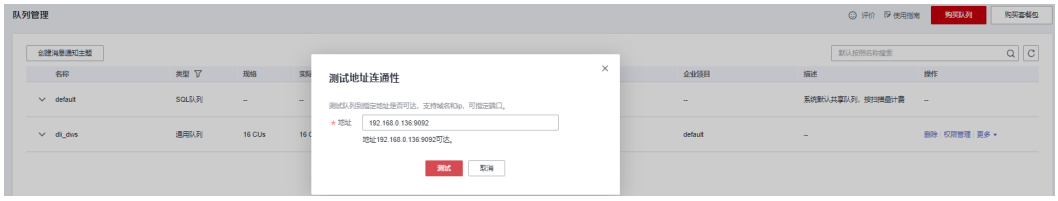
图 2-12 创建连接

步骤4 单击“确定”。等待Kafka连接创建成功。

步骤5 左侧导航栏选择“资源管理 > 队列管理”，选择dli_dws所在行操作列的“更多 > 测试地址连通性”。

步骤6 在地址栏中，输入**步骤1.1**获取的Kafka实例的内网IP和端口（Kafka的地址有三个，输入一个即可）。

图 2-13 测试 kafka 连通性

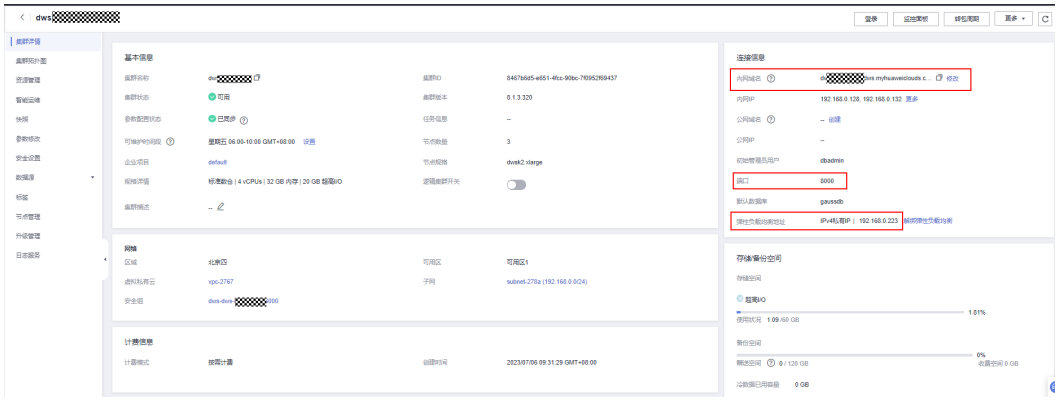


步骤7 单击“测试”，验证DLI连通Kafka成功。

步骤8 进入到DWS管理控制台，左侧导航栏单击“集群管理”，单击集群名称进入DWS集群详情。

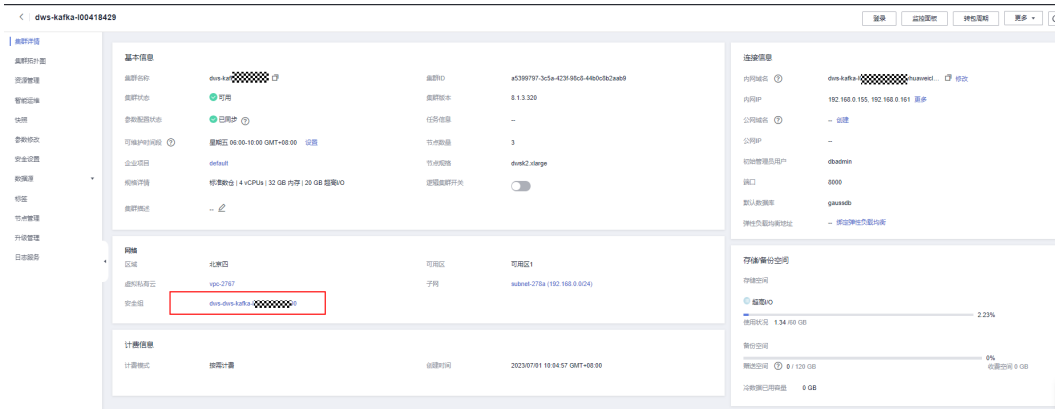
步骤9 如下图，记录下DWS集群的内网域名、端口和弹性负载均衡地址，以备后面步骤需要。

图 2-14 内网域名和 ELB 地址



步骤10 单击安全组名称。

图 2-15 DWS 安全组



步骤11 选择“入方向规则 > 添加规则”，如下图，添加DLI队列的网段地址，本实践为172.16.0.0/18，实际请与**步骤三：创建DLI队列**的时候填入的网段保持一致。

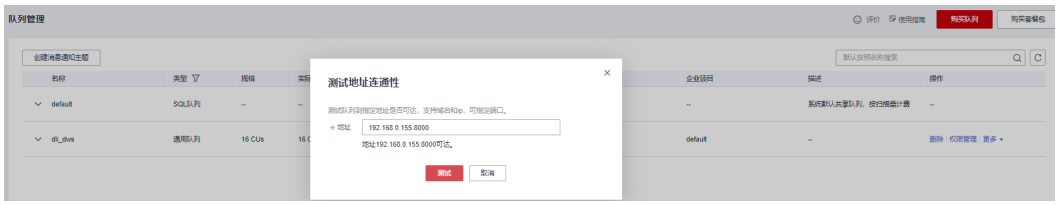
图 2-16 DWS 安全组添加规则

步骤12 单击“确定”。

步骤13 再切换到DLI控制台，左侧选择“资源管理 > 队列管理”，选择dli_dws所在行操作列的“更多 > 测试地址连通性”。

步骤14 在地址栏中，输入**步骤9**获取的GaussDB(DWS)集群的弹性负载均衡IP和端口。

图 2-17 测试 GaussDB(DWS)连通



步骤15 单击“测试”，验证DLI连通GaussDB(DWS)成功。

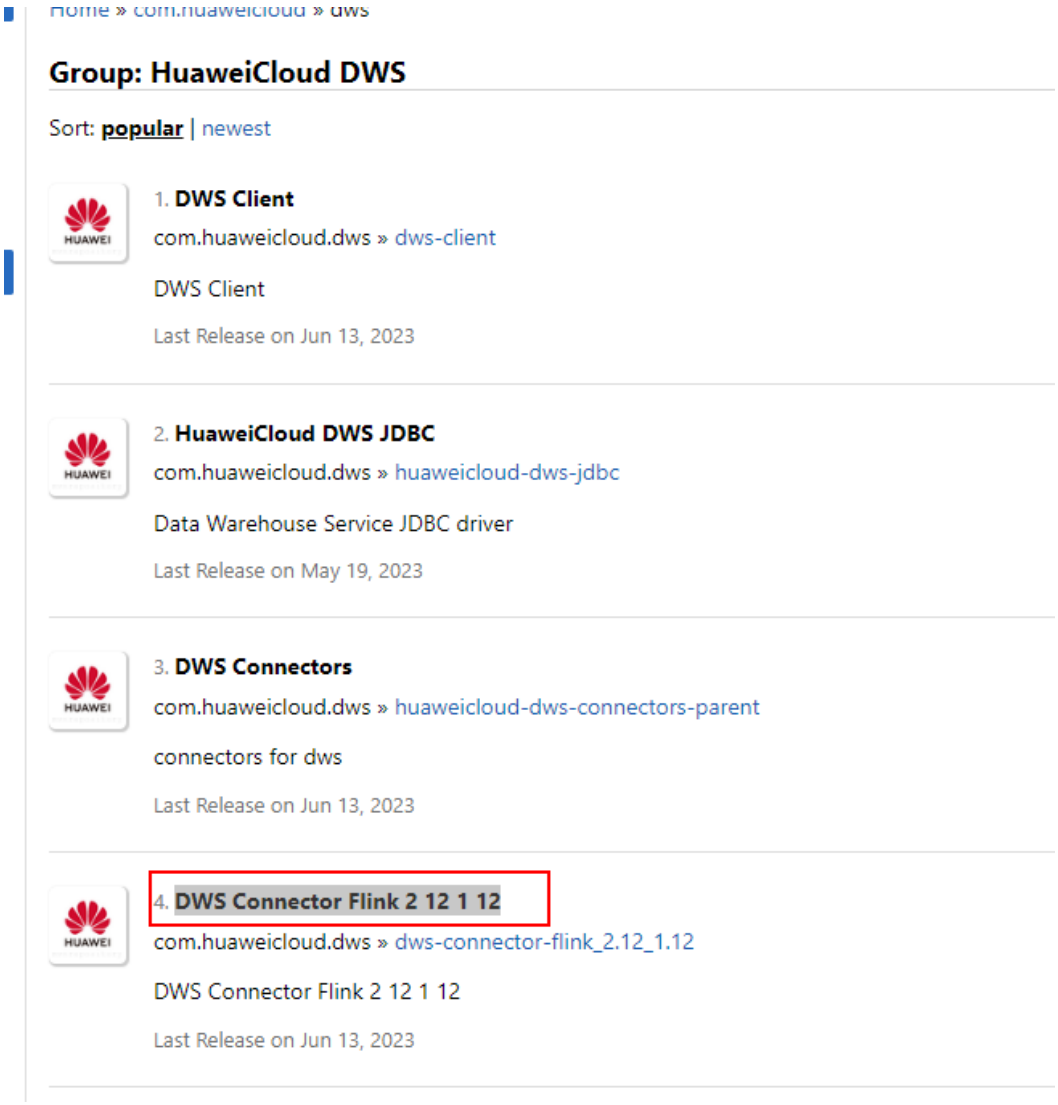
----结束

步骤五：准备 DWS 对接 Flink 工具 dws-connector-flink

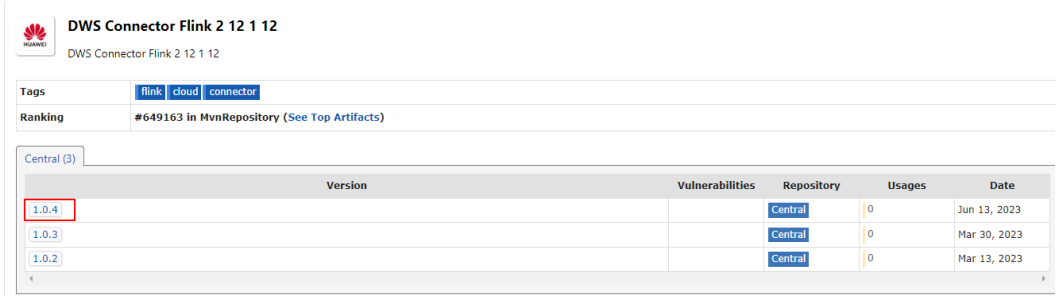
dws-connector-flink是一款基于DWS JDBC接口实现对接Flink的一个工具。在配置DLI作业阶段，将该工具及依赖放入Flink类加载目录，提升Flink作业入库DWS的能力。

步骤1 浏览器访问**https://mvnrepository.com/artifact/com.huaweicloud.dws**。

步骤2 在软件列表中选择最新版本的DWS Connectors Flink，本实践选择**DWS Connector Flink 2 12 1 12**。

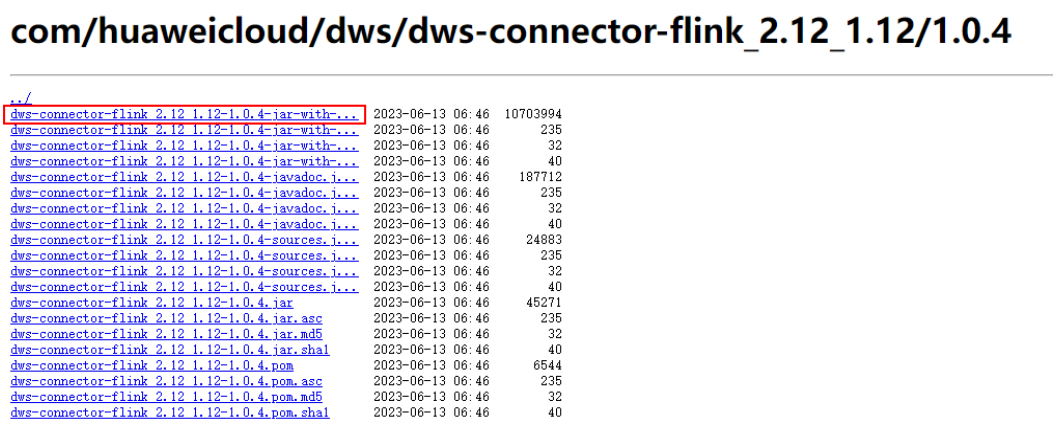


步骤3 单击“1.0.4”分支，实际请以官网发布的新分支为准。



步骤4 单击“View ALL”。

步骤5 单击dws-connector-flink_2.12_1.12-1.0.4-jar-with-dependencies.jar，下载到本地。



步骤6 创建OBS桶，本实践桶名设置为obs-flink-dws，并将此文件上传到OBS桶下，注意桶也保持与DLI在一个区域下，本实践为 华北-北京四。

图 2-18 上传 jar 包到 OBS 桶

----结束

步骤六：创建并编辑 DLI Flink 作业

步骤1 回到DLI管理控制台，左侧选择“作业管理 > Flink作业”，单击右上角“创建作业”。

步骤2 类型选择“Flink OpenSource SQL”，名称填写kafka-dws。

图 2-19 创建作业

步骤3 单击“确定”。系统自动进入到作业的编辑页面。

步骤4 在页面右侧填写如下参数，其他参数项如表中未说明，默认即可。

表 2-16 flink 作业参数

参数项	参数值
所属队列	dli_dws
Flink版本	1.12

参数项	参数值
UDF Jar	选择**步骤五：准备DWS对接Flink工具dws-connector-flink**的OBS桶中的jar文件。 
OBS桶	选择**步骤五：准备DWS对接Flink工具dws-connector-flink**的桶。
开启Checkpoint	勾选
其他参数	保持默认

图 2-20 编辑作业

步骤5 将以下符合Flink要求的SQL代码复制到左侧的SQL代码窗。

其中“Kafka实例内网IP地址和端口”参见**步骤1.1**获取，“DWS内网域名”由**步骤9**获取。

```
CREATE TABLE user_kafka (
  id string,
  name string,
  age int
) WITH (
```

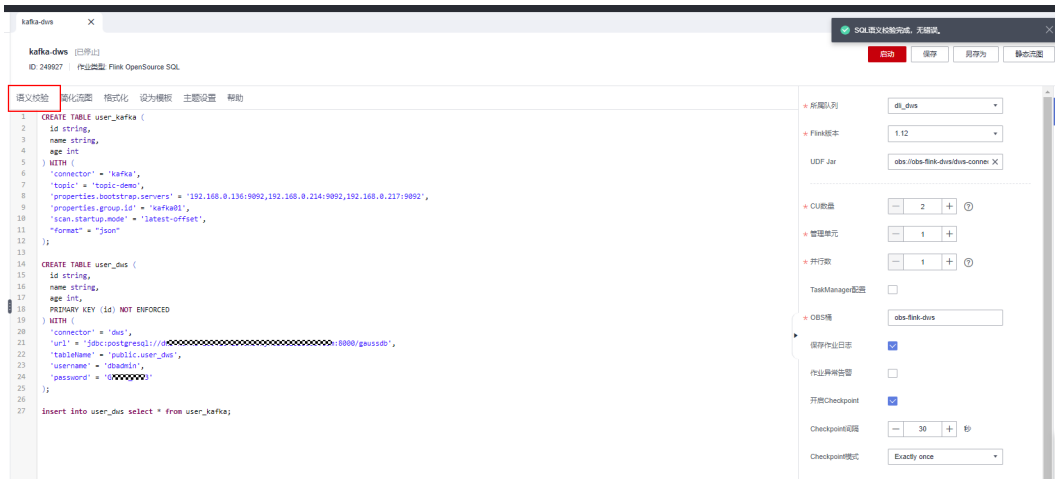


```
  'connector' = 'kafka',
  'topic' = 'topic-demo',
  'properties.bootstrap.servers' = 'Kafka实例内网IP地址和端口',
  'properties.group.id' = 'kafka01',
  'scan.startup.mode' = 'latest-offset',
  "format" = "json"
);

CREATE TABLE user_dws (
  id string,
  name string,
  age int,
  PRIMARY KEY (id) NOT ENFORCED
) WITH (
  'connector' = 'dws',
  'url' = 'jdbc:postgresql://DWS内网域名:8000/gaussdb',
  'tableName' = 'public.user_dws',
  'username' = 'dbadmin',
  'password' = '数据库用户dbdamin密码'
);

insert into user_dws select * from user_kafka;
```

步骤6 单击“语义校验”，等待校验成功。

如校验失败，则检查SQL的输入是否存在语法错误。

图 2-21 作业的 SQL 语句



步骤7 单击“保存”。

步骤8 回到DLI控制台首页，左侧选择“作业管理 > Flink作业”。

步骤9 单击作业名称kafka-dws右侧的“启动”，单击“立即启动”。

等待约1分钟，再刷新页面，状态在“运行中”表示作业成功运行。

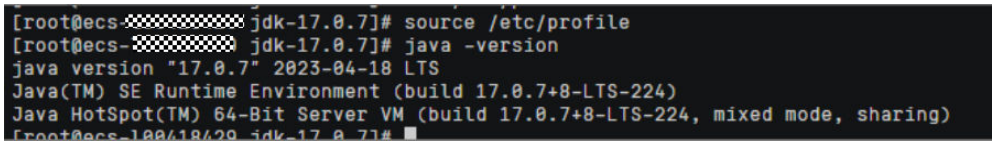
图 2-22 作业运行状态

----结束

步骤七：通过 Kafka 客户端生产和修改消息

步骤1 参见ECS文档创建一台ECS，具体创建步骤此处不再赘述。创建时，确保ECS的区域、虚拟私有云保持与Kafka一致。

步骤2 安装JDK。

1. 登录ECS，进入到/usr/local，下载JDK包。
   ```
   cd /usr/local
   wget https://download.oracle.com/java/17/latest/jdk-17_linux-x64_bin.tar.gz
   ```
2. 解压下载好的JDK包。
   ```
   tar -zxvf jdk-17_linux-x64_bin.tar.gz
   ```
3. 执行以下命令进入/etc/profile文件。
   ```
   vim /etc/profile
   ```
4. 按i进入编辑模式，将以下内容增加到/etc/profile文件的末尾。
   ```
   export JAVA_HOME=/usr/local/jdk-17.0.7  #jdk安装目录
   export JRE_HOME=${JAVA_HOME}/jre
   export CLASSPATH=.:${JAVA_HOME}/lib:${JRE_HOME}/lib:${JAVA_HOME}/test:${JAVA_HOME}/lib/gsjdbc4.jar:${JAVA_HOME}/lib/dt.jar:${JAVA_HOME}/lib/tools.jar:$CLASSPATH
   export JAVA_PATH=${JAVA_HOME}/bin:${JRE_HOME}/bin
   export PATH=$PATH:${JAVA_PATH}
   ```
   
5. 按ESC，输入:wq!按回车，保存退出。
6. 执行命令，使环境变量生效。
   ```
   source /etc/profile
   ```
7. 执行以下命令，提示如下信息表示jdk安装成功。
   ```
   java -version
   ```
   

步骤3 安装Kafka客户端。

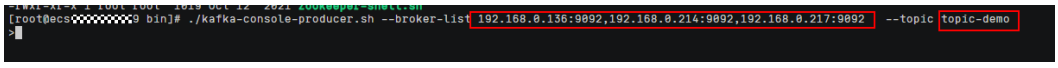
1. 进入/opt目录，执行以下命令获取Kafka客户端软件包。
   ```
   cd /opt
   wget https://archive.apache.org/dist/kafka/2.7.2/kafka_2.12-2.7.2.tgz
   ```
2. 解压下载好的软件包。
   ```
   tar -zxf kafka_2.12-2.7.2.tgz
   ```
3. 进入到Kafka客户端目录。
   ```
   cd /opt/kafka_2.12-2.7.2/bin
   ```

步骤4 执行以下命令连接Kafka。其中 {连接地址}为Kafka的内网连接地址，参见**步骤1.1**获取，topic为**步骤6**创建的Kafka的topic名称。

```
./kafka-console-producer.sh --broker-list {连接地址} --topic {Topic名称}
```

本实践示例如下：

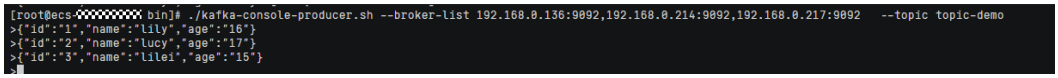
```
./kafka-console-producer.sh --broker-list
192.168.0.136:9092,192.168.0.214:9092,192.168.0.217:9092 --topic topic-demo
```



如上图出现>符号，无其他报错，表示连接成功。

步骤5 在已连接kafka的客户端窗口下，根据**场景描述**规划的数据，复制以下内容（注意一次复制一行），按回车发送，进行生产消息。

```
{"id":"1","name":"lily","age":"16"}
{"id":"2","name":"lucy","age":"17"}
{"id":"3","name":"lilei","age":"15"}
```



步骤6 回到DWS控制台，左侧选择“集群管理”，单击DWS集群右侧“登录”，进入SQL页面。

步骤7 执行以下SQL语句，发现数据实时入库成功。

```
SELECT * FROM user_dws ORDER BY id;
```

	id	name	age
1	1	lily	16
2	2	lucy	17
3	3	lilei	15

步骤8 继续回到ECS中连接Kafka的客户端窗口，复制以下内容（注意一次复制一行），按回车发送，进行生产消息。

```
{"id":"2","name":"jim","age":"17"}
{"id":"3","name":"tom","age":"15"}
```

步骤9 回到DWS已打开的SQL窗口，执行以下SQL语句，发现id为2和3的姓名已修改为jim和tom。

符合场景描述预期，本实践结束。

```
SELECT * FROM user_dws ORDER BY id;
```

	id	name	age
1	1	lily	16
2	2	jim	17
3	3	tom	15

----结束

2.4 基于 GDS 实现两套 DWS 集群间的数据互联互通实践

本实践演示基于GDS导入导出的高并发能力，实现两套DWS集群之间1500万行数据的分钟级迁移。

说明

- 该功能仅8.1.2及以上集群版本支持。
- GDS是GaussDB(DWS)自研的高并发导入导出工具，了解更多请参考**GDS使用说明**。
- 本章节仅演示操作实践，GDS互联互通介绍、语法说明请参见**基于GDS的跨集群互联互通**。

本实践预计时长90分钟，实践用到的云服务资源主要是**数据仓库服务 GaussDB(DWS)**、**弹性云服务 ECS**、**虚拟私有云服务 VPC**，基本流程如下：

1. **准备工作**
2. **步骤一：创建两套DWS集群**
3. **步骤二：准备源端数据**
4. **步骤三：安装并启动GDS服务器**

5. **步骤四：实现跨DWS集群的数据互联互通**

支持区域

当前已上传OBS数据的区域如**表2-17**所示。

表 2-17 区域和 OBS 桶名

区域	OBS桶名
华北-北京一	dws-demo-cn-north-1
华北-北京二	dws-demo-cn-north-2
华北-北京四	dws-demo-cn-north-4
华北-乌兰察布一	dws-demo-cn-north-9
华东-上海一	dws-demo-cn-east-3
华东-上海二	dws-demo-cn-east-2
华南-广州	dws-demo-cn-south-1
华南-广州友好	dws-demo-cn-south-4
中国-香港	dws-demo-ap-southeast-1
亚太-新加坡	dws-demo-ap-southeast-3
亚太-曼谷	dws-demo-ap-southeast-2
拉美-圣地亚哥	dws-demo-la-south-2
非洲-约翰内斯堡	dws-demo-af-south-1
拉美-墨西哥城一	dws-demo-na-mexico-1
拉美-墨西哥城二	dws-demo-la-north-2
莫斯科二	dws-demo-ru-northwest-2
拉美-圣保罗一	dws-demo-sa-brazil-1

约束限制

本实践中两套DWS、ECS服务在同一个区域和虚拟私有云VPC下，确保网络互通。

准备工作

- 获取此账号的“AK/SK”。
- 已创建虚拟私有云和子网，参见**创建虚拟私有云和子网**。

步骤一：创建两套 DWS 集群

参见**创建集群**创建两套DWS集群，建议创建在华北-北京四区域。两套集群名称分别为dws-demo01和dws-demo02。

步骤二：准备源端数据

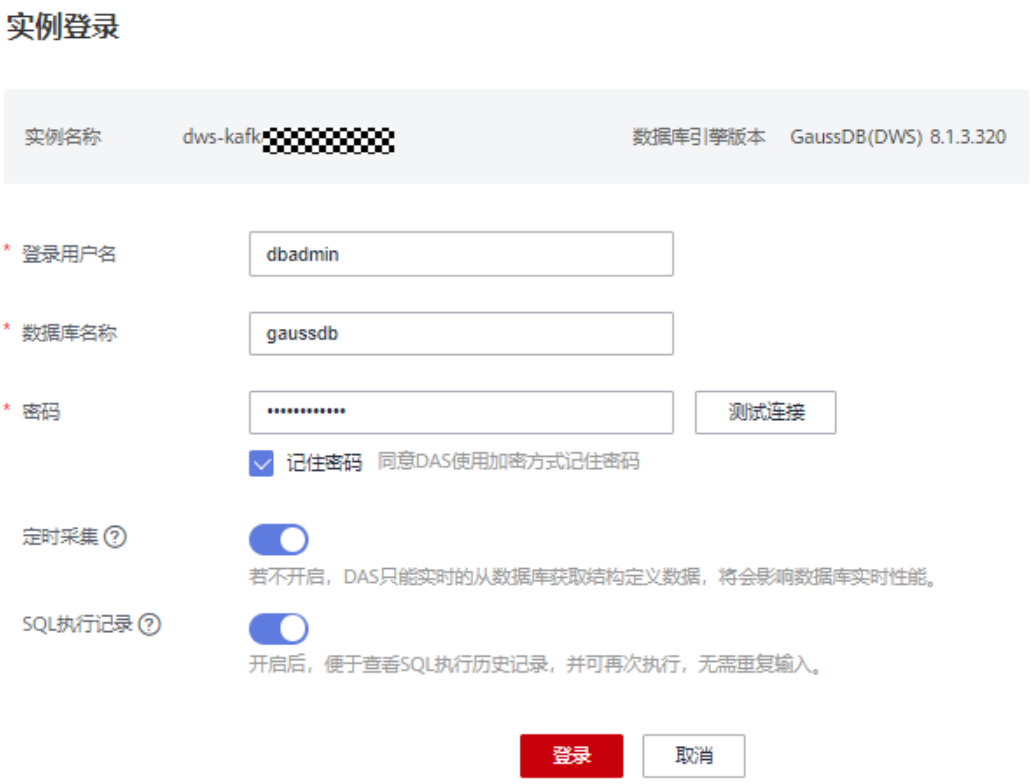
步骤1 在GaussDB(DWS)控制台的“集群管理”，单击源集群dws-demo01所在行操作列的“登录”按钮。

说明

本实践以8.1.3.x版本为例，8.1.2及以前版本不支持此登录方式，可以**使用Data Studio连接集群**。

步骤2 登录用户名为dbadmin，数据库名称为gaussdb，密码为创建GaussDB(DWS)集群时设置的dbadmin用户密码，勾选“记住密码”，打开“定时采集”，“SQL执行记录”，单击“登录”。

图 2-23 登录 DWS



步骤3 单击“gaussdb”库名，再单击右上角的“SQL窗口”，进入SQL编辑器。

步骤4 复制如下SQL语句到SQL窗口中，单击“执行SQL”，创建测试TPC-H表ORDERS。

```
CREATE TABLE ORDERS
(
O_ORDERKEY BIGINT NOT NULL ,
O_CUSTKEY BIGINT NOT NULL ,
O_ORDERSTATUS CHAR(1) NOT NULL ,
O_TOTALPRICE DECIMAL(15,2) NOT NULL ,
O_ORDERDATE DATE NOT NULL ,
```

```
O_ORDERPRIORITY CHAR(15) NOT NULL ,
O_CLERK CHAR(15) NOT NULL ,
O_SHIPPRIORITY BIGINT NOT NULL ,
O_COMMENT VARCHAR(79) NOT NULL)
with (orientation = column)
distribute by hash(O_ORDERKEY)
PARTITION BY RANGE(O_ORDERDATE)
(
PARTITION O_ORDERDATE_1 VALUES LESS THAN('1993-01-01 00:00:00'),
PARTITION O_ORDERDATE_2 VALUES LESS THAN('1994-01-01 00:00:00'),
PARTITION O_ORDERDATE_3 VALUES LESS THAN('1995-01-01 00:00:00'),
PARTITION O_ORDERDATE_4 VALUES LESS THAN('1996-01-01 00:00:00'),
PARTITION O_ORDERDATE_5 VALUES LESS THAN('1997-01-01 00:00:00'),
PARTITION O_ORDERDATE_6 VALUES LESS THAN('1998-01-01 00:00:00'),
PARTITION O_ORDERDATE_7 VALUES LESS THAN('1999-01-01 00:00:00')
);
```

步骤5 继续执行以下SQL语句，创建OBS外表。

其中AK值、SK值替换成实际账号的AK、SK值。*<obs_bucket_name>*由*支持区域*获取。

说明

认证用的AK和SK硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全。

```
CREATE FOREIGN TABLE ORDERS01
 (
LIKE orders
 )
 SERVER gsmpp_server
 OPTIONS (
 ENCODING 'utf8',
 LOCATION obs://<obs_bucket_name>/tpch/orders.tbl',
 FORMAT 'text',
 DELIMITER '|',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
 CHUNKSIZE '64',
 IGNORE_EXTRA_DATA 'on'
 );
```

步骤6 执行以下SQL，从OBS外表导入数据到源DWS集群。导入时间预计2分钟，请等待。

说明

如果出现导入错误，则是因为上面的外表AK值、SK值有误导致，请执行DROP FOREIGN TABLE order01;删除外表后，重新创建外表，再重试执行以下语句导入数据。

```
INSERT INTO orders SELECT * FROM orders01;
```

步骤7 按以上方法，登录目标端集群dws-demo02，执行以下SQL，创建目标表orders。

```
CREATE TABLE ORDERS
 (
 O_ORDERKEY BIGINT NOT NULL ,
 O_CUSTKEY BIGINT NOT NULL ,
 O_ORDERSTATUS CHAR(1) NOT NULL ,
 O_TOTALPRICE DECIMAL(15,2) NOT NULL ,
 O_ORDERDATE DATE NOT NULL ,
 O_ORDERPRIORITY CHAR(15) NOT NULL ,
 O_CLERK CHAR(15) NOT NULL ,
 O_SHIPPRIORITY BIGINT NOT NULL ,
 O_COMMENT VARCHAR(79) NOT NULL)
 with (orientation = column)
 distribute by hash(O_ORDERKEY)
 PARTITION BY RANGE(O_ORDERDATE)
 (
 PARTITION O_ORDERDATE_1 VALUES LESS THAN('1993-01-01 00:00:00'),
```



```
PARTITION O_ORDERDATE_2 VALUES LESS THAN('1994-01-01 00:00:00'),
PARTITION O_ORDERDATE_3 VALUES LESS THAN('1995-01-01 00:00:00'),
PARTITION O_ORDERDATE_4 VALUES LESS THAN('1996-01-01 00:00:00'),
PARTITION O_ORDERDATE_5 VALUES LESS THAN('1997-01-01 00:00:00'),
PARTITION O_ORDERDATE_6 VALUES LESS THAN('1998-01-01 00:00:00'),
PARTITION O_ORDERDATE_7 VALUES LESS THAN('1999-01-01 00:00:00')
);
```

----结束

步骤三：安装并启动 GDS 服务器

步骤1 参见弹性云服务的**购买弹性云服务器**创建弹性云服务器，注意ECS与DWS创建在同一个区域、VPC内，本例ECS镜像选择CentOS 7.6版本。

步骤2 下载GDS工具包。

1. 登录GaussDB(DWS)管理控制台。
2. 在左侧导航栏中，单击“连接管理”。
3. 在“命令行客户端”的下拉列表中，选择对应版本的GDS客户端。
   请根据集群版本和安装客户端的操作系统，选择对应版本。

   说明

   客户端CPU架构要和集群一致，如果集群是X86规格，则也应该选择X86客户端。
4. 单击“下载”。

步骤3 使用SFTP工具将下载的客户端（本实验以**dws_client_8.2.x_redhat_x64.zip**为例）上传至ECS的/opt目录下。

步骤4 使用root用户登录ECS执行以下命令，进入/opt目录，解压客户端。

```
cd /opt
unzip dws_client_8.2.x_redhat_x64.zip
```

步骤5 创建GDS专有用户及所属用户组，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/gds/bin
chown -R gds_user:gdsgrp /opt
```

步骤7 切换到gds用户。

```
su - gds_user
```

步骤8 执行以下命令进入gds目录，并执行环境变量。

```
cd /opt/gds/bin
source gds_env
```

步骤9 执行以下命令启动gds，其中ECS内网IP通过ECS控制台查看如下。

```
/opt/gds/bin/gds -d /opt -p ECS内网IP:5000 -H 0.0.0.0/0 -l /opt/gds/bin/gds_log.txt -D -t 2
```

步骤10 放通ECS到DWS之间的网络端口。

由于GDS服务器（即本实验ECS）与DWS需要建立通讯，ECS默认的安全组入方向并没有放通GDS端口5000和DWS端口8000，需执行以下步骤：

1. 回到弹性云服务器ECS的控制台，单击云服务器名称进入详情。
2. 切换到“安全组”页签，单击“配置规则”。
3. 选择“入方向规则”，单击“添加规则”，优先级输入1，协议端口输入5000，单击“确认”。

4. 按以上方式，另外添加一行8000的入方向规则。

----结束

步骤四：实现跨 DWS 集群的数据互联互通

步骤1 创建server。

1. 获取源端DWS集群的内网IP：切换到DWS控制台，左侧选择“集群管理”，单击源端集群名称dws-demo01。
2. 进入集群详情名称，记录DWS的内网IP。

3. 切回到DWS控制台，单击目标端dws-demo02所在行操作列的“登录”按钮，进入到SQL窗口，

执行以下命令创建server。

其中，*源端DWS集群内网IP*由上步获取，*ECS服务器内网IP*从ECS控制台获取，*用户dbadmin的登录密码*在创建DWS集群时设定。

```
CREATE SERVER server_remote FOREIGN DATA WRAPPER GC_FDW OPTIONS
 (
 address '源端DWS集群内网IP.8000',
 dbname 'gaussdb',
 username 'dbadmin',
 password '用户dbadmin的登录密码',
 syncsrv 'gsfs://ECS服务器内网IP.5000'
 )
 ;
```

步骤2 创建互联互通外表。

继续在目标端集群dws-demo02的SQL窗口执行以下命令创建互联互通外表。

```
CREATE FOREIGN TABLE ft_orders
 (
 O_ORDERKEY BIGINT ,
 O_CUSTKEY BIGINT ,
 O_ORDERSTATUS CHAR(1) ,
 O_TOTALPRICE DECIMAL(15,2) ,
 O_ORDERDATE DATE ,
 O_ORDERPRIORITY CHAR(15) ,
 O_CLERK CHAR(15) ,
 O_SHIPPRIORITY BIGINT ,
 O_COMMENT VARCHAR(79)

 )
 SERVER server_remote
 OPTIONS
 (
 schema_name 'public',
 table_name 'orders',
 encoding 'SQL_ASCII'
 );
```

步骤3 执行全表数据导入。

在SQL窗口中继续执行以下SQL语句，从ft_orders外表中导入全量数据。等待约1分钟导入。

```
INSERT INTO orders SELECT * FROM ft_orders;
```

执行以下SQL查询，发现导入1500万行数据成功。

```
SELECT count(*) FROM orders;
```

步骤4 按过滤条件执行数据导入。

执行以下SQL，按过滤条件进行数据导入。

```
INSERT INTO orders SELECT * FROM ft_orders WHERE o_orderkey < '10000000';
```

----结束

3 调优表实践

3.1 表结构设计

在进行调优表实践之前，需要先了解下表结构设计相关的内容。因为进行数据库设计时，表设计上的一些关键项将严重影响后续整库的查询性能。表设计对数据存储也有影响：好的表设计能够减少I/O操作及最小化内存使用，进而提升查询性能。

本小节介绍如何设计GaussDB(DWS)表结构（包括选择表存储方式、压缩级别、分布方式、分布列以及使用分区表和局部聚簇等），从而实现表性能的优化。

选择存储方式

表的存储模型选择是表定义的第一步。业务属性是表的存储模型的决定性因素，根据下表选择适合当前业务的存储模型。

一般情况下，如果表的字段比较多（大宽表），查询中涉及到的列不多的情况下，适合列存储。如果表的字段个数比较少，查询大部分字段，那么选择行存储比较好。

存储模型	适用场景
行存	点查询(返回记录少，基于索引的简单查询)。 增删改比较多的场景。
列存	统计分析类查询 (group，join多的场景)。

表的行/列存储通过表定义的orientation属性定义。当指定orientation属性为row时，表为行存储；当指定orientation属性为column时，表为列存储；如果不指定，默认为行存储。

使用表压缩

表压缩可以在创建表时开启，压缩表能够使表中的数据以压缩格式存储，意味着占用相对少的内存。

对于I/O读写量大，CPU富足（计算相对小）的场景，选择高压缩比；反之选择低压缩比。建议依据此原则进行不同压缩下的测试和对比，以选择符合自身业务情况的最优压缩比。压缩比通过COMPRESSION参数指定，其支持的取值如下：

- 列存表为：YES/NO/LOW/MIDDLE/HIGH，默认值为LOW。
- 行存表为：YES/NO，默认值为NO。（行存表压缩功能暂未商用，如需使用请联系技术支持工程师）

各压缩级别所适用的业务场景说明如下：

压缩级别	所适用的业务场景
低级别压缩	系统CPU使用率高，存储磁盘空间充足。
中度压缩	系统CPU使用率适中，但存储磁盘空间不是特别充足。
高级别压缩	系统CPU使用率低，磁盘空间不充裕。

选择分布方式

GaussDB(DWS)支持的分布方式有复制表（Replication）、哈希表（Hash）和轮询表（Roundrobin）。

说明

轮询表（Roundrobin）8.1.2及以上集群版支持。

策略	描述	适用场景	优势与劣势
复制表（Replication）	集群中每一个DN实例上都有一份全量表数据。	小表、维度表。	• Replication优点是每个DN上都有此表的全量数据，在join操作中可以避免数据重分布操作，从而减小网络开销，同时减少了plan segment（每个plan segment都会起对应的线程）。 • Replication缺点是每个DN都保留了表的完整数据，造成数据的冗余。一般情况下只有较小的维度表才会定义为Replication表。
哈希表（Hash）	表数据通过hash方式散列到集群中的所有DN实例上。	数据量较大的事实表。	• 在读/写数据时可以利用各个节点的IO资源，大大提升表的读/写速度。 • 一般情况下大表（1000000条记录以上）定义为Hash表。

策略	描述	适用场景	优势与劣势
轮询表（Roundrobin）	表的每一行被轮番地发送给各个DN，数据会被均匀地分布在各个DN中。	数据量较大的事实表，且使用Hash分布时找不到合适的分布列。	• Roundrobin优点是保证了数据不会发生倾斜，从而提高了集群的空间利用率。 • Roundrobin缺点是无法像Hash表一样进行DN本地化优化，查询性能通常不如Hash表。 • 一般在大表无法找到合适的分布列时，定义为Roundrobin表，若大表能够找到合适的分布列，优先选择性能更好的Hash分布。

选择分布列

采用Hash分布方式，需要为用户表指定一个分布列（distribute key）。当插入一条记录时，系统会根据分布列的值进行hash运算后，将数据存储在对应的DN中。

所以Hash分布列选取至关重要，需要满足以下原则：

1. **列值应比较离散，以便数据能够均匀分布到各个DN**。例如，考虑选择表的主键为分布列，如在人员信息表中选择身份证号码为分布列。
2. **在满足第一条原则的情况下尽量不要选取存在常量filter的列**。例如，表dwcjk相关的部分查询中出现dwcjk的列zqdh存在常量的约束(例如zqdh=’000001’)，那么就应当尽量不用zqdh做分布列。
3. **在满足前两条原则的情况，考虑选择查询中的连接条件为分布列**，以便Join任务能够下推到DN中执行，且减少DN之间的通信数据量。

   对于Hash分表策略，如果分布列选择不当，可能导致数据倾斜，查询时出现部分DN的I/O短板，从而影响整体查询性能。因此在采用Hash分表策略之后需对表的数据进行数据倾斜性检查，以确保数据在各个DN上是均匀分布的。可以使用以下SQL检查数据倾斜性：

   ```
   SELECT
   xc_node_id, count(1)
   FROM tablename
   GROUP BY xc_node_id
   ORDER BY xc_node_id desc;
   ```

   其中xc_node_id对应DN，一般来说，**不同DN的数据量相差5%以上即可视为倾斜，如果相差10%以上就必须要调整分布列**。
4. 一般不建议用户新增一列专门用作分布列，尤其不建议用户新增一列，然后用SEQUENCE的值来填充作为分布列。因为SEQUENCE可能会带来性能瓶颈和不必要的维护成本。

使用分区表

分区表是把逻辑上的一张表根据某种方案分成几张物理块进行存储。这张逻辑上的表称之为分区表，物理块称之为分区。分区表是一张逻辑表，不存储数据，数据实际是存储在分区上的。分区表和普通表相比具有以下优点：

1. 改善查询性能：对分区对象的查询可以仅搜索自己关心的分区，提高检索效率。
2. 增强可用性：如果分区表的某个分区出现故障，表在其他分区的数据仍然可用。
3. 方便维护：如果分区表的某个分区出现故障，需要修复数据，只修复该分区即可。

GaussDB(DWS)支持的分区表为范围分区表和列表分区（列表分区8.1.3集群版本支持）。

使用局部聚簇

局部聚簇（Partial Cluster Key）是列存下的一种技术。这种技术可以通过min/max稀疏索引较快的实现基表扫描的filter过滤。Partial Cluster Key可以指定多列，但是一般不建议超过2列。Partial Cluster Key的选取原则：

1. 受基表中的简单表达式约束。这种约束一般形如col op const，其中col为列名，op为操作符 =、>、>=、<=、<，const为常量值。
2. 尽量采用选择度比较高（过滤掉更多数据）的简单表达式中的列。
3. 尽量把选择度比较低的约束col放在Partial Cluster Key中的前面。
4. 尽量把枚举类型的列放在Partial Cluster Key中的前面。

选择数据类型

高效数据类型，主要包括以下三方面：

1. **尽量使用执行效率比较高的数据类型**

   一般来说整型数据运算（包括=、＞、＜、≧、≦、≠等常规的比较运算，以及group by）的效率比字符串、浮点数要高。比如某客户场景中对列存表进行点查询，filter条件在一个numeric列上，执行时间为10+s；修改numeric为int类型之后，执行时间缩短为1.8s左右。

2. **尽量使用短字段的数据类型**

   长度较短的数据类型不仅可以减小数据文件的大小，提升IO性能；同时也可以减小相关计算时的内存消耗，提升计算性能。比如对于整型数据，如果可以用smallint就尽量不用int，如果可以用int就尽量不用bigint。

3. **使用一致的数据类型**

   表关联列尽量使用相同的数据类型。如果表关联列数据类型不同，数据库必须动态地转化为相同的数据类型进行比较，这种转换会带来一定的性能开销。

使用索引

- 建立索引的目的是为了加速查询，所以请确保索引能在一些查询中被使用。如果一个索引不会被任何查询语句用到，那这个索引是没有意义的，请删除这个索引。
- 避免创建不需要的二级索引，有用的二级索引能加速查询，但是要注意索引占用空间也会随这索引数量的增加而增加。每增加一个索引，在插入一条数据的时候，就要额外新增一个 Key-Value，所以索引越多，写入越慢，并且空间占用越大。另外过多的索引也会影响优化器运行时间，并且不合适的索引会误导优化器。所以索引并不是越多越好。
- 根据具体的业务特点创建合适的索引。原则上需要对查询中需要用到的列创建索引，目的是提高性能。下面几种情况适合创建索引：
  - 区分度比较大的列，通过索引能显著地减少过滤后的行数。例如推荐在人的身份证号码这一列上创建索引，但不推荐在人的性别这一列上创建索引。

  - 有多个查询条件时，可以选择组合索引，注意需要把等值条件的列放在组合索引的前面。这里举一个例子，假设常用的查询是 SELECT * FROM t where c1 = 10 and c2 = 100 and c3 > 10，那么可以考虑建立组合索引 Index cidx (c1, c2, c3)，这样可以用查询条件构造出一个索引前缀进行 Scan。
- 在查询条件中使用索引列作为条件时，不要在索引列上做计算、函数或者类型转换的操作，会导致优化器无法使用该索引。
- 尽量使索引列包含查询列，避免总是 SELECT * 查询所有列的语句。
- 查询条件使用 !=，NOT IN 时，无法使用索引。
- 使用LIKE时如果条件是以通配符 % 开头，也无法使用索引。
- 当查询条件有多个索引可供使用，但用户知道用哪一个索引是最优的时，推荐使用优化器Hint来强制优化器使用这个索引，这样可以避免优化器因为统计信息不准或其他问题时，选错索引。
- 查询条件使用IN表达式时，后面匹配的条件数量不宜过多，否则执行效率会较差。

3.2 调优表概述

在本实践中，您将学习如何优化表的设计。您首先不指定存储方式，分布键、分布方式和压缩方式创建表，然后为这些表加载测试数据并测试系统性能。接下来，您将应用优秀实践以使用新的存储方式、分布键、分布方式和压缩方式重新创建这些表，并再次为这些表加载测试数据和测试系统性能，以便比较不同的设计对表的加载性能、存储空间和查询性能的影响。

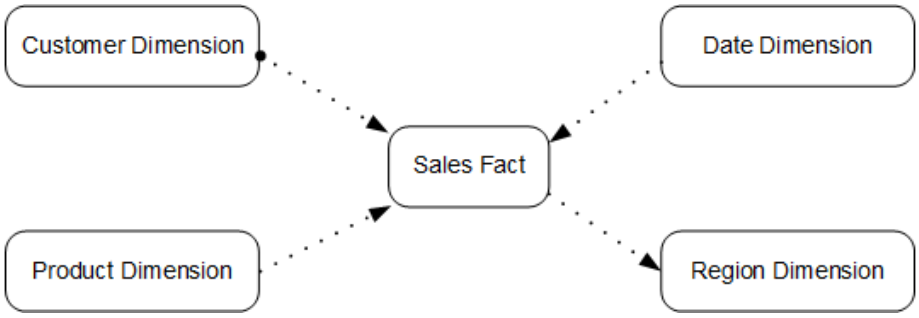
估计时间：60 分钟。

3.3 选择表模型

在设计数据仓库模型的时候，最常见的有两种：星型模型与雪花模型。选择哪一种模型需要根据业务需求以及性能的多重考量来定。

- **星型模型**由包含数据库核心数据的中央事实数据表和为事实数据表提供描述性属性信息的多个维度表组成。维度表通过主键关联事实表中的外键。如图3-1。
  - 所有的事实都必须保持同一个粒度。
  - 不同的维度之间没有任何关联。

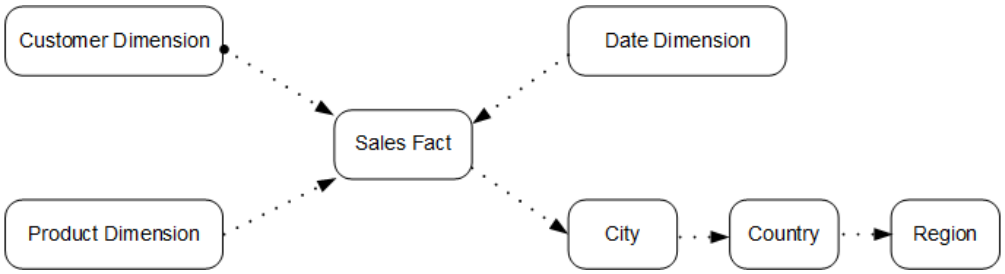
图 3-1 星型模型



- **雪花模型**是在基于星型模型之上拓展来的，每一个维度可以再扩散出更多的维度，根据维度的层级拆分成颗粒度不同的多张表。如图3-2。

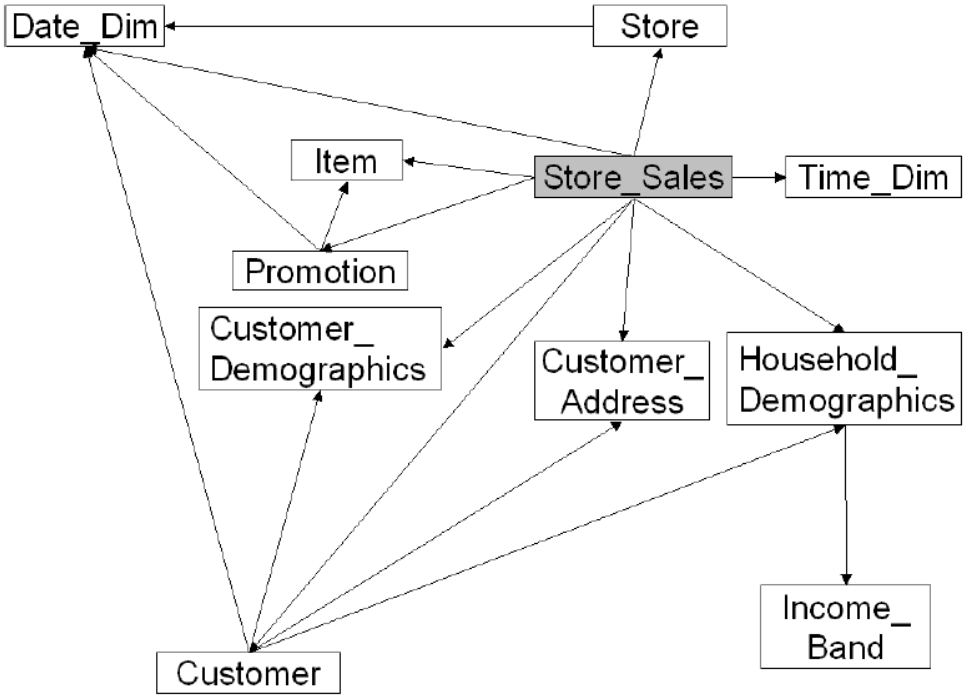
- 优点是减少维度表的数据量，各个维度表之间按需关联。
- 缺点是需要额外维护维度表的数量。

图 3-2 雪花模型



本实践基于TPC-DS的SS（Store Sales）模型做验证。该模型为雪花模型，图3-3显示了该数据模型的结构。

图 3-3 TPC-DS Store Sales ER-Diagram



有关该模型中事实表Store_Sales及各维度表的信息，请查阅TPC-DS官方文档：**http://www.tpc.org/tpc_documents_current_versions/current_specifications5.asp**。

3.4 步骤 1：创建初始表并加装样例数据

支持区域

当前已上传OBS数据的区域如表3-1所示。

表 3-1 区域和 OBS 桶名

区域	OBS桶名
华北-北京一	dws-demo-cn-north-1
华北-北京二	dws-demo-cn-north-2
华北-北京四	dws-demo-cn-north-4
华北-乌兰察布一	dws-demo-cn-north-9
华东-上海一	dws-demo-cn-east-3
华东-上海二	dws-demo-cn-east-2
华南-广州	dws-demo-cn-south-1
华南-广州友好	dws-demo-cn-south-4
中国-香港	dws-demo-ap-southeast-1
亚太-新加坡	dws-demo-ap-southeast-3
亚太-曼谷	dws-demo-ap-southeast-2
拉美-圣地亚哥	dws-demo-la-south-2
非洲-约翰内斯堡	dws-demo-af-south-1
拉美-墨西哥城一	dws-demo-na-mexico-1
拉美-墨西哥城二	dws-demo-la-north-2
莫斯科二	dws-demo-ru-northwest-2
拉美-圣保罗一	dws-demo-sa-brazil-1

创建一组不设置存储方式，无分布键、分布方式和压缩方式的表。然后，为这些表加载样例数据。

步骤1 （可选）创建集群。

如果已经有可供使用的集群，则可跳过这一步。创建集群的操作，请按**创建DWS 2.0集群**中的步骤操作。

同时请参考**连接集群的方式**中的方法连接到集群并测试连接。

本实践所使用的是8节点集群。也可以使用4节点集群进行测试。

步骤2 使用最少的属性创建SS（Store_Sales）测试表。

说明

如果SS表在当前数据库中已存在，需要先删除这些表。删除表使用DROP TABLE命令。如下示例表示删除表store_sales。

```
DROP TABLE store_sales;
```

考虑到本实践的目的，首次创建表时，没有设置存储方式、分布键、分布方式和压缩方式。

执行CREATE TABLE命令创建图3-3中的11张表。限于篇幅，这里仅附store_sales的创建语法。请从附录初始表创建中拷贝所有建表语法进行创建。

```
CREATE TABLE store_sales
(
    ss_sold_date_sk           integer                       ,
    ss_sold_time_sk           integer                       ,
    ss_item_sk                integer               not null,
    ss_customer_sk            integer                       ,
    ss_cdemo_sk               integer                       ,
    ss_hdemo_sk               integer                       ,
    ss_addr_sk                integer                       ,
    ss_store_sk               integer                       ,
    ss_promo_sk               integer                       ,
    ss_ticket_number          bigint                not null,
    ss_quantity               integer                       ,
    ss_wholesale_cost         decimal(7,2)                  ,
    ss_list_price             decimal(7,2)                  ,
    ss_sales_price            decimal(7,2)                  ,
    ss_ext_discount_amt       decimal(7,2)                  ,
    ss_ext_sales_price        decimal(7,2)                  ,
    ss_ext_wholesale_cost     decimal(7,2)                  ,
    ss_ext_list_price         decimal(7,2)                  ,
    ss_ext_tax                decimal(7,2)                  ,
    ss_coupon_amt             decimal(7,2)                  ,
    ss_net_paid               decimal(7,2)                  ,
    ss_net_paid_inc_tax       decimal(7,2)                  ,
    ss_net_profit             decimal(7,2)
) ;
```

步骤3 为这些表加载样例数据。

OBS存储桶中提供了本次实践的样例数据。该存储桶向所有经过身份验证的云用户提供了读取权限。请按照下面的步骤加载这些样例数据：

1. 为每个表创建对应的外表。

   GaussDB(DWS)应用Postgres提供的外部数据封装器FDW（Foreign Data Wrapper）进行数据并行导入。因此需要先创建FDW表，又称外表。限于篇幅，此处仅给出“store_sales”表对应的外表“obs_from_store_sales_001”的创建语法。请从附录外表创建拷贝其他外表的语法进行创建。

   说明

   - 注意，以下语句中的*<obs_bucket_name>*代表OBS桶名，仅支持部分区域，当前支持的区域和对应的OBS桶名请参见表3-1。GaussDB(DWS) 集群不支持跨区域访问OBS桶数据。
   - 外表字段需与即将注入数据的普通表字段保持一致。例如此处store_sales表及其对应的外表obs_from_store_sales_001，他们的字段是一致的。
   - 这些外表语法能够帮助您获取OBS存储桶中为本次实践所提供的样例数据。如果您需要加载其他样例数据，需进行SERVER gsmpp_server OPTIONS的调整。具体可参考关于OBS并行导入。
   - 认证用的AK和SK硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全。

```
CREATE FOREIGN TABLE obs_from_store_sales_001
(
    ss_sold_date_sk           integer                       ,
    ss_sold_time_sk           integer                       ,
    ss_item_sk                integer               not null,
    ss_customer_sk            integer                       ,
    ss_cdemo_sk               integer                       ,
    ss_hdemo_sk               integer                       ,
    ss_addr_sk                integer                       ,
    ss_store_sk               integer                       ,
    ss_promo_sk               integer                       ,
```



```
    ss_ticket_number          bigint               not null,
    ss_quantity               integer                      ,
    ss_wholesale_cost         decimal(7,2)                 ,
    ss_list_price             decimal(7,2)                 ,
    ss_sales_price            decimal(7,2)                 ,
    ss_ext_discount_amt       decimal(7,2)                 ,
    ss_ext_sales_price        decimal(7,2)                 ,
    ss_ext_wholesale_cost     decimal(7,2)                 ,
    ss_ext_list_price         decimal(7,2)                 ,
    ss_ext_tax                decimal(7,2)                 ,
    ss_coupon_amt             decimal(7,2)                 ,
    ss_net_paid               decimal(7,2)                 ,
    ss_net_paid_inc_tax       decimal(7,2)                 ,
    ss_net_profit             decimal(7,2)
)
-- Configure OBS server information and data format details.
SERVER gsmpp_server
OPTIONS (
LOCATION 'obs://<obs_bucket_name>/tpcds/store_sales',
FORMAT 'text',
DELIMITER '|',
ENCODING 'utf8',
NOESCAPING 'true',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
REJECT_LIMIT 'unlimited',
CHUNKSIZE '64'
)
-- If create foreign table failed,record error message
WITH err_obs_from_store_sales_001;
```

2. 将创建外表语句中的参数ACCESS_KEY和SECRET_ACCESS_KEY替换为实际值，然后在客户端工具中执行替换后的语句创建外表。

   ACCESS_KEY和SECRET_ACCESS_KEY的值，请参见的**创建访问密钥（AK和SK）**章节进行获取，然后将获取到的值替换到创建外表语句中。

3. 执行数据导入。

   创建包含如下语句的insert.sql脚本文件，并执行.sql脚本文件。

   ```
   \timing on
   \parallel on 4
   INSERT INTO store_sales SELECT * FROM obs_from_store_sales_001;
   INSERT INTO date_dim SELECT * FROM obs_from_date_dim_001;
   INSERT INTO store SELECT * FROM obs_from_store_001;
   INSERT INTO item SELECT * FROM obs_from_item_001;
   INSERT INTO time_dim SELECT * FROM obs_from_time_dim_001;
   INSERT INTO promotion SELECT * FROM obs_from_promotion_001;
   INSERT INTO customer_demographics SELECT * from obs_from_customer_demographics_001 ;
   INSERT INTO customer_address SELECT * FROM obs_from_customer_address_001 ;
   INSERT INTO household_demographics SELECT * FROM obs_from_household_demographics_001;
   INSERT INTO customer SELECT * FROM obs_from_customer_001;
   INSERT INTO income_band SELECT * FROM obs_from_income_band_001;
   \parallel off
   ```

   结果应该类似如下：

   ```
   SET
   Timing is on.
   SET
   Time: 2.831 ms
   Parallel is on with scale 4.
   Parallel is off.
   INSERT 0 402
   Time: 1820.909 ms
   INSERT 0 73049
   Time: 2715.275 ms
   INSERT 0 86400
   Time: 2377.056 ms
   INSERT 0 1000
   Time: 4037.155 ms
   ```



```
INSERT 0 204000
Time: 7124.190 ms
INSERT 0 7200
Time: 2227.776 ms
INSERT 0 1920800
Time: 8672.647 ms
INSERT 0 20
Time: 2273.501 ms
INSERT 0 1000000
Time: 11430.991 ms
INSERT 0 1981703
Time: 20270.750 ms
INSERT 0 287997024
Time: 341395.680 ms
total time: 341584  ms
```

4. 计算所有11张表的总执行时间。该数字将作为加载时间记录在下一小节步骤**步骤1**中的基准表内。
5. 执行以下命令，验证每个表是否都已正确加载并将行数记录到表中。

```
SELECT COUNT(*) FROM store_sales;
SELECT COUNT(*) FROM date_dim;
SELECT COUNT(*) FROM store;
SELECT COUNT(*) FROM item;
SELECT COUNT(*) FROM time_dim;
SELECT COUNT(*) FROM promotion;
SELECT COUNT(*) FROM customer_demographics;
SELECT COUNT(*) FROM customer_address;
SELECT COUNT(*) FROM household_demographics;
SELECT COUNT(*) FROM customer;
SELECT COUNT(*) FROM income_band;
```

以下显示每个SS表的行数：

表名称	行数
Store_Sales	287997024
Date_Dim	73049
Store	402
Item	204000
Time_Dim	86400
Promotion	1000
Customer_Demographics	1920800
Customer_Address	1000000
Household_Demographics	7200
Customer	1981703
Income_Band	20

步骤4 执行ANALYZE更新统计信息。

```
ANALYZE;
```

返回ANALYZE后，表示执行成功。


```
ANALYZE
```

ANALYZE语句可收集数据库中与表内容相关的统计信息，统计结果存储在系统表PG_STATISTIC中。查询优化器会使用这些统计数据，以生成最有效的执行计划。

建议在执行了大批量插入/删除操作后，例行对表或全库执行ANALYZE语句更新统计信息。

----结束

3.5 步骤 2：测试初始表结构下的系统性能并建立基线

在优化表结构前后，请测试和记录以下详细信息以对比系统性能差异：

- 数据加载时间。
- 表占用的存储空间大小。
- 查询性能。

本次实践中的示例基于使用8节点的dws.d2.xlarge集群。因为系统性能受到许多因素的影响，即使您使用相同的集群配置，结果也会有所不同。

机器型号	dws.d2.xlarge VM
CPU	4*CPU E5-2680 v2 @ 2.80GHZ
内存	32GB
网络	1GB
磁盘	1.63TB
节点数目	8

请使用下面的基准表来记录结果。

基准	**优化前**	**优化后**
加载时间（11张表）	341584 ms	-
占用存储		
Store_Sales	-	-
Date_Dim	-	-
Store	-	-
Item	-	-
Time_Dim	-	-
Promotion	-	-
Customer_Demographics	-	-

基准	优化前	优化后
Customer_Address	-	-
Household_Demographics	-	-
Customer	-	-
Income_Band	-	-
总存储空间	-	-
查询执行时间		
查询1	-	-
查询2	-	-
查询3	-	-
总执行时间	-	-

执行以下步骤测试优化前的系统性能，以建立基准。

步骤1 将上一节记下的所有11张表的累计加载时间填入基准表的“优化前”一列。

步骤2 记录各表的存储使用情况。

使用pg_size_pretty函数查询每张表使用的磁盘空间，并将结果记录到基准表中。

```
SELECT T_NAME, PG_SIZE_PRETTY(PG_RELATION_SIZE(t_name)) FROM (VALUES('store_sales'),('date_dim'),
('store'),('item'),('time_dim'),('promotion'),('customer_demographics'),('customer_address'),
('household_demographics'),('customer'),('income_band')) AS names1(t_name);
```

显示结果如下：

```
        t_name          | pg_size_pretty
------------------------+----------------
 store_sales            | 42 GB
 date_dim               | 11 MB
 store                  | 232 kB
 item                   | 110 MB
 time_dim               | 11 MB
 promotion              | 256 kB
 customer_demographics  | 171 MB
 customer_address       | 170 MB
 household_demographics | 504 kB
 customer               | 441 MB
 income_band            | 88 kB
(11 rows)
```

步骤3 测试查询性能。

运行如下三个查询，并记录每个查询的耗费时间。考虑到操作系统缓存的影响，同一查询在每次执行时耗时会有不同属正常现象，建议多测试几次，取一组平均值。

```
\timing on
SELECT * FROM (SELECT  COUNT(*)
FROM store_sales
    ,household_demographics
    ,time_dim, store
```



```
WHERE ss_sold_time_sk = time_dim.t_time_sk
    AND ss_hdemo_sk = household_demographics.hd_demo_sk
    AND ss_store_sk = s_store_sk
    AND time_dim.t_hour = 8
    AND time_dim.t_minute >= 30
    AND household_demographics.hd_dep_count = 5
    AND store.s_store_name = 'ese'
ORDER BY COUNT(*)
 ) LIMIT 100;

SELECT * FROM (SELECT  i_brand_id brand_id, i_brand brand, i_manufact_id, i_manufact,
SUM(ss_ext_sales_price) ext_price
 FROM date_dim, store_sales, item,customer,customer_address,store
 WHERE d_date_sk = ss_sold_date_sk
   AND ss_item_sk = i_item_sk
   AND i_manager_id=8
   AND d_moy=11
   AND d_year=1999
   AND ss_customer_sk = c_customer_sk
   AND c_current_addr_sk = ca_address_sk
   AND substr(ca_zip,1,5) <> substr(s_zip,1,5)
   AND ss_store_sk = s_store_sk
 GROUP BY i_brand
      ,i_brand_id
      ,i_manufact_id
      ,i_manufact
 ORDER BY ext_price desc
         ,i_brand
         ,i_brand_id
         ,i_manufact_id
         ,i_manufact
 ) LIMIT 100;

SELECT * FROM (SELECT  s_store_name, s_store_id,
        SUM(CASE WHEN (d_day_name='Sunday') THEN ss_sales_price ELSE null END) sun_sales,
        SUM(CASE WHEN (d_day_name='Monday') THEN ss_sales_price ELSE null END) mon_sales,
        SUM(CASE WHEN (d_day_name='Tuesday') THEN ss_sales_price ELSE  null END) tue_sales,
        SUM(CASE WHEN (d_day_name='Wednesday') THEN ss_sales_price ELSE null END) wed_sales,
        SUM(CASE WHEN (d_day_name='Thursday') THEN ss_sales_price ELSE null END) thu_sales,
        SUM(CASE WHEN (d_day_name='Friday') THEN ss_sales_price ELSE null END) fri_sales,
        SUM(CASE WHEN (d_day_name='Saturday') THEN ss_sales_price ELSE null END) sat_sales
 FROM date_dim, store_sales, store
 WHERE d_date_sk = ss_sold_date_sk AND
       s_store_sk = ss_store_sk AND
       s_gmt_offset = -5 AND
       d_year = 2000
 GROUP BY s_store_name, s_store_id
 ORDER BY s_store_name, s_store_id,sun_sales,mon_sales,tue_sales,wed_sales,thu_sales,fri_sales,sat_sales
  ) LIMIT 100;
```

----结束

经过上面的统计后，记录的基准表信息如下：

基准	优化前	优化后
加载时间（11张表）	341584ms	-
占用存储		
Store_Sales	42GB	-
Date_Dim	11MB	-
Store	232kB	-
Item	110MB	-

基准	优化前	优化后
Time_Dim	11MB	-
Promotion	256kB	-
Customer_Demographics	171MB	-
Customer_Address	170MB	-
Household_Demographics	504kB	-
Customer	441MB	-
Income_Band	88kB	-
总存储空间	42GB	-
查询执行时间		
查询1	14552.05ms	-
查询2	27952.36ms	-
查询3	17721.15ms	-
总执行时间	60225.56ms	-

3.6 步骤 3：调优表操作具体步骤

选择存储方式

此实践中所使用的样例表为典型的TPC-DS表，是典型的多字段表，统计分析类查询场景多，因此选择列存存储方式。

```
WITH (ORIENTATION = column )
```

选择压缩级别

在**步骤1：创建初始表并加装样例数据**中没有指定压缩比，GaussDB(DWS)默认为用户选择LOW级别压缩比。在这一步中我们把压缩比调整为MIDDLE级别，进行验证对比。

增加存储方式和压缩比后的建表样例如下：

```
CREATE TABLE store_sales
(
    ss_sold_date_sk           integer                       ,
    ss_sold_time_sk           integer                       ,
    ss_item_sk                integer               not null,
    ss_customer_sk            integer                       ,
    ss_cdemo_sk               integer                       ,
    ss_hdemo_sk               integer                       ,
    ss_addr_sk                integer                       ,
    ss_store_sk               integer                       ,
    ss_promo_sk               integer                       ,
```

```
    ss_ticket_number          bigint               not null,
    ss_quantity               integer                      ,
    ss_wholesale_cost         decimal(7,2)                 ,
    ss_list_price             decimal(7,2)                 ,
    ss_sales_price            decimal(7,2)                 ,
    ss_ext_discount_amt       decimal(7,2)                 ,
    ss_ext_sales_price        decimal(7,2)                 ,
    ss_ext_wholesale_cost     decimal(7,2)                 ,
    ss_ext_list_price         decimal(7,2)                 ,
    ss_ext_tax                decimal(7,2)                 ,
    ss_coupon_amt             decimal(7,2)                 ,
    ss_net_paid               decimal(7,2)                 ,
    ss_net_paid_inc_tax       decimal(7,2)                 ,
    ss_net_profit             decimal(7,2)
)
WITH (ORIENTATION = column,COMPRESSION=middle);
```

选择分布方式

依据**步骤2：测试初始表结构下的系统性能并建立基线**中所基线的各表大小，分布方式设置如下：

表名	行数	分布方式
Store_Sales	287997024	Hash
Date_Dim	73049	Replication
Store	402	Replication
Item	204000	Replication
Time_Dim	86400	Replication
Promotion	1000	Replication
Customer_Demographics	1920800	Hash
Customer_Address	1000000	Hash
Household_Demographics	7200	Replication
Customer	1981703	Hash
Income_Band	20	Replication

选择分布列

当表的分布方式选择了Hash分布策略时，分布列选取至关重要。在这一步中，建议按照**选择分布列**选择分布键：

选择各表的主键作为Hash表分布键。

表名	记录数	分布方式	分布键
Store_Sales	287997024	Hash	ss_item_sk

表名	记录数	分布方式	分布键
Date_Dim	73049	Replication	-
Store	402	Replication	-
Item	204000	Replication	-
Time_Dim	86400	Replication	-
Promotion	1000	Replication	-
Customer_Demographics	1920800	Hash	cd_demo_sk
Customer_Address	1000000	Hash	ca_address_sk
Household_Demographics	7200	Replication	-
Customer	1981703	Hash	c_customer_sk
Income_Band	20	Replication	-

3.7 步骤 4：创建新表并加载数据

为每张表选择了存储方式、压缩级别、分布方式和分布列后，使用这些属性创建表并重新加载数据。以便对比表设计前后的系统性能。

步骤1 执行CREATE TABLE创建表前，删除前面创建的表。

```
DROP TABLE store_sales;
DROP TABLE date_dim;
DROP TABLE store;
DROP TABLE item;
DROP TABLE time_dim;
DROP TABLE promotion;
DROP TABLE customer_demographics;
DROP TABLE customer_address;
DROP TABLE household_demographics;
DROP TABLE customer;
DROP TABLE income_band;

DROP FOREIGN TABLE obs_from_store_sales_001;
DROP FOREIGN TABLE obs_from_date_dim_001;
DROP FOREIGN TABLE obs_from_store_001;
DROP FOREIGN TABLE obs_from_item_001;
DROP FOREIGN TABLE obs_from_time_dim_001;
DROP FOREIGN TABLE obs_from_promotion_001;
DROP FOREIGN TABLE obs_from_customer_demographics_001;
DROP FOREIGN TABLE obs_from_customer_address_001;
DROP FOREIGN TABLE obs_from_household_demographics_001;
DROP FOREIGN TABLE obs_from_customer_001;
DROP FOREIGN TABLE obs_from_income_band_001;
```

步骤2 创建具有存储方式和分布方式的表。

限于篇幅，此处仅给出再次创建store_sales的语法。请从附录**设计调优后二次表创建**中拷贝其他表的语法进行创建。

```
CREATE TABLE store_sales
(
```



```
    ss_sold_date_sk           integer                       ,
    ss_sold_time_sk           integer                       ,
    ss_item_sk                integer               not null,
    ss_customer_sk            integer                       ,
    ss_cdemo_sk               integer                       ,
    ss_hdemo_sk               integer                       ,
    ss_addr_sk                integer                       ,
    ss_store_sk               integer                       ,
    ss_promo_sk               integer                       ,
    ss_ticket_number          bigint                not null,
    ss_quantity               integer                       ,
    ss_wholesale_cost         decimal(7,2)                  ,
    ss_list_price             decimal(7,2)                  ,
    ss_sales_price            decimal(7,2)                  ,
    ss_ext_discount_amt       decimal(7,2)                  ,
    ss_ext_sales_price        decimal(7,2)                  ,
    ss_ext_wholesale_cost     decimal(7,2)                  ,
    ss_ext_list_price         decimal(7,2)                  ,
    ss_ext_tax                decimal(7,2)                  ,
    ss_coupon_amt             decimal(7,2)                  ,
    ss_net_paid               decimal(7,2)                  ,
    ss_net_paid_inc_tax       decimal(7,2)                  ,
    ss_net_profit             decimal(7,2)
)
WITH (ORIENTATION = column,COMPRESSION=middle)
DISTRIBUTE BY hash (ss_item_sk);
```

步骤3 **为这些表加载样例数据**。

步骤4 在基准表中记录加载时间。

基准	优化前	优化后
加载时间（11张表）	341584ms	257241ms
占用存储		
Store_Sales	42GB	-
Date_Dim	11MB	-
Store	232kB	-
Item	110MB	-
Time_Dim	11MB	-
Promotion	256kB	-
Customer_Demographics	171MB	-
Customer_Address	170MB	-
Household_Demographics	504kB	-
Customer	441MB	-
Income_Band	88kB	-
总存储空间	42GB	-
查询执行时间		
查询1	14552.05ms	-

基准	优化前	优化后
查询2	27952.36ms	-
查询3	17721.15ms	-
总执行时间	60225.56ms	-

步骤5 执行ANALYZE更新统计信息。

```
ANALYZE;
```

返回ANALYZE后，表示执行成功。

```
ANALYZE
```

步骤6 检查数据倾斜性。

对于Hash分表策略，如果分布列选择不当，可能导致数据倾斜，查询时出现部分DN的I/O短板，从而影响整体查询性能。因此在采用Hash分表策略之后需对表的数据进行数据倾斜性检查，以确保数据在各个DN上是均匀分布的。可以使用以下SQL检查数据倾斜性

```
SELECT a.count,b.node_name FROM (SELECT count(*) AS count,xc_node_id FROM table_name GROUP BY xc_node_id) a, pgxc_node b WHERE a.xc_node_id=b.node_id ORDER BY a.count desc;
```

其中xc_node_id对应DN，一般来说，**不同DN的数据量相差5%以上即可视为倾斜，如果相差10%以上就必须要调整分布列**。GaussDB(DWS)支持多分布列特性，可以更好地满足数据分布的均匀性要求。

----结束

3.8 步骤 5：测试新的表结构下的系统性能

重新创建了具有存储方式、压缩级别、分布方式和分布列的测试数据集后，重新测试系统性能。

步骤1 记录各表的存储使用情况。

使用pg_size_pretty函数查询每张表使用的磁盘空间，并将结果记录到基准表中。

```
SELECT T_NAME, PG_SIZE_PRETTY(PG_RELATION_SIZE(t_name)) FROM (VALUES('store_sales'),('date_dim'),('store'),('item'),('time_dim'),('promotion'),('customer_demographics'),('customer_address'),('household_demographics'),('customer'),('income_band')) AS names1(t_name);
        t_name          | pg_size_pretty
------------------------+----------------
 store_sales            | 14 GB
 date_dim               | 27 MB
 store                  | 4352 kB
 item                   | 259 MB
 time_dim               | 14 MB
 promotion              | 3200 kB
 customer_demographics  | 11 MB
 customer_address       | 27 MB
 household_demographics | 1280 kB
 customer               | 111 MB
 income_band            | 896 kB
(11 rows)
```

步骤2 测试查询性能，并将性能数据录入基准表中。

再次运行如下三个查询，并记录每个查询的耗费时间。


```
\timing on
SELECT * FROM (SELECT  COUNT(*)
FROM store_sales
    ,household_demographics
    ,time_dim, store
WHERE ss_sold_time_sk = time_dim.t_time_sk
    AND ss_hdemo_sk = household_demographics.hd_demo_sk
    AND ss_store_sk = s_store_sk
    AND time_dim.t_hour = 8
    AND time_dim.t_minute >= 30
    AND household_demographics.hd_dep_count = 5
    AND store.s_store_name = 'ese'
ORDER BY COUNT(*)
 ) LIMIT 100;

SELECT * FROM (SELECT  i_brand_id brand_id, i_brand brand, i_manufact_id, i_manufact,
SUM(ss_ext_sales_price) ext_price
 FROM date_dim, store_sales, item,customer,customer_address,store
 WHERE d_date_sk = ss_sold_date_sk
   AND ss_item_sk = i_item_sk
   AND i_manager_id=8
   AND d_moy=11
   AND d_year=1999
   AND ss_customer_sk = c_customer_sk
   AND c_current_addr_sk = ca_address_sk
   AND substr(ca_zip,1,5) <> substr(s_zip,1,5)
   AND ss_store_sk = s_store_sk
 GROUP BY i_brand
      ,i_brand_id
      ,i_manufact_id
      ,i_manufact
 ORDER BY ext_price desc
         ,i_brand
         ,i_brand_id
         ,i_manufact_id
         ,i_manufact
 ) LIMIT 100;

SELECT * FROM (SELECT  s_store_name, s_store_id,
        SUM(CASE WHEN (d_day_name='Sunday') THEN ss_sales_price ELSE null END) sun_sales,
        SUM(CASE WHEN (d_day_name='Monday') THEN ss_sales_price ELSE null END) mon_sales,
        SUM(CASE WHEN (d_day_name='Tuesday') THEN ss_sales_price ELSE  null END) tue_sales,
        SUM(CASE WHEN (d_day_name='Wednesday') THEN ss_sales_price ELSE null END) wed_sales,
        SUM(CASE WHEN (d_day_name='Thursday') THEN ss_sales_price ELSE null END) thu_sales,
        SUM(CASE WHEN (d_day_name='Friday') THEN ss_sales_price ELSE null END) fri_sales,
        SUM(CASE WHEN (d_day_name='Saturday') THEN ss_sales_price ELSE null END) sat_sales
 FROM date_dim, store_sales, store
 WHERE d_date_sk = ss_sold_date_sk AND
       s_store_sk = ss_store_sk AND
       s_gmt_offset = -5 AND
       d_year = 2000
 GROUP BY s_store_name, s_store_id
 ORDER BY s_store_name, s_store_id,sun_sales,mon_sales,tue_sales,wed_sales,thu_sales,fri_sales,sat_sales
  ) LIMIT 100;
```

下面的基准表显示了本次实践中所用集群的验证结果。您的结果可能会因多方面的原因而有所变化，但规律性应该相差不大。考虑到操作系统缓存的影响，相同表结构的同一查询在每次执行时耗时会有不同属正常现象，建议多测试几次，取一组平均值。

基准	优化前	优化后
加载时间（11张表）	341584ms	257241ms
占用存储		
Store_Sales	42GB	14GB

基准	优化前	优化后
Date_Dim	11MB	27MB
Store	232kB	4352kB
Item	110MB	259MB
Time_Dim	11MB	14MB
Promotion	256kB	3200kB
Customer_Demographics	171MB	11MB
Customer_Address	170MB	27MB
Household_Demographics	504kB	1280kB
Customer	441MB	111MB
Income_Band	88kB	896kB
总存储空间	42GB	15GB
查询执行时间		
查询1	14552.05ms	1783.353ms
查询2	27952.36ms	14247.803ms
查询3	17721.15ms	11441.659ms
总执行时间	60225.56ms	27472.815ms

步骤3 如果对表设计后的性能还有更高期望，可以运行EXPLAIN PERFORMANCE以查看执行计划进行调优。

关于执行计划的更详细介绍及查询优化请参考**SQL执行计划介绍**及**优化查询性能概述** 。

----结束

3.9 步骤 6：调优表性能评估

经过测试，得到了优化表前后的加载时间、存储占用情况和查询执行时间，并记录了结果，针对结果进行对比分析。

下表显示了本次实践所用集群的示例结果。您的结果会有所不同，但应显示出相似的提升。

基准	优化前	优化后	改变	百分比
加载时间（11张表）	341584ms	257241ms	-84343ms	-24.7%

基准	优化前	优化后	改变	百分比
占用存储			-	-
Store_Sales	42GB	14GB	-28GB	-66.7%
Date_Dim	11MB	27MB	16MB	145.5%
Store	232kB	4352kB	4120kB	1775.9%
Item	110MB	259MB	149MB	1354.5%
Time_Dim	11MB	14MB	13MB	118.2%
Promotion	256kB	3200kB	2944kB	1150%
Customer_Demographics	171MB	11MB	-160MB	-93.6
Customer_Address	170MB	27MB	-143MB	-84.1%
Household_Demographics	504kB	1280kB	704kB	139.7%
Customer	441MB	111MB	-330MB	-74.8%
Income_Band	88kB	896kB	808kB	918.2%
总存储空间	42GB	15GB	-27GB	-64.3%
查询执行时间			-	-
查询1	14552.05ms	1783.353ms	-12768.697ms	-87.7%
查询2	27952.36ms	14247.803ms	-13704.557ms	-49.0%
查询3	17721.15ms	11441.659ms	-6279.491ms	-35.4%
总执行时间	60225.56ms	27472.815ms	-32752.745ms	-54.4%

调优后表的评估

- 加载时间减少了24.7%。

  分布方式对加载的影响明显，Hash分布方式提升加载效率，Replication分布方式会降低加载效率。在CPU和I/O均充足的情况下，压缩级别对加载效率影响不大。通常，列存表的加载效率比行存要高。

- 存储占用减少了64.3%。

  压缩级别、列存和Hash分布均能够节省存储空间。Replication表会明显加大存储占用，但是可以减小网络开销。通过对小表采用Replication方式，是使用小量空间换取性能的正向做法。

- 查询性能（速度）提升了54.4%，即查询时间减少了54.4%。

  查询性能方面的提升源于对存储方式、分布方式和分布列的优化。在多字段表，统计分析类查询场景下，列存可以提升查询性能。对于Hash分布表，在读/写数据时可以利用各个节点的IO资源，提升表的读/写速度。

  重写查询和配置工作负载管理 (WLM) 通常可进一步提升查询性能。有关更多信息，请参阅**优化查询性能概述**。

基于调优表实践的具体步骤，您可以进一步应用“**调优表实践**”中的优秀实践方法来改进表的分配，以达到您所期望的数据加载、存储和查询方面的效果。

清除资源

在完成本次实践之后，应遵循**删除集群**中的步骤删除集群。

如果希望保留集群，但撤销SS表使用的存储空间，请执行以下命令。

```
DROP TABLE store_sales;
DROP TABLE date_dim;
DROP TABLE store;
DROP TABLE item;
DROP TABLE time_dim;
DROP TABLE promotion;
DROP TABLE customer_demographics;
DROP TABLE customer_address;
DROP TABLE household_demographics;
DROP TABLE customer;
DROP TABLE income_band;
```

3.10 附录：表创建语法

3.10.1 附录使用说明

本节所附为调优表实践中使用到的SQL测试语句，推荐您将每节的SQL语句拷贝并另存为.sql文件。例如，创建一个包含“**初始表创建**”SQL语句的create_table_fir.sql文件。创建后使用SQL客户端工具执行.sql文件效率更高，且利于统计用例的总耗费时间。使用gsql运行.sql文件的方法如下：

```
gsql -d database_name -h dws_ip -U username -p port_number -W password -f XXX.sql
```

示例中的部分信息请替换成您所用GaussDB(DWS)集群的实际值：

```
gsql -d postgres -h 10.10.0.1 -U dbadmin -p 8000 -W password -f create_table_fir.sql
```

如示例中涉及的以下信息可根据实际情况替换：

- postgres：所要连接的数据库名称。
- 10.10.0.1：集群连接地址。
- dbadmin：集群数据库的用户名。默认管理员用户为“dbadmin”。
- 8000：创建集群时设置的“数据库端口”。
- password：创建集群时设置的密码。

3.10.2 初始表创建

此小节所附为本Tutorial首次创建表时所用到的表创建语法。这些表没有设置存储方式、分布键、分布方式和压缩方式。

```
CREATE TABLE store_sales
(
    ss_sold_date_sk           integer                       ,
    ss_sold_time_sk           integer                       ,
    ss_item_sk                integer               not null,
    ss_customer_sk            integer                       ,
    ss_cdemo_sk               integer                       ,
    ss_hdemo_sk               integer                       ,
    ss_addr_sk                integer                       ,
    ss_store_sk               integer                       ,
    ss_promo_sk               integer                       ,
    ss_ticket_number          bigint                not null,
    ss_quantity               integer                       ,
    ss_wholesale_cost         decimal(7,2)                  ,
    ss_list_price             decimal(7,2)                  ,
    ss_sales_price            decimal(7,2)                  ,
    ss_ext_discount_amt       decimal(7,2)                  ,
    ss_ext_sales_price        decimal(7,2)                  ,
    ss_ext_wholesale_cost     decimal(7,2)                  ,
    ss_ext_list_price         decimal(7,2)                  ,
    ss_ext_tax                decimal(7,2)                  ,
    ss_coupon_amt             decimal(7,2)                  ,
    ss_net_paid               decimal(7,2)                  ,
    ss_net_paid_inc_tax       decimal(7,2)                  ,
    ss_net_profit             decimal(7,2)
) ;

CREATE TABLE date_dim
(
    d_date_sk                 integer               not null,
    d_date_id                 char(16)              not null,
    d_date                    date                          ,
    d_month_seq               integer                       ,
    d_week_seq                integer                       ,
    d_quarter_seq             integer                       ,
    d_year                    integer                       ,
    d_dow                     integer                       ,
    d_moy                     integer                       ,
    d_dom                     integer                       ,
    d_qoy                     integer                       ,
    d_fy_year                 integer                       ,
    d_fy_quarter_seq          integer                       ,
    d_fy_week_seq             integer                       ,
    d_day_name                char(9)                       ,
    d_quarter_name            char(6)                       ,
    d_holiday                 char(1)                       ,
    d_weekend                 char(1)                       ,
    d_following_holiday       char(1)                       ,
    d_first_dom               integer                       ,
    d_last_dom                integer                       ,
    d_same_day_ly             integer                       ,
    d_same_day_lq             integer                       ,
    d_current_day             char(1)                       ,
    d_current_week            char(1)                       ,
    d_current_month           char(1)                       ,
    d_current_quarter         char(1)                       ,
    d_current_year            char(1)
) ;

CREATE TABLE store
(
    s_store_sk                integer               not null,
    s_store_id                char(16)              not null,
    s_rec_start_date          date                          ,
    s_rec_end_date            date                          ,
    s_closed_date_sk          integer                       ,
    s_store_name              varchar(50)                   ,
    s_number_employees        integer                       ,
    s_floor_space             integer                       ,
```

```
    s_hours                 char(20)                ,
    s_manager               varchar(40)             ,
    s_market_id             integer                 ,
    s_geography_class       varchar(100)            ,
    s_market_desc           varchar(100)            ,
    s_market_manager        varchar(40)             ,
    s_division_id           integer                 ,
    s_division_name         varchar(50)             ,
    s_company_id            integer                 ,
    s_company_name          varchar(50)             ,
    s_street_number         varchar(10)             ,
    s_street_name           varchar(60)             ,
    s_street_type           char(15)                ,
    s_suite_number          char(10)                ,
    s_city                  varchar(60)             ,
    s_county                varchar(30)             ,
    s_state                 char(2)                 ,
    s_zip                   char(10)                ,
    s_country               varchar(20)             ,
    s_gmt_offset            decimal(5,2)            ,
    s_tax_precentage        decimal(5,2)
) ;

CREATE TABLE item
(
    i_item_sk               integer             not null,
    i_item_id               char(16)            not null,
    i_rec_start_date        date                ,
    i_rec_end_date          date                ,
    i_item_desc             varchar(200)        ,
    i_current_price         decimal(7,2)        ,
    i_wholesale_cost        decimal(7,2)        ,
    i_brand_id              integer             ,
    i_brand                 char(50)            ,
    i_class_id              integer             ,
    i_class                 char(50)            ,
    i_category_id           integer             ,
    i_category              char(50)            ,
    i_manufact_id           integer             ,
    i_manufact              char(50)            ,
    i_size                  char(20)            ,
    i_formulation           char(20)            ,
    i_color                 char(20)            ,
    i_units                 char(10)            ,
    i_container             char(10)            ,
    i_manager_id            integer             ,
    i_product_name          char(50)
) ;

CREATE TABLE time_dim
(
    t_time_sk               integer             not null,
    t_time_id               char(16)            not null,
    t_time                  integer             ,
    t_hour                  integer             ,
    t_minute                integer             ,
    t_second                integer             ,
    t_am_pm                 char(2)             ,
    t_shift                 char(20)            ,
    t_sub_shift             char(20)            ,
    t_meal_time             char(20)
) ;

CREATE TABLE promotion
(
    p_promo_sk              integer             not null,
    p_promo_id              char(16)            not null,
    p_start_date_sk         integer             ,
    p_end_date_sk           integer             ,
```

```
    p_item_sk                 integer                       ,
    p_cost                    decimal(15,2)                 ,
    p_response_target         integer                       ,
    p_promo_name              char(50)                      ,
    p_channel_dmail           char(1)                       ,
    p_channel_email           char(1)                       ,
    p_channel_catalog         char(1)                       ,
    p_channel_tv              char(1)                       ,
    p_channel_radio           char(1)                       ,
    p_channel_press           char(1)                       ,
    p_channel_event           char(1)                       ,
    p_channel_demo            char(1)                       ,
    p_channel_details         varchar(100)                  ,
    p_purpose                 char(15)                      ,
    p_discount_active         char(1)
) ;

CREATE TABLE customer_demographics
(
    cd_demo_sk                integer               not null,
    cd_gender                 char(1)                       ,
    cd_marital_status         char(1)                       ,
    cd_education_status       char(20)                      ,
    cd_purchase_estimate      integer                       ,
    cd_credit_rating          char(10)                      ,
    cd_dep_count              integer                       ,
    cd_dep_employed_count     integer                       ,
    cd_dep_college_count      integer
) ;

CREATE TABLE customer_address
(
    ca_address_sk             integer               not null,
    ca_address_id             char(16)              not null,
    ca_street_number          char(10)                      ,
    ca_street_name            varchar(60)                   ,
    ca_street_type            char(15)                      ,
    ca_suite_number           char(10)                      ,
    ca_city                   varchar(60)                   ,
    ca_county                 varchar(30)                   ,
    ca_state                  char(2)                       ,
    ca_zip                    char(10)                      ,
    ca_country                varchar(20)                   ,
    ca_gmt_offset             decimal(5,2)                  ,
    ca_location_type          char(20)
) ;

CREATE TABLE household_demographics
(
    hd_demo_sk                integer               not null,
    hd_income_band_sk         integer                       ,
    hd_buy_potential          char(15)                      ,
    hd_dep_count              integer                       ,
    hd_vehicle_count          integer
) ;

CREATE TABLE customer
(
    c_customer_sk             integer               not null,
    c_customer_id             char(16)              not null,
    c_current_cdemo_sk        integer                       ,
    c_current_hdemo_sk        integer                       ,
    c_current_addr_sk         integer                       ,
    c_first_shipto_date_sk    integer                       ,
    c_first_sales_date_sk     integer                       ,
    c_salutation              char(10)                      ,
    c_first_name              char(20)                      ,
    c_last_name               char(30)                      ,
    c_preferred_cust_flag     char(1)                       ,
```

```
    c_birth_day               integer                   ,
    c_birth_month             integer                   ,
    c_birth_year              integer                   ,
    c_birth_country           varchar(20)               ,
    c_login                   char(13)                  ,
    c_email_address           char(50)                  ,
    c_last_review_date        char(10)
) ;

CREATE TABLE income_band
(
    ib_income_band_sk         integer               not null,
    ib_lower_bound            integer                   ,
    ib_upper_bound            integer
) ;
```

3.10.3 设计调优后二次表创建

本节所附为调优表实践中进行了存储方式、压缩级别、分布方式和分布列选择后，二次建表语法。

```
CREATE TABLE store_sales
(
    ss_sold_date_sk           integer                   ,
    ss_sold_time_sk           integer                   ,
    ss_item_sk                integer               not null,
    ss_customer_sk            integer                   ,
    ss_cdemo_sk               integer                   ,
    ss_hdemo_sk               integer                   ,
    ss_addr_sk                integer                   ,
    ss_store_sk               integer                   ,
    ss_promo_sk               integer                   ,
    ss_ticket_number          bigint                not null,
    ss_quantity               integer                   ,
    ss_wholesale_cost         decimal(7,2)              ,
    ss_list_price             decimal(7,2)              ,
    ss_sales_price            decimal(7,2)              ,
    ss_ext_discount_amt       decimal(7,2)              ,
    ss_ext_sales_price        decimal(7,2)              ,
    ss_ext_wholesale_cost     decimal(7,2)              ,
    ss_ext_list_price         decimal(7,2)              ,
    ss_ext_tax                decimal(7,2)              ,
    ss_coupon_amt             decimal(7,2)              ,
    ss_net_paid               decimal(7,2)              ,
    ss_net_paid_inc_tax       decimal(7,2)              ,
    ss_net_profit             decimal(7,2)
)
WITH (ORIENTATION = column,COMPRESSION=middle)
DISTRIBUTE BY hash (ss_item_sk);

CREATE TABLE date_dim
(
    d_date_sk                 integer               not null,
    d_date_id                 char(16)              not null,
    d_date                    date                      ,
    d_month_seq               integer                   ,
    d_week_seq                integer                   ,
    d_quarter_seq             integer                   ,
    d_year                    integer                   ,
    d_dow                     integer                   ,
    d_moy                     integer                   ,
    d_dom                     integer                   ,
    d_qoy                     integer                   ,
    d_fy_year                 integer                   ,
    d_fy_quarter_seq          integer                   ,
    d_fy_week_seq             integer                   ,
    d_day_name                char(9)                   ,
    d_quarter_name            char(6)                   ,
    d_holiday                 char(1)                   ,
```

```
    d_weekend                 char(1)                   ,
    d_following_holiday       char(1)                   ,
    d_first_dom               integer                   ,
    d_last_dom                integer                   ,
    d_same_day_ly             integer                   ,
    d_same_day_lq             integer                   ,
    d_current_day             char(1)                   ,
    d_current_week            char(1)                   ,
    d_current_month           char(1)                   ,
    d_current_quarter         char(1)                   ,
    d_current_year            char(1)
)
WITH (ORIENTATION = column,COMPRESSION=middle)
DISTRIBUTE BY replication;

CREATE TABLE store
(
    s_store_sk                integer               not null,
    s_store_id                char(16)              not null,
    s_rec_start_date          date                  ,
    s_rec_end_date            date                  ,
    s_closed_date_sk          integer               ,
    s_store_name              varchar(50)           ,
    s_number_employees        integer               ,
    s_floor_space             integer               ,
    s_hours                   char(20)              ,
    s_manager                 varchar(40)           ,
    s_market_id               integer               ,
    s_geography_class         varchar(100)          ,
    s_market_desc             varchar(100)          ,
    s_market_manager          varchar(40)           ,
    s_division_id             integer               ,
    s_division_name           varchar(50)           ,
    s_company_id              integer               ,
    s_company_name            varchar(50)           ,
    s_street_number           varchar(10)           ,
    s_street_name             varchar(60)           ,
    s_street_type             char(15)              ,
    s_suite_number            char(10)              ,
    s_city                    varchar(60)           ,
    s_county                  varchar(30)           ,
    s_state                   char(2)               ,
    s_zip                     char(10)              ,
    s_country                 varchar(20)           ,
    s_gmt_offset              decimal(5,2)          ,
    s_tax_precentage          decimal(5,2)
)
WITH (ORIENTATION = column,COMPRESSION=middle)
DISTRIBUTE BY replication;

CREATE TABLE item
(
    i_item_sk                 integer               not null,
    i_item_id                 char(16)              not null,
    i_rec_start_date          date                  ,
    i_rec_end_date            date                  ,
    i_item_desc               varchar(200)          ,
    i_current_price           decimal(7,2)          ,
    i_wholesale_cost          decimal(7,2)          ,
    i_brand_id                integer               ,
    i_brand                   char(50)              ,
    i_class_id                integer               ,
    i_class                   char(50)              ,
    i_category_id             integer               ,
    i_category                char(50)              ,
    i_manufact_id             integer               ,
    i_manufact                char(50)              ,
    i_size                    char(20)              ,
    i_formulation             char(20)              ,
```

```
    i_color                 char(20)                ,
    i_units                 char(10)                ,
    i_container             char(10)                ,
    i_manager_id            integer                 ,
    i_product_name          char(50)
)
WITH (ORIENTATION = column,COMPRESSION=middle)
DISTRIBUTE BY replication;

CREATE TABLE time_dim
(
    t_time_sk               integer             not null,
    t_time_id               char(16)            not null,
    t_time                  integer                 ,
    t_hour                  integer                 ,
    t_minute                integer                 ,
    t_second                integer                 ,
    t_am_pm                 char(2)                 ,
    t_shift                 char(20)                ,
    t_sub_shift             char(20)                ,
    t_meal_time             char(20)
)
WITH (ORIENTATION = column,COMPRESSION=middle)
DISTRIBUTE BY replication;

CREATE TABLE promotion
(
    p_promo_sk              integer             not null,
    p_promo_id              char(16)            not null,
    p_start_date_sk         integer                 ,
    p_end_date_sk           integer                 ,
    p_item_sk               integer                 ,
    p_cost                  decimal(15,2)           ,
    p_response_target       integer                 ,
    p_promo_name            char(50)                ,
    p_channel_dmail         char(1)                 ,
    p_channel_email         char(1)                 ,
    p_channel_catalog       char(1)                 ,
    p_channel_tv            char(1)                 ,
    p_channel_radio         char(1)                 ,
    p_channel_press         char(1)                 ,
    p_channel_event         char(1)                 ,
    p_channel_demo          char(1)                 ,
    p_channel_details       varchar(100)            ,
    p_purpose               char(15)                ,
    p_discount_active       char(1)
)
WITH (ORIENTATION = column,COMPRESSION=middle)
DISTRIBUTE BY replication;

CREATE TABLE customer_demographics
(
    cd_demo_sk              integer             not null,
    cd_gender               char(1)                 ,
    cd_marital_status       char(1)                 ,
    cd_education_status     char(20)                ,
    cd_purchase_estimate    integer                 ,
    cd_credit_rating        char(10)                ,
    cd_dep_count            integer                 ,
    cd_dep_employed_count   integer                 ,
    cd_dep_college_count    integer
)
WITH (ORIENTATION = column,COMPRESSION=middle)
DISTRIBUTE BY hash (cd_demo_sk);

CREATE TABLE customer_address
(
    ca_address_sk           integer             not null,
    ca_address_id           char(16)            not null,
```



```
    ca_street_number          char(10)                  ,
    ca_street_name            varchar(60)               ,
    ca_street_type            char(15)                  ,
    ca_suite_number           char(10)                  ,
    ca_city                   varchar(60)               ,
    ca_county                 varchar(30)               ,
    ca_state                  char(2)                   ,
    ca_zip                    char(10)                  ,
    ca_country                varchar(20)               ,
    ca_gmt_offset             decimal(5,2)              ,
    ca_location_type          char(20)
)
WITH (ORIENTATION = column,COMPRESSION=middle)
DISTRIBUTE BY hash (ca_address_sk);

CREATE TABLE household_demographics
(
    hd_demo_sk                integer               not null,
    hd_income_band_sk         integer                       ,
    hd_buy_potential          char(15)                      ,
    hd_dep_count              integer                       ,
    hd_vehicle_count          integer
)
 WITH (ORIENTATION = column,COMPRESSION=middle)
 DISTRIBUTE BY replication;

 CREATE TABLE customer
(
    c_customer_sk             integer               not null,
    c_customer_id             char(16)               not null,
    c_current_cdemo_sk        integer                       ,
    c_current_hdemo_sk        integer                       ,
    c_current_addr_sk         integer                       ,
    c_first_shipto_date_sk    integer                       ,
    c_first_sales_date_sk     integer                       ,
    c_salutation              char(10)                      ,
    c_first_name              char(20)                      ,
    c_last_name               char(30)                      ,
    c_preferred_cust_flag     char(1)                       ,
    c_birth_day               integer                       ,
    c_birth_month             integer                       ,
    c_birth_year              integer                       ,
    c_birth_country           varchar(20)                   ,
    c_login                   char(13)                      ,
    c_email_address           char(50)                      ,
    c_last_review_date        char(10)
)
WITH (ORIENTATION = column,COMPRESSION=middle)
DISTRIBUTE BY hash (c_customer_sk);

CREATE TABLE income_band
(
    ib_income_band_sk         integer               not null,
    ib_lower_bound            integer                       ,
    ib_upper_bound            integer
)
WITH (ORIENTATION = column,COMPRESSION=middle)
DISTRIBUTE BY replication;
```

3.10.4 外表创建

本小节所附外表语法用于获取本Tutorial使用到的示例数据。这些示例数据存储在OBS存储桶中，该存储桶向所有经过身份验证的云用户提供了读取权限。

说明

- 注意，以下语句中的*<obs_bucket_name>*代表OBS桶名，仅支持部分区域，当前支持的区域和对应的OBS桶名请参见**支持区域**。GaussDB(DWS) 集群不支持跨区域访问OBS桶数据。
- 运行时请将示例中的ACCESS_KEY和SECRET_ACCESS_KEY替换用户账户自己的凭证。
- 创建OBS外表的时候，只做映射关系，不会将数据拉取到GaussDB(DWS)磁盘。

```
CREATE FOREIGN TABLE obs_from_store_sales_001
(
    ss_sold_date_sk           integer                       ,
    ss_sold_time_sk           integer                       ,
    ss_item_sk                integer             not null,
    ss_customer_sk            integer                       ,
    ss_cdemo_sk               integer                       ,
    ss_hdemo_sk               integer                       ,
    ss_addr_sk                integer                       ,
    ss_store_sk               integer                       ,
    ss_promo_sk               integer                       ,
    ss_ticket_number          bigint              not null,
    ss_quantity               integer                       ,
    ss_wholesale_cost         decimal(7,2)                  ,
    ss_list_price             decimal(7,2)                  ,
    ss_sales_price            decimal(7,2)                  ,
    ss_ext_discount_amt       decimal(7,2)                  ,
    ss_ext_sales_price        decimal(7,2)                  ,
    ss_ext_wholesale_cost     decimal(7,2)                  ,
    ss_ext_list_price         decimal(7,2)                  ,
    ss_ext_tax                decimal(7,2)                  ,
    ss_coupon_amt             decimal(7,2)                  ,
    ss_net_paid               decimal(7,2)                  ,
    ss_net_paid_inc_tax       decimal(7,2)                  ,
    ss_net_profit             decimal(7,2)
)
SERVER gsmpp_server
OPTIONS (
LOCATION 'obs://<obs_bucket_name>/tpcds/store_sales',
FORMAT 'text',
DELIMITER '|',
ENCODING 'utf8',
NOESCAPING 'true',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
REJECT_LIMIT 'unlimited',
CHUNKSIZE '64'
)
WITH err_obs_from_store_sales_001;

CREATE FOREIGN TABLE obs_from_date_dim_001
(
    d_date_sk                 integer             not null,
    d_date_id                 char(16)            not null,
    d_date                    date                         ,
    d_month_seq               integer                      ,
    d_week_seq                integer                      ,
    d_quarter_seq             integer                      ,
    d_year                    integer                      ,
    d_dow                     integer                      ,
    d_moy                     integer                      ,
    d_dom                     integer                      ,
    d_qoy                     integer                      ,
    d_fy_year                 integer                      ,
    d_fy_quarter_seq          integer                      ,
    d_fy_week_seq             integer                      ,
    d_day_name                char(9)                      ,
    d_quarter_name            char(6)                      ,
    d_holiday                 char(1)                      ,
    d_weekend                 char(1)                      ,
    d_following_holiday       char(1)                      ,
```

```
    d_first_dom               integer                   ,
    d_last_dom                integer                   ,
    d_same_day_ly             integer                   ,
    d_same_day_lq             integer                   ,
    d_current_day             char(1)                   ,
    d_current_week            char(1)                   ,
    d_current_month           char(1)                   ,
    d_current_quarter         char(1)                   ,
    d_current_year            char(1)
)
SERVER gsmpp_server
OPTIONS (
LOCATION 'obs://<obs_bucket_name>/tpcds/date_dim' ,
FORMAT 'text',
DELIMITER '|',
ENCODING 'utf8',
NOESCAPING 'true',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
REJECT_LIMIT 'unlimited',
CHUNKSIZE '64'
)
WITH err_obs_from_date_dim_001;

CREATE FOREIGN TABLE obs_from_store_001
(
    s_store_sk                integer               not null,
    s_store_id                char(16)              not null,
    s_rec_start_date          date                  ,
    s_rec_end_date            date                  ,
    s_closed_date_sk          integer               ,
    s_store_name              varchar(50)           ,
    s_number_employees        integer               ,
    s_floor_space             integer               ,
    s_hours                   char(20)              ,
    s_manager                 varchar(40)           ,
    s_market_id               integer               ,
    s_geography_class         varchar(100)          ,
    s_market_desc             varchar(100)          ,
    s_market_manager          varchar(40)           ,
    s_division_id             integer               ,
    s_division_name           varchar(50)           ,
    s_company_id              integer               ,
    s_company_name            varchar(50)           ,
    s_street_number           varchar(10)           ,
    s_street_name             varchar(60)           ,
    s_street_type             char(15)              ,
    s_suite_number            char(10)              ,
    s_city                    varchar(60)           ,
    s_county                  varchar(30)           ,
    s_state                   char(2)               ,
    s_zip                     char(10)              ,
    s_country                 varchar(20)           ,
    s_gmt_offset              decimal(5,2)          ,
    s_tax_precentage          decimal(5,2)
)
SERVER gsmpp_server
OPTIONS (
LOCATION 'obs://<obs_bucket_name>/tpcds/store' ,
FORMAT 'text',
DELIMITER '|',
ENCODING 'utf8',
NOESCAPING 'true',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
REJECT_LIMIT 'unlimited',
CHUNKSIZE '64'
)
WITH err_obs_from_store_001;
```

```
CREATE FOREIGN TABLE obs_from_item_001
(
    i_item_sk               integer               not null,
    i_item_id               char(16)              not null,
    i_rec_start_date          date                        ,
    i_rec_end_date            date                        ,
    i_item_desc             varchar(200)                  ,
    i_current_price           decimal(7,2)                ,
    i_wholesale_cost          decimal(7,2)                ,
    i_brand_id              integer                       ,
    i_brand                 char(50)                      ,
    i_class_id              integer                       ,
    i_class                 char(50)                      ,
    i_category_id             integer                     ,
    i_category              char(50)                      ,
    i_manufact_id             integer                     ,
    i_manufact              char(50)                      ,
    i_size                  char(20)                      ,
    i_formulation             char(20)                    ,
    i_color                 char(20)                      ,
    i_units                 char(10)                      ,
    i_container               char(10)                    ,
    i_manager_id              integer                     ,
    i_product_name            char(50)
)
SERVER gsmpp_server
OPTIONS (
LOCATION 'obs://<obs_bucket_name>/tpcds/item' ,
FORMAT 'text',
DELIMITER '|',
ENCODING 'utf8',
NOESCAPING 'true',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
REJECT_LIMIT 'unlimited',
CHUNKSIZE '64'
)
WITH err_obs_from_item_001;

CREATE FOREIGN TABLE obs_from_time_dim_001
(
    t_time_sk               integer               not null,
    t_time_id               char(16)              not null,
    t_time                  integer                       ,
    t_hour                  integer                       ,
    t_minute                integer                       ,
    t_second                integer                       ,
    t_am_pm                   char(2)                     ,
    t_shift                 char(20)                      ,
    t_sub_shift               char(20)                    ,
    t_meal_time               char(20)
)
SERVER gsmpp_server
OPTIONS (
LOCATION 'obs://<obs_bucket_name>/tpcds/time_dim' ,
FORMAT 'text',
DELIMITER '|',
ENCODING 'utf8',
NOESCAPING 'true',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
REJECT_LIMIT 'unlimited',
CHUNKSIZE '64'
)
WITH err_obs_from_time_dim_001;

CREATE FOREIGN TABLE obs_from_promotion_001
(
```



```
    p_promo_sk               integer               not null,
    p_promo_id               char(16)              not null,
    p_start_date_sk          integer                       ,
    p_end_date_sk            integer                       ,
    p_item_sk                integer                       ,
    p_cost                   decimal(15,2)                 ,
    p_response_target        integer                       ,
    p_promo_name             char(50)                      ,
    p_channel_dmail          char(1)                       ,
    p_channel_email          char(1)                       ,
    p_channel_catalog        char(1)                       ,
    p_channel_tv             char(1)                       ,
    p_channel_radio          char(1)                       ,
    p_channel_press          char(1)                       ,
    p_channel_event          char(1)                       ,
    p_channel_demo           char(1)                       ,
    p_channel_details        varchar(100)                  ,
    p_purpose                char(15)                      ,
    p_discount_active        char(1)
)
SERVER gsmpp_server
OPTIONS (
LOCATION 'obs://<obs_bucket_name>/tpcds/promotion' ,
FORMAT 'text',
DELIMITER '|',
ENCODING 'utf8',
NOESCAPING 'true',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
REJECT_LIMIT 'unlimited',
CHUNKSIZE '64'
)
WITH err_obs_from_promotion_001;

CREATE FOREIGN TABLE obs_from_customer_demographics_001
(
    cd_demo_sk               integer               not null,
    cd_gender                char(1)                       ,
    cd_marital_status        char(1)                       ,
    cd_education_status      char(20)                      ,
    cd_purchase_estimate     integer                       ,
    cd_credit_rating         char(10)                      ,
    cd_dep_count             integer                       ,
    cd_dep_employed_count    integer                       ,
    cd_dep_college_count     integer
)
SERVER gsmpp_server
OPTIONS (
LOCATION 'obs://<obs_bucket_name>/tpcds/customer_demographics' ,
FORMAT 'text',
DELIMITER '|',
ENCODING 'utf8',
NOESCAPING 'true',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
REJECT_LIMIT 'unlimited',
CHUNKSIZE '64'
)
WITH err_obs_from_customer_demographics_001;

CREATE FOREIGN TABLE obs_from_customer_address_001
(
ca_address_sk integer not null,
ca_address_id char(16) not null,
ca_street_number char(10) ,
ca_street_name varchar(60) ,
ca_street_type char(15) ,
ca_suite_number char(10) ,
ca_city varchar(60) ,
```



```
ca_county varchar(30) ,
ca_state char(2) ,
ca_zip char(10) ,
ca_country varchar(20) ,
ca_gmt_offset float4 ,
ca_location_type char(20)
)
SERVER gsmpp_server
OPTIONS (
LOCATION 'obs://<obs_bucket_name>/tpcds/customer_address' ,
FORMAT 'text',
DELIMITER '|',
ENCODING 'utf8',
NOESCAPING 'true',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
REJECT_LIMIT 'unlimited',
CHUNKSIZE '64'
)
WITH err_obs_from_customer_address_001;

CREATE FOREIGN TABLE obs_from_household_demographics_001
(
    hd_demo_sk              integer             not null,
    hd_income_band_sk       integer                     ,
    hd_buy_potential        char(15)                    ,
    hd_dep_count            integer                     ,
    hd_vehicle_count        integer
)
SERVER gsmpp_server
OPTIONS (
LOCATION 'obs://<obs_bucket_name>/tpcds/household_demographics' ,
FORMAT 'text',
DELIMITER '|',
ENCODING 'utf8',
NOESCAPING 'true',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
REJECT_LIMIT 'unlimited',
CHUNKSIZE '64'
)
WITH err_obs_from_household_demographics_001;

CREATE FOREIGN TABLE obs_from_customer_001
(
    c_customer_sk           integer             not null,
    c_customer_id           char(16)            not null,
    c_current_cdemo_sk      integer                     ,
    c_current_hdemo_sk      integer                     ,
    c_current_addr_sk       integer                     ,
    c_first_shipto_date_sk  integer                     ,
    c_first_sales_date_sk   integer                     ,
    c_salutation            char(10)                    ,
    c_first_name            char(20)                    ,
    c_last_name             char(30)                    ,
    c_preferred_cust_flag   char(1)                     ,
    c_birth_day             integer                     ,
    c_birth_month           integer                     ,
    c_birth_year            integer                     ,
    c_birth_country         varchar(20)                 ,
    c_login                 char(13)                    ,
    c_email_address         char(50)                    ,
    c_last_review_date      char(10)
)
SERVER gsmpp_server
OPTIONS (
LOCATION 'obs://<obs_bucket_name>/tpcds/customer' ,
FORMAT 'text',
DELIMITER '|',
```



```
ENCODING 'utf8',
NOESCAPING 'true',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
REJECT_LIMIT 'unlimited',
CHUNKSIZE '64'
)
WITH err_obs_from_customer_001;

CREATE FOREIGN TABLE obs_from_income_band_001
(
    ib_income_band_sk         integer               not null,
    ib_lower_bound            integer                     ,
    ib_upper_bound            integer
)
SERVER gsmpp_server
OPTIONS (
LOCATION 'obs://<obs_bucket_name>/tpcds/income_band' ,
FORMAT 'text',
DELIMITER '|',
ENCODING 'utf8',
NOESCAPING 'true',
ACCESS_KEY 'access_key_value_to_be_replaced',
SECRET_ACCESS_KEY 'secret_access_key_value_to_be_replaced',
REJECT_LIMIT 'unlimited',
CHUNKSIZE '64'
)
WITH err_obs_from_income_band_001;
```

4 高级特性

4.1 快速创建时序表

场景介绍

时序表继承普通表的行存和列存语法，降低了用户学习成本，易理解和使用；

时序表具备数据生命周期管理的能力，每天各种维度的数据爆炸式增长，需要定期给表增加新的分区，避免新数据无法存储。而对于很久之前的数据，其价值较低且不经常访问，可以定期删除无用的数据。因此时序表需要具备定时增加分区和定时删除分区的能力。

本实践主要讲解如何快速创建适合自己业务的时序表，并对时序表进行分区管理，从而真正发挥时序表的优势。将对应的列指定为合适的类型，能够帮助我们更好的提高导入、查询等场景的性能，让业务场景运行的更加高效。如下图所示，以发电机组数据采样为例：

图 4-1 发电机组数据采样示意图

图 4-2 存储数据表

tag					field				time
发电机	生产厂商	型号	位置	ID	电压	功率	频率	电流相角	timestamp
发电机组1	SX	V310	V1-5-C253S	9527	330	1680	60	20	2022-0315T00:00:00Z
发电机组2	SH	V350	V1-5-C451S	8975	321	1556	50	13	2022-0315T00:00:00Z
发电机组3	XJ	V420	V1-5-C650S	8571	339	1597	58	33	2022-0315T00:00:00Z
发电机组1	SX	V310	V1-5-C253S	9527	350	1730	75	40	2022-0315T00:10:00Z
发电机组2	SH	V350	V1-5-C451S	8975	450	1658	55	25	2022-0315T00:10:00Z
发电机组3	XJ	V420	V1-5-C650S	8571	377	1678	70	39	2022-0315T00:10:00Z
……	……	……	……	……	……	……	……	……	……
发电机组1	SX	V310	V1-5-C253S	9527	1020	3980	240	175	2022-0315T00:80:00Z
发电机组2	SH	V350	V1-5-C451S	8975	1340	4219	225	190	2022-0315T00:80:00Z
发电机组3	XJ	V420	V1-5-C650S	8571	1211	4387	320	155	2022-0315T00:80:00Z

- 对于不随时间变化而变化，描述发电机的属性信息的列（发电机信息、生产厂商、型号、位置、ID）被设置为tag列，在建表时需要将对应的列后面指定为TSTag；
- 对于采样数据的维度（电压、功率、频率、电流相角）这些对应的采样数值随时间的变化而变，将这些维度设置为field列，建表语句数据类型后面指定为TSField；
- 最后一列指定为时间列time，存储field列数据对应的时间信息，建表时将指定为TSTime。

基本流程

本实践预计时长：30分钟，基本流程如下：

1. **创建ECS**。
2. **创建IoT数仓**。
3. **使用gsql命令行客户端连接集群**。
4. **创建时序表**。

创建 ECS

参见**自定义购买弹性云服务器**购买。购买后，参见**登录Linux弹性云服务器**进行登录。

须知

创建ECS过程中，注意选择与后续的IoT数仓在同一个区域、可用区和同一个VPC子网下，ECS的操作系统选择与gsql客户端（本例以CentOS 7.6为例），并选择以密码方式登录。

创建 IoT 数仓

步骤1 登录华为云管理控制台。

步骤2 在“服务列表”中，选择“大数据 > 数据仓库服务”，单击右上角“创建数据仓库集群”。

步骤3 参见表4-1进行参数配置。

表 4-1 软件配置

参数名称	配置方式
区域	选择“华北-北京四”。 **说明** • 本指导以“华北-北京四”为例进行介绍，如果您需要选择其他区域进行操作，请确保所有操作均在同一区域进行。 • 请确保DWS跟ECS在同一个区域、可用区和同一个VPC子网下。
可用分区	可用区2
产品类型	IoT数仓
计算类型	弹性云服务器
存储类型	SSD云盘
CPU架构	X86
节点规格	dwsx2.rt.2xlarge.m6（8 vCPU | 64GB | 100~4,000 GB SSD） **说明** 如规格售罄，可选择其他可用区或规格。
热数据存储	200 GB / 节点
节点数量	3
集群名称	dws-demo01
管理员用户	dbadmin
管理员密码	用户自定义
确认密码	再次输入自定义的管理员密码
数据库端口	8000
虚拟私有云	vpc-default
子网	subnet-default(192.168.0.0/24) **须知** 请确保与ECS在同一个VPC。
安全组	自动创建安全组
公网访问	现在购买
企业项目	default
高级配置	默认配置

步骤4 信息核对无误，单击“立即购买”，单击“提交”。

步骤5 等待约10分钟，待集群创建成功后，单击集群名称进入“基本信息”，在“网络”区域，单击安全组名称，确认安全组规则已添加，以IP为192.168.0.x的客户端网段为例（本例gsql所在ECS的内网IP为192.168.0.90），需要添加192.168.0.0/24，端口为8000的安全组规则。

步骤6 返回到集群“基本信息”界面，记录下“内网IP”。

----结束

使用 gsql 命令行客户端连接集群

步骤1 使用root用户远程登录到需要安装gsql的Linux主机，然后在Linux命令窗口，执行以下命令下载gsql客户端：

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

步骤2 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3 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4 执行以下命令，使用gsql客户端连接GaussDB(DWS)集群中的数据库，其中password为用户创建集群时自定义的密码。

```
gsql -d gaussdb -p 8000 -h 192.168.0.86 -U dbadmin -W password -r
```

显示如下信息表示gsql工具已经连接成功：

```
gaussdb=>
```

----结束

创建时序表

1. 以发电机组的场景作为示例，创建一张存储发电机组采样数据的时序表GENERATOR：

```
CREATE TABLE IF NOT EXISTS GENERATOR(
genset text TSTag,
manufacturer text TSTag,
model text TSTag,
location text TSTag,
ID bigint TSTag,
voltage numeric TSField,
power bigint TSField,
```



```
frequency numeric TSField,
angle numeric TSField,
time timestamptz TSTime) with (orientation=TIMESERIES, period='1 hour', ttl='1 month') distribute by
hash(model);
```

2. 查询当前时间：

```
select now();
              now
-------------------------------
2022-05-25 15:28:38.520757+08
(1 row)
```

3. 查询默认的分区与分区边界：

```
SELECT relname, boundaries FROM pg_partition where parentid=(SELECT oid FROM pg_class where
relname='generator') order by boundaries ;
   relname     |         boundaries
 ----------------+----------------------------
default_part_1 | {"2022-05-25 16:00:00+08"}
default_part_2 | {"2022-05-25 17:00:00+08"}
p1653505200    | {"2022-05-25 18:00:00+08"}
p1653541200    | {"2022-05-25 19:00:00+08"}
p1653577200    | {"2022-05-25 20:00:00+08"}
......
```

TSTag列支持类型：text, char, bool, int, big int。

TSTime列支持类型：timestamp with time zone, timestamp without time zone。在兼容Oracle语法的数据库中，也支持date类型。如果涉及到时区相关操作时，请选择带时区的时间类型。

TSField列支持的数据类型同列存表保持一致。

说明

- 写建表语句时，对于tag列的顺序，可以适当进行优化，将唯一性（distinct值）较高的列尽量写在前面，这样对于时序场景的性能有一些提升。
- 创建时序表时需要指定表级参数orientation属性设置为timeseries。
- 时序表不需要手动指定DISTRIBUTE BY和PARTITION BY, 默认按照所有tag列分布，且分区健默认为tstime指定的时间列。
- 对于**create table like**语法，需要自动从源表中继承列名和对应的kv_type类型。因此如果源表是非时序表，新表是时序表，对应的列的kv_type类型无法确定，则无法创建成功。
- 时序表必须指定一个时间属性（**TSTime**），且只能指定一个，且**TSTime**类型的列不能被删除。至少存在一个**TSTag**和**TSField**列，否则建表时会报错。

时序表以TSTIME列为分区键，具有自动分区管理功能。创建具有自动分区管理功能的分区表，帮助用户大大减少运维操作的时间。在上面的建表语句中，在表级参数项中可以看到，时序表指定了自动分区管理两个参数period和ttl。

- **period**：设置自动创建分区的间隔时间，默认值为1 day, 取值范围：1 hour ~ 100 years。默认会为时序表创建自增分区任务。自增分区任务动态为我们创建分区，保证当前时刻有足够充裕的分区用于导入数据。
- **ttl**：设置自动淘汰分区的时间，取值范围：1 hour ~ 100 years。默认不创建淘汰分区任务，需要用户自己在建表手动指定，或者建表后通过ALTER TABLE语法设置。淘汰分区的策略是通过计算 nowtime - 分区boundary > ttl，满足该条件的分区将被drop掉。帮助用户定时清理过期的旧数据。

说明

分区边界的设置分为以下几种情况：

- period设置为“小时”，分区起始边界值为下个小时整点，分区的间隔为period的值。
- period设置为“天”，分区起始边界值为第二天零点，分区的间隔为period的值。
- period设置为“月”，分区起始边界值为下个月零点，分区的间隔为period的值。
- period设置为“年”，分区起始边界值为明年零点，分区的间隔为period的值。

创建时序表（手动设置分区边界）

1. 手动指定分区边界的起始值，例如手动设置默认的分区边界时间P1为“2022-05-30 16:32:45”、P2为“2022-05-31 16:56:12”，创建时序表GENERATOR1如下：

```
CREATE TABLE IF NOT EXISTS GENERATOR1(
genset text TSTag,
manufacturer text TSTag,
model text TSTag,
location text TSTag,
ID bigint TSTag,
voltage numeric TSField,
power bigint TSField,
frequency numeric TSField,
angle numeric TSField,
time timestamptz TSTime) with (orientation=TIMESERIES, period='1 day') distribute by hash(model)
partition by range(time)
(
PARTITION P1 VALUES LESS THAN('2022-05-30 16:32:45'),
PARTITION P2 VALUES LESS THAN('2022-05-31 16:56:12')
);
```

2. 查询当前时间：

```
select now();
              now
-------------------------------
2022-05-31 20:36:09.700096+08(1 row)
```

3. 查询分区以及分区边界：

```
SELECT relname, boundaries FROM pg_partition where parentid=(SELECT oid FROM pg_class where
relname='generator1') order by boundaries ;
   relname   |         boundaries
-------------+----------------------------
p1           | {"2022-05-30 16:32:45+08"}
p2           | {"2022-05-31 16:56:12+08"}
p1654073772 | {"2022-06-01 16:56:12+08"}
p1654160172 | {"2022-06-02 16:56:12+08"}
......
```

4.2 冷热数据管理优秀实践

场景介绍

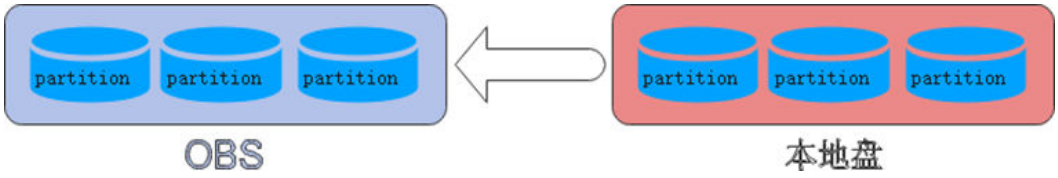
海量大数据场景下，随着业务和数据量的不断增长，数据存储与消耗的资源也日益增长。根据业务系统中用户对不同时期数据的不同使用需求，对膨胀的数据进行“冷热”分级管理，不仅可以提高数据分析性能还能降低业务成本。针对数据使用的一些场景，可以将数据按照时间分为：热数据、冷数据。

冷热数据主要从数据访问频率、更新频率进行划分。

- Hot（热数据）：访问、更新频率较高，对访问的响应时间要求很高的数据。

- Cold（冷数据）：不允许更新或更新访问频率较低，对访问的响应时间要求不高的数据。

用户可以定义冷热管理表，将符合规则的冷数据切换至OBS上进行存储，可以按照分区自动进行冷热数据的判断和迁移。



冷热切换的策略名称支持LMT（last modify time）和HPN（hot partition number），LMT指按分区的最后更新时间切换，HPN指保留热分区的个数切换。

- LMT：表示切换[day]时间前修改的热分区数据为冷分区，将该数据迁至OBS表空间中。其中[day]为整型，范围[0, 36500]，单位为天。
- HPN：表示保留HPN个有数据的分区为热分区。在冷热切换时，需要将数据迁移至OBS表空间中。其中HPN为整型，范围为[0,1600]。

约束限制

- 对于同时存在冷热分区的表，查询时会变慢，因为冷数据存储在OBS上，读写速度和时延都比在本地查询要慢。
- 目前冷热表只支持列存2.0版本的分区表，外表不支持冷热分区。
- 仅支持从热数据切换为冷数据，不支持从冷数据切换为热数据。

基本流程

本实践预计时长：30分钟，基本流程如下：

1. **创建集群**。
2. **使用gsql命令行客户端连接集群**。
3. **创建冷热表**。
4. **冷热数据切换**。
5. **查看冷热表数据分布**。

创建集群

步骤1 登录华为云管理控制台。

步骤2 在“服务列表”中，选择“大数据 > 数据仓库服务”，单击右上角“创建数据仓库集群”。

步骤3 参见**表4-2**进行参数配置。

表 4-2 软件配置

参数名称	配置方式
区域	选择“华北-北京四”。 **说明** 本指导以“华北-北京四”为例进行介绍，如果您需要选择其他区域进行操作，请确保所有操作均在同一区域进行。
可用区	可用区2
产品类型	标准数仓
CPU架构	X86
节点规格	dws2.m6.4xlarge.8（16 vCPU | 128GB | 2000GB SSD） **说明** 如规格售罄，可选择其他可用区或规格。
节点数量	3
集群名称	dws-demo
管理员用户	dbadmin
管理员密码	-
确认密码	-
数据库端口	8000
虚拟私有云	vpc-default
子网	subnet-default(192.168.0.0/24)
安全组	自动创建安全组
公网访问	现在购买
宽带	1Mbit/s
高级配置	默认配置

步骤4 信息核对无误，单击“立即购买”，单击“提交”。

步骤5 等待约6分钟，待集群创建成功后，单击集群名称前面的，弹出集群信息，记录下“公网访问地址”，例如dws-demov.dws.huaweicloud.com。

----结束

使用 gsql 命令行客户端连接集群

步骤1 使用root用户远程登录到需要安装gsql的Linux主机，然后在Linux命令窗口，执行以下命令下载gsql客户端：

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

步骤2 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3 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4 执行以下命令，使用gsql客户端连接GaussDB(DWS)集群中的数据库，其中password为用户创建集群时自定义的密码。

```
gsql -d gaussdb -p 8000 -h 192.168.0.86 -U dbadmin -W password -r
```

显示如下信息表示gsql工具已经连接成功：

```
gaussdb=>
```

----结束

创建冷热表

创建列存冷热数据管理表lifecycle_table，指定热数据有效期LMT为100天。

```
CREATE TABLE lifecycle_table(i int, val text) WITH (ORIENTATION = COLUMN, storage_policy = 'LMT:100')
PARTITION BY RANGE (i)
(
PARTITION P1 VALUES LESS THAN(5),
PARTITION P2 VALUES LESS THAN(10),
PARTITION P3 VALUES LESS THAN(15),
```



```
PARTITION P8 VALUES LESS THAN(MAXVALUE)
)
ENABLE ROW MOVEMENT;
```

冷热数据切换

切换冷数据至OBS表空间。

- 自动切换：每日0点调度框架自动触发，无需关注切换情况。

  可使用函数pg_obs_cold_refresh_time(table_name, time)自定义自动切换时间。例如，根据业务情况调整自动触发时间为每天早晨6点30分。

  ```
  SELECT * FROM pg_obs_cold_refresh_time('lifecycle_table', '06:30:00');
  pg_obs_cold_refresh_time
  --------------------------
   SUCCESS
  (1 row)
  ```

- 手动切换。

  使用ALTER TABLE语句手动切换单表：

  ```
  ALTER TABLE lifecycle_table refresh storage;
  ALTER TABLE
  ```

  使用函数pg_refresh_storage()批量切换所有冷热表：

  ```
  SELECT pg_catalog.pg_refresh_storage();
   pg_refresh_storage
  --------------------
   (1,0)
  (1 row)
  ```

查看冷热表数据分布

- 查看单表数据分布情况。

  ```
  SELECT * FROM pg_catalog.pg_lifecycle_table_data_distribute('lifecycle_table');
  schemaname |    tablename    |   nodename   | hotpartition | coldpartition | switchablepartition |
  hotdatasize | colddatasize | switchabledatasize
  ------------+-----------------+--------------+--------------+---------------+---------------------+-------------
  +--------------+--------------------
   public     | lifecycle_table | dn_6001_6002 | p1,p2,p3,p8  |               |                     | 96 KB       | 0
  bytes      | 0 bytes
   public     | lifecycle_table | dn_6003_6004 | p1,p2,p3,p8  |               |                     | 96 KB       | 0
  bytes      | 0 bytes
   public     | lifecycle_table | dn_6005_6006 | p1,p2,p3,p8  |               |                     | 96 KB       | 0
  bytes      | 0 bytes
  (3 rows)
  ```

- 查看所有冷热表数据分布情况。

  ```
  SELECT * FROM pg_catalog.pg_lifecycle_node_data_distribute();
  schemaname |    tablename    |   nodename   | hotpartition | coldpartition | switchablepartition |
  hotdatasize | colddatasize | switchabledatasize
  ------------+-----------------+--------------+--------------+---------------+---------------------+-------------
  +--------------+--------------------
   public     | lifecycle_table | dn_6001_6002 | p1,p2,p3,p8  |               |                     |       98304 |
  0 |                0
   public     | lifecycle_table | dn_6003_6004 | p1,p2,p3,p8  |               |                     |       98304 |
  0 |                0
   public     | lifecycle_table | dn_6005_6006 | p1,p2,p3,p8  |               |                     |       98304 |
  0 |                0
  (3 rows)
  ```

4.3 分区自动管理优秀实践

场景介绍

对于分区列为时间的分区表，分区自动管理功能可以自动创建新分区和删除过期分区，降低分区表的维护成本，改善查询性能。为了便于查询和维护数据，用户通常使用分区列为时间的分区表来存储时间相关的数据，例如电商的订单信息、 物联网采集的实时数据。这些时间相关的数据导入分区表时，需要保证分区表要有对应时间的分区，由于普通的分区表不会自动创建新的分区和删除过期的分区，所以维护人员需要定期创建新分区和删除过期分区，提高了运维成本。

为解决上述问题，GaussDB(DWS) 引入了分区自动管理特性。可通过设置表级参数period、ttl开启分区自动管理功能，使分区表可以自动创建新分区和删除过期分区，降低分区表的维护成本，改善查询性能。

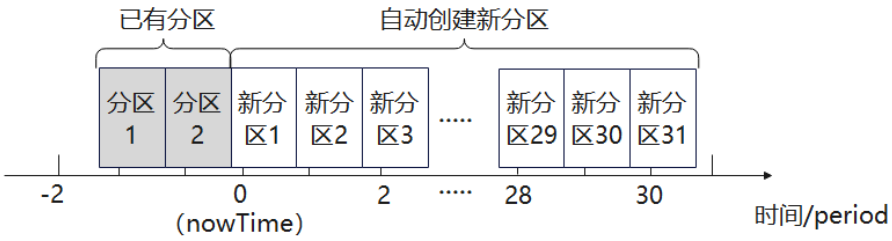
period：设置自动创建分区的间隔时间，默认值为1 day，取值范围：1 hour ~ 100 years。

ttl：设置自动淘汰分区的时间，取值范围：1 hour ~ 100 years。淘汰分区的策略是通过计算 nowtime - 分区boundary > ttl，满足该条件的分区将被清理掉。

- 自动创建新分区

  分区自动管理每隔period的时间就会自动创建分区，每次创建一个或多个时间范围为period的新分区，以推进最大的分区边界时间，保证其大于nowTime +30*period。由于每次创建分区时，都动态地为未来时间创建了预留分区，所以只要有一次自动创建新分区成功，就可以保证在未来30个period的时间之内，都不会出现实时数据因为没有对应分区而导入失败的情况。

  图 4-3 自动创建分区示意图

- 自动删除过期分区

  边界时间早于nowTime-ttl的分区被认为是过期分区。分区自动管理每隔period的时间就会遍历检测所有分区，并删除其中的过期分区，如果所有的分区都是过期分区，则保留一个分区，并TRUNCATE该表。

约束限制

在使用分区管理功能时，需要满足如下约束：

- 不支持在小型机、加速集群、单机集群上使用。
- 支持在8.1.3及以上集群版本中使用。
- 仅支持行存范围分区表、列存范围分区表、时序表以及冷热表。

- 分区键唯一且类型仅支持timestamp、timestamptz、date类型。
- 不支持存在maxvalue分区。
- (nowTime - boundaryTime) / period需要小于分区个数上限，其中nowTime为当前时间，boundaryTime为现有分区中最早的分区边界时间。
- period、ttl取值范围为1hour ~ 100years。另外，在兼容Teradata或MySQL的数据库中，分区键类型为date时，period不能小于1day。
- 表级参数ttl不支持单独存在，必须要提前或同时设置period，并且要大于或等于period。
- 集群在线扩容期间，自动增加分区会失败，但是由于每次增分区时，都预留了足够的分区，所以不影响使用。

创建 ECS

参见**自定义购买弹性云服务器**购买。购买后，参见**登录Linux弹性云服务器**进行登录。

须知

创建ECS过程中，注意选择与后续的IoT数仓在同一个区域、可用区和同一个VPC子网下，ECS的操作系统选择与gsql客户端（本例以CentOS 7.6为例），并选择以密码方式登录。

创建集群

步骤1 登录华为云管理控制台。

步骤2 在“服务列表”中，选择“大数据 > 数据仓库服务”，单击右上角“创建数据仓库集群”。

步骤3 参见**表4-3**进行参数配置。

表 4-3 软件配置

参数名称	配置方式
区域	选择“华北-北京四”。 **说明** 本指导以“华北-北京四”为例进行介绍，如果您需要选择其他区域进行操作，请确保所有操作均在同一区域进行。
可用区	可用区2
产品类型	标准数仓
CPU架构	X86
节点规格	dws2.m6.4xlarge.8（16 vCPU | 128GB | 2000GB SSD） **说明** 如规格售罄，可选择其他可用区或规格。
节点数量	3

参数名称	配置方式
集群名称	dws-demo
管理员用户	dbadmin
管理员密码	-
确认密码	-
数据库端口	8000
虚拟私有云	vpc-default
子网	subnet-default(192.168.0.0/24)
安全组	自动创建安全组
公网访问	现在购买
宽带	1Mbit/s
高级配置	默认配置

步骤4 信息核对无误，单击“立即购买”，单击“提交”。

步骤5 等待约6分钟，待集群创建成功后，单击集群名称前面的，弹出集群信息，记录下“公网访问地址”，例如dws-demov.dws.huaweicloud.com。

----结束

使用 gsql 命令行客户端连接集群

步骤1 使用root用户远程登录到需要安装gsql的Linux主机，然后在Linux命令窗口，执行以下命令下载gsql客户端：

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

步骤2 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3 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4 执行以下命令，使用gsql客户端连接GaussDB(DWS)集群中的数据库，其中password为用户创建集群时自定义的密码。

```
gsql -d gaussdb -p 8000 -h 192.168.0.86 -U dbadmin -W password -r
```

显示如下信息表示gsql工具已经连接成功：

```
gaussdb=>
```

----结束

分区自动管理

分区管理功能是和表级参数period、ttl绑定的，只要成功设置了表级参数period，即开启了自动创建新分区功能；成功设置了表级参数ttl，即开启了自动删除过期分区功能。第一次自动创建分区或删除分区的时间为设置period或ttl后30秒。

有两种开启分区管理功能的方式，具体如下：

- 建表时指定period、ttl。

  该方式适用于新建分区管理表时使用。新建分区管理表有两种语法，一种是建表时指定分区，另一种是建表时不指定分区。

  建分区管理表时如果指定分区，则语法规则和建普通分区表相同，唯一的区别就是会指定表级参数period、ttl。

  示例：创建分区管理表CPU1，指定分区。

  ```
  CREATE TABLE CPU1(
      id integer,
      IP text,
      time timestamp
  ) with (TTL='7 days',PERIOD='1 day')
  partition by range(time)
  (
      PARTITION P1 VALUES LESS THAN('2023-02-13 16:32:45'),
      PARTITION P2 VALUES LESS THAN('2023-02-15 16:48:12')
  );
  ```

  建分区管理表时可以只指定分区键不指定分区，此时将创建两个默认分区，这两个默认分区的分区时间范围均为period。其中，第一个默认分区的边界时间是大于当前时间的第一个整时/整天/整周/整月/整年的时间，具体选择哪种整点时间取决于period的最大单位；第二个默认分区的边界时间是第一个分区边界时间加period。假设当前时间是2023-02-17 16:32:45，各种情况的第一个默认分区的分区边界选择如下表：

  表 4-4 period 参数说明

period	period最大单位	第一个默认分区的分区边界
1hour	Hour	2023-02-17 17:00:00

period	period最大单位	第一个默认分区的分区边界
1day	Day	2023-02-18 00:00:00
1month	Month	2023-03-01 00:00:00
13months	Year	2024-01-01 00:00:00

创建分区管理表CPU2，不指定分区：

```
CREATE TABLE CPU2(
    id integer,
    IP text,
    time timestamp
) with (TTL='7 days',PERIOD='1 day')
partition by range(time);
```

- 使用ALTER TABLE RESET的方式设置period、ttl。

  该方式适用于给一张满足分区管理约束的普通分区表增加分区管理功能。

  - 创建普通分区表CPU3：

    ```
    CREATE TABLE CPU3(
        id integer,
        IP text,
        time timestamp
    )
    partition by range(time)
    (
        PARTITION P1 VALUES LESS THAN('2023-02-14 16:32:45'),
        PARTITION P2 VALUES LESS THAN('2023-02-15 16:56:12')
    );
    ```

  - 同时开启自动创建和自动删除分区功能：

    ```
    ALTER TABLE CPU3 SET (PERIOD='1 day',TTL='7 days');
    ```

  - 只开启自动创建分区功能：

    ```
    ALTER TABLE CPU3 SET (PERIOD='1 day');
    ```

  - 只开启自动删除分区功能，如果没有提前开启自动创建分区功能，则开启失败：

    ```
    ALTER TABLE CPU3 SET (TTL='7 days');
    ```

  - 通过修改period和ttl修改分区管理功能：

    ```
    ALTER TABLE CPU3 SET (TTL='10 days',PERIOD='2 days');
    ```

- 关闭分区管理功能。

  使用ALTER TABLE RESET语句可以删除表级参数period、ttl，进而关闭相应的分区管理功能。

  说明

  - 不能在存在ttl的情况下，单独删除period。
  - 时序表不支持ALTER TABLE RESET。

  - 同时关闭自动创建和自动删除分区功能：

    ```
    ALTER TABLE CPU1 RESET (PERIOD,TTL);
    ```

  - 只关闭自动删除分区功能：

    ```
    ALTER TABLE CPU3 RESET (TTL);
    ```

  - 只关闭自动创建分区功能，如果该表有ttl参数，则关闭失败：

    ```
    ALTER TABLE CPU3 RESET (PERIOD);
    ```


4.4 GaussDB(DWS)视图解耦与自动重建

为了解决视图和表依赖而无法单独修改基表对象的问题，GaussDB(DWS)实现了视图的解耦与重建功能。本文重点介绍视图自动重建功能的使用场景与使用方法。

场景介绍

GaussDB(DWS)使用对象标识符（oid）来保存对象之间的引用关系，这使得视图在定义的时候就绑定了其依赖的数据库对象的oid，不管视图名称怎么改变，都不会改变这层依赖关系。如果要对基表进行一些字段修改，会因为与视图字段存在强绑定而报错。如果要删除某个表字段或整个表，就需要连同其关联的视图一起使用cascade关键字删除，表字段删除完成或表重建后再依次重建各级视图，给用户的使用增加了很大的工作量，导致易用性较差。

为了解决这一问题，GaussDB(DWS)在8.1.0集群版本实现了视图的解耦，使得存在视图依赖的基表或其他数据库对象（视图、同义词、函数、表字段）可以单独删除，而其对应对象上关联的依赖视图依然存在，而在基表重建后，可以通过ALTER VIEW REBUILD命令重建依赖关系。而8.1.1集群版本在此基础上又实现了自动重建，可以无感知自动重建依赖关系，开启自动重建后会有锁冲突，因此不建议用户开启自动重建。

使用方法

步骤1 在管理控制台上创建集群，具体操作步骤请参考**创建集群**。

步骤2 打开GUC参数view_independent参数。

视图解耦功能由GUC参数view_independent进行控制，默认关闭。使用时需要用户手动打开，可登录管理控制台后，单击集群名称，进入“集群详情”页面，单击“参数修改”页签，并在“参数列表”模块搜索view_independent参数，修改后保存。

步骤3 使用DAS连接集群。在集群列表中找到所需要的集群，单击“操作”栏中的“登录”按钮，跳转至数据库管理服务（DAS）页面，填写登录用户名、数据库名称、密码信息后测试连接，测试无误后登录集群。更多详细步骤请参考**使用DAS连接集群**。

步骤4 创建示例表t1并插入数据。

```
SET current_schema='public';
CREATE TABLE t1 (a int, b int, c char(10)) DISTRIBUTE BY HASH (a);
INSERT INTO t1 VALUES(1,1,'a'),(2,2,'b');
```

步骤5 创建视图v1依赖表t1，创建视图v11依赖视图v1。查询视图v11。

```
CREATE VIEW v1 AS SELECT a, b FROM t1;
CREATE VIEW v11 AS SELECT a FROM v1;

SELECT * FROM v11;
 a
---
 1
 2
(2 rows)
```

步骤6 删除表t1后，查询视图v11会因表t1不存在而报错，但视图是依旧存在的。

GaussDB(DWS)提供GS_VIEW_INVALID视图查询当前用户可见的所有不可用的视图。如果该视图依赖的基础表或函数或同义词存在异常，该视图validtype列显示为“invalid”。

```
DROP TABLE t1;

SELECT * FROM v11;
ERROR:  relation "public.t1" does not exist

SELECT * FROM gs_view_invalid;
  oid   | schemaname | viewname | viewowner |          definition          | validtype
--------+------------+----------+-----------+------------------------------+-----------
 213563 | public     | v1       | dbadmin   | SELECT a, b FROM public.t1; | invalid
 213567 | public     | v11      | dbadmin   | SELECT a FROM public.v1;    | invalid
(2 rows)
```

步骤7 重建表t1后，视图自动重建。视图只有使用才能自动刷新。

```
CREATE TABLE t1 (a int, b int, c char(10)) DISTRIBUTE BY HASH (a);
INSERT INTO t1 VALUES(1,1,'a'),(2,2,'b');

SELECT * from v1;
 a | b
---+---
 1 | 1
 2 | 2
(2 rows)

SELECT * FROM gs_view_invalid;
  oid   | schemaname | viewname | viewowner |         definition         | validtype
--------+------------+----------+-----------+----------------------------+-----------
 213567 | public     | v11      | dbadmin   | SELECT a FROM public.v1; | invalid
(1 row)

SELECT * from v11;
 a
---
 1
 2
(2 rows)

SELECT * FROM gs_view_invalid;
 oid | schemaname | viewname | viewowner | definition | validtype
-----+------------+----------+-----------+------------+-----------
(0 rows)
```

----结束

4.5 列存 delta 表优秀实践

基本原理

在GaussDB(DWS)中，列存表按列存储数据，每列默认60000行存储在一个CU中，CU是列存表存储数据的最小单元，CU生成后数据固定不可更改。无论是向列存表中插入

1条还是60000条数据，都只会生成一个CU，在多次插入少量数据时，不能有效的利用列存压缩能力，从而导致数据膨胀影响查询的性能和磁盘使用率。

由于CU文件数据不能更改只能追加写，对CU中的数据做更新或删除都不会真正更改这个CU，删除是将老数据在字典中标记为作废，更新操作是标记老数据删除后，再写入一条新记录到新CU，CU不会有任何的更改。在对列存表进行多次更新/删除，或每次只插入少量数据后，会导致列存表空间膨胀，大量空间无法有效利用。

因为列存表在设计上就是为了大批量数据导入以及海量数据按列存储/查询。为了解决上述问题，引入了列存delta表就是列存表附带的行存表。在启用delta表后，单条或者小批量数据导入时，数据将进入delta表中，避免小CU的产生，delta表的增删改查与行存表一致。开启delta表后，将显著提升列存表单条导入的性能。

使用场景

列存delta表应用于行列混合存储，适合实时分析统计，解决了实时小批量数据入库引起的性能问题，定期合并到主表保证分析查询的性能。在实际使用时需要根据业务场景来评估是否需要开启delta表，否则不仅无法充分发挥GaussDB(DWS)列存表的优势，反而会造成额外的空间和时间的浪费。

准备工作

- 已注册账号，且在使用GaussDB(DWS) 前检查账号状态，账号不能处于欠费或冻结状态。
- 获取此账号的“AK/SK”。
- 参考**交通卡口通行车辆分析**，GaussDB(DWS) 已预先将样例数据上传到OBS桶的“traffic-data”文件夹中，并给所有华为云用户赋予了该OBS桶的只读访问权限。

操作步骤

步骤1 使用DAS连接集群。在集群列表中找到所需要的集群，单击“操作”栏中的“登录”按钮，跳转至数据库管理服务（DAS）页面，填写登录用户名、数据库名称、密码信息后测试连接，测试无误后登录集群。更多详细步骤请参考**使用DAS连接集群**。

步骤2 执行以下语句，创建traffic数据库。

```
CREATE DATABASE traffic encoding 'utf8' template template0;
```

步骤3 执行以下语句，分别创建用于存储卡口车辆信息的数据库表GCJL和GCJL2。其中GCJL默认不开启delta表，GCJL2为开启delta表。

```
CREATE SCHEMA traffic_data;
SET current_schema= traffic_data;
DROP TABLE if exists GCJL;
CREATE TABLE GCJL
(
    kkbh   VARCHAR(20),
    hphm   VARCHAR(20),
    gcsj   DATE ,
    cplx   VARCHAR(8),
    cllx   VARCHAR(8),
```



```
    csys   VARCHAR(8)
)
with (orientation = column, COMPRESSION=MIDDLE)
distribute by hash(hphm);

DROP TABLE if exists GCJL2;
CREATE TABLE GCJL2
(
    kkbh   VARCHAR(20),
    hphm   VARCHAR(20),
    gcsj   DATE ,
    cplx   VARCHAR(8),
    cllx   VARCHAR(8),
    csys   VARCHAR(8)
)
with (orientation = column, COMPRESSION=MIDDLE, ENABLE_DELTA = TRUE)
distribute by hash(hphm);
```

说明

- Delta表默认是关闭的，若需要开启delta表，可以在创建列存表时指定enable_delta为true。
- 如果未开启delta表，也可以执行如下命令开启delta表：
  ```
  ALTER TABLE table_name SET (enable_delta=TRUE);
  ```
- 如果已开启delta表，需要关闭delta表时可以执行如下命令：
  ```
  ALTER TABLE table_name SET (enable_delta=FALSE);
  ```

步骤4 创建外表。外表用于识别和关联OBS上的源数据。

须知

- 其中，*<obs_bucket_name>*代表OBS桶名，仅支持部分区域，当前支持的区域和对应的OBS桶名请参见**支持区域**。GaussDB(DWS) 集群不支持跨区域访问OBS桶数据。
- 本实践以“华北-北京四”地区为例，可填入dws-demo-cn-north-4，<Access_Key_Id>和<Secret_Access_Key>替换为实际值。
- 创建外表如果提示“ERROR: schema "*xxx*" does not exist Position”，则说明schema不存在，请先参照上一步创建schema。

```
DROP FOREIGN table if exists GCJL_OBS;
CREATE FOREIGN TABLE GCJL_OBS
(
    like traffic_data.GCJL
)
SERVER gsmpp_server
OPTIONS (
    encoding 'utf8',
    location 'obs://<obs_bucket_name>/traffic-data/gcxx',
    format 'text',
    delimiter ',',
    access_key '<Access_Key_Id>',
    secret_access_key '<Secret_Access_Key>',
    chunksize '64',
    IGNORE_EXTRA_DATA 'on'
);
```

步骤5 执行以下语句，将数据从外表导入到数据库表中。

```
INSERT INTO traffic_data.GCJL select * from GCJL_OBS;
INSERT INTO traffic_data.GCJL2 select * from GCJL_OBS;
```

导入数据需要一些时间，请耐心等待。

步骤6 执行以下语句，查看导入数据库表后的存储空间大小。

```
SELECT pg_size_pretty(pg_total_relation_size('traffic_data.GCJL'));
SELECT pg_size_pretty(pg_total_relation_size('traffic_data.GCJL2'));
```

通过对比两种场景下的列存表，开启delta表后，存储空间由8953 MB减少为6053 MB，大大提升了存储能力。

步骤7 执行以下语句，查询表中数据，对比之后可以发现开启delta表后查询速度有所提升。

```
SELECT * FROM traffic_data.GCJL where hphm =  'YD38641';
SELECT * FROM traffic_data.GCJL2 where hphm =  'YD38641';
```

----结束

开启 Delta 表的影响

- 开启列存表的delta表功能，在导入单条或者小规模数据进入表中时，能够防止小CU的产生，所以开启delta表能够带来显著的性能提升，例如在3CN、6DN的集群上操作，每次导入100条数据，导入时间能减少25%，存储空间减少97%，所以在需要多次插入小批量数据前应该先开启delta表，等到确定接下来没有小批量数据导入了再关闭。
- Delta表就是列存表附带的行存表，那么将数据插入delta表后将失去列存表的高压缩比等优势，正常情况下使用列存表的场景都是大批量数据导入，所以默认关闭delta表，如果开启delta表做大批量数据导入，反而会额外消耗更多时间和空间，同样在3CN、6DN的集群上操作，每次导入10000条数据时，开启delta表会比不开启时慢4倍，额外消耗10倍以上的空间。所以开启delta表需谨慎，根据实际业务需要来选择开启和关闭。

4.6 GIN 索引使用实践

GIN是一个存储对（key、posting list）集合的索引结构，其中key是一个键值，posting list是一组出现过key的位置。如‘hello', '14:2 23:4'中，表示hello在14:2和23:4这两个位置出现过。通过GIN索引结构可以快速的查找到包含指定关键字的元组，因此GIN索引适用于多值类型的元素搜索。本章节将介绍如何使用GIN索引查询数组类型、JSONB类型，如何进行全文检索。

使用 GIN 索引查询数组类型

创建一个GIN索引来加快对标签进行搜索的查询。

步骤1 在管理控制台上创建集群，具体操作步骤请参考**创建集群**。

步骤2 使用DAS连接集群。在集群列表中找到所需要的集群，单击“操作”栏中的“登录”按钮，跳转至数据库管理服务（DAS）页面，填写登录用户名、数据库名称、密码信息后测试连接，测试无误后登录集群。更多详细步骤请参考**使用DAS连接集群**。

步骤3 创建表books，其中列tags存储了书籍的标签信息，使用数组类型来表示。

```
CREATE TABLE books (id SERIAL PRIMARY KEY, title VARCHAR(100), tags TEXT[]);
```

步骤4 插入数据。

```
INSERT INTO books (title, tags)
VALUES   ('Book 1', ARRAY['fiction', 'adventure']),
  ('Book 2', ARRAY['science', 'fiction']),
  ('Book 3', ARRAY['romance', 'fantasy']),
  ('Book 4', ARRAY['adventure']);
```

步骤5 创建GIN索引。

```
CREATE INDEX idx_books_tags_gin ON books USING GIN (tags);
```

步骤6 使用GIN索引执行搜索查询，以便在tags列中查找包含特定标签的书籍。查找包含标签“fiction”的书籍：

```
SELECT * FROM books WHERE tags @> ARRAY['fiction'];
 id | title  |        tags
----+--------+---------------------
  1 | Book 1 | {fiction,adventure}
  2 | Book 2 | {science,fiction}
(2 rows)
```

步骤7 使用GIN索引查找同时包含“fiction”和“adventure”标签的书籍记录：

```
SELECT * FROM books WHERE tags @> ARRAY['fiction', 'adventure'];
 id | title  |        tags
----+--------+---------------------
  1 | Book 1 | {fiction,adventure}
(1 row)
```

----结束

使用 GIN 索引查询 JSONB 类型

当使用JSONB数据类型存储和查询JSON数据时，可以使用GIN索引来提高查询性能。GIN索引适用于查询包含大量不同的键值对的JSONB列。

步骤1 创建表my_table，其中列data存储了每个人的相关信息，使用JSONB类型来表示。

```
CREATE TABLE my_table (id SERIAL PRIMARY KEY, data JSONB);
```

步骤2 插入数据。

```
INSERT INTO my_table (data)
 VALUES ('{"name": "John", "age": 30, "address": {"career": "announcer", "state": "NY"}}'),
     ('{"name": "Alice", "age": 25, "address": {"career": "architect", "state": "CA"}}'),
     ('{"name": "Bob", "age": 35, "address": {"career": "dentist", "state": "WA"}}');
```

步骤3 创建一个GIN索引来加速JSONB列的查询。

```
CREATE INDEX my_table_data_gin_index ON my_table USING GIN (data);
```

步骤4 使用GIN索引来执行JSONB列的查询。例如，查找职业是建筑师的人：

```
SELECT * FROM my_table WHERE data @> '{"address": {"career": "dentist"}}';
 id |                                data
```



```
----+--------------------------------------------------------------------------
  3 | {"age": 35, "name": "Bob", "address": {"state": "WA", "career": "dentist"}}
(1 row)
```

步骤5 GIN索引还可以在JSONB列的键上进行查询。例如，查找年龄大于等于30岁的人：

```
SELECT * FROM my_table WHERE data ->> 'age' >= '30';
 id |                                 data
----+--------------------------------------------------------------------------
  3 | {"age": 35, "name": "Bob", "address": {"state": "WA", "career": "dentist"}}
  1 | {"age": 30, "name": "John", "address": {"state": "NY", "career": "announcer"}}
(2 rows)
```

----结束

使用 GIN 索引全文检索

当使用GIN索引进行全文搜索时，可以使用tsvector和tsquery数据类型以及相关的函数来实现。

步骤1 创建articles表，其中列content存储了文章的内容。

```
CREATE TABLE articles (id SERIAL PRIMARY KEY,title VARCHAR(100),content TEXT);
```

步骤2 插入数据。

```
INSERT INTO articles (title, content)
 VALUES   ('Article 1', 'This is the content of article 1.'),
   ('Article 2', 'Here is the content for article 2.'),
   ('Article 3', 'This article discusses various topics.'),
   ('Article 4', 'The content of the fourth article is different.');
```

步骤3 为content列创建一个辅助列tsvector，该列将存储已处理的文本索引。

```
ALTER TABLE articles ADD COLUMN content_vector tsvector;
```

步骤4 更新content_vector列的值，将content列的文本转换为tsvector类型。

```
UPDATE articles SET content_vector = to_tsvector('english', content);
```

步骤5 创建GIN索引。

```
CREATE INDEX idx_articles_content_gin ON articles USING GIN (content_vector);
```

步骤6 执行全文搜索查询，使用tsquery类型来指定搜索条件。例如，查找包含单词“content”的文章：

```
SELECT * FROM articles WHERE content_vector @@ to_tsquery('english', 'content');
```

----结束

⚠ 注意

要构建一个tsquery对象，需要使用to_tsquery函数，并提供搜索条件和相应的文本搜索配置（在本例中为english）。

还可以使用其他文本搜索函数和操作符来进行更复杂的全文搜索查询，例如plainto_tsquery、ts_rank 等。具体的用法取决于实际的需求。

5 数据库管理

5.1 资源管理优秀实践

本实践将演示GaussDB(DWS)的资源管理功能，帮助企业客户解决数据分析过程中，多用户查询作业遇到的性能瓶颈，最终实现多用户执行SQL作业互不影响，节省资源消耗。

本实践预计时长60分钟，基本流程如下：

1. **步骤一：创建集群**
2. **步骤二：连接集群并导入数据**
3. **步骤三：创建资源池**
4. **步骤四：异常规则验证**

场景介绍

当有多个数据库用户同时在GaussDB(DWS)上执行SQL作业时，可能出现以下情况：

1. 一些复杂SQL可能会长时间占用集群资源，从而影响其他查询的性能。例如一组数据库用户不断提交复杂、耗时的查询，而另一组用户经常提交短查询。在这种情况下，短时查询可能不得不在资源池中等待耗时查询完成。
2. 一些SQL由于数据倾斜、执行计划未调优等原因，占用过多内存空间，导致其他语句因申请不到内存而报错，或占用过多磁盘空间，导致磁盘满而触发集群只读，无法进行写入。

为了提高系统整体吞吐量，避免坏SQL影响系统整体运行，您可以使用GaussDB(DWS)工作负载管理功能处理这类问题，例如，将经常提交复杂查询作业的数据库用户分为一类，为这类用户创建一个资源池并给这个资源池分配多一些的资源，之后将这类用户添加至这个资源池中，那么这类用户所提交的复杂作业只能使用所创建资源池拥有的资源；同时再创建一个占用资源较少的资源池分配给执行短查询的用户使用，这样两种作业就能够同时执行互不影响。

以xx银行为例，该银行业务场景主要分为联机交易（OLTP）和报表分析（OLAP）两大类，其中报表服务的优先级相对较低，在合理的情况下优先保障业务系统的正常运行。业务系统中运行的SQL分为简单SQL和复杂SQL，大量复杂SQL的并发执行会导致数据库服务器资源争抢，简单SQL的大量并发对服务器不构成持续压力，短时间内可执行完成，不会造成业务堆积。其中报表服务中运行的SQL以复杂SQL居多，整体业务

逻辑相对复杂，在数据库层面需要分别对核心交易和报表服务进行合理的资源管控，以保障业务系统正常运行。

报表分析类业务的优先级和实时性相对较低，但是复杂度更高，为有效进行资源管控，将报表分析和核心交易业务进行数据库用户分离，例如核心交易业务使用数据库用户**budget_config_user**，报表分析业务使用数据库用户**report_user**。针对交易用户和报表用户分别进行CPU资源和并发数控制以保障数据库稳定运行。

结合报表分析业务的负载调研、日常监控和测试验证，50并发以内的复杂报表SQL不会引起服务器资源争抢，不会引起业务系统卡慢，配置报表用户可使用20%的CPU资源。

结合核心交易业务的负载调研、日常监控和测试验证，100并发以内的查询SQL不会对系统造成持续压力，配合交易用户可使用60%的CPU资源。

- 报表用户资源配置（对应资源池pool_1）：CPU=20%，内存=20%，存储=1024000MB，并发=20。
- 交易用户资源配置（对应资源池pool_2）：CPU=60%，内存=60%，存储=1024000MB，并发=200。

设置单个语句最大内存使用量，超过使用量则报错退出，避免单个语句占用过多内存。

异常规则中设置阻塞时间=1200S，执行所消耗时间1800s，强制终止。

步骤一：创建集群

参见**创建集群**完成集群创建。

步骤二：连接集群并导入数据

步骤1 参见**使用gsql命令行客户端连接集群**连接集群。

步骤2 导入样例数据。参见**导入TPC-H数据**。

步骤3 执行以下语句创建核心交易用户**budget_config_user**和报表用户**report_user**。

```
CREATE USER budget_config_user PASSWORD 'password';
CREATE USER report_user PASSWORD 'password';
```

步骤4 为测试需要，将tpch模式下所有表的所有权限授予两个用户。

```
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA tpch to budget_config_user,report_user;
```

步骤5 查看当前两个用户的资源分配情况。

```
SELECT * FROM PG_TOTAL_USER_RESOURCE_INFO where username in ('budget_config_user' , 'report_user');
```

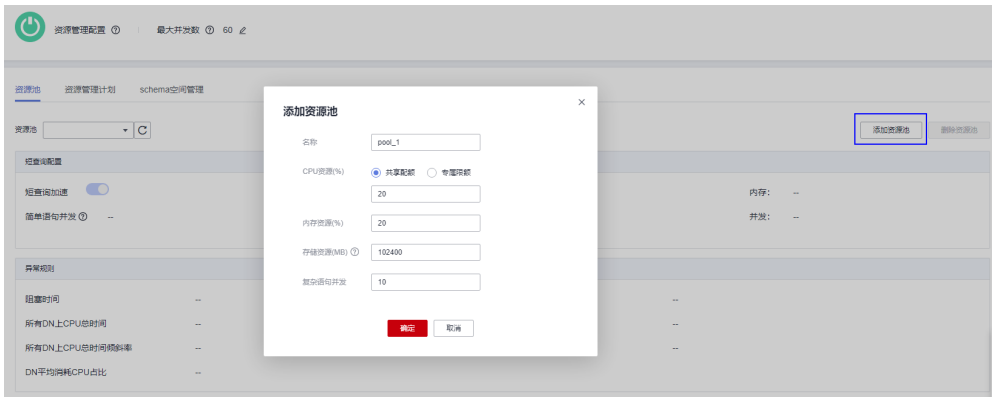
```
tpch=> SELECT * FROM PG_TOTAL_USER_RESOURCE_INFO where username in ('budget_config_user' , 'report_user');
      username      | used_memory | total_memory | used_cpu | total_cpu | used_space | total_space | used_temp_space | total_temp_space | used_spill_space | total_spill_space | read_kbytes | write_kbyte
s | read_counts | write_counts | read_speed | write_speed
--------------------+-------------+--------------+----------+-----------+------------+-------------+-----------------+------------------+------------------+-------------------+-------------+-----------
--+-------------+--------------+------------+-------------
 budget_config_user |           0 |        10796 |        0 |         8 |          0 |          -1 |               0 |               -1 |                0 |                -1 |           0 |
0 |           0 |            0 |          0 |         0 |          0 |          -1 |               0 |               -1 |                0 |                -1 |           0 |
 report_user       |           0 |        10796 |        0 |         8 |          0 |          -1 |               0 |               -1 |                0 |                -1 |           0 |
0 |           0 |            0 |          0 |         0 |          0 |          -1 |               0 |               -1 |                0 |                -1 |           0 |
(2 rows)
```

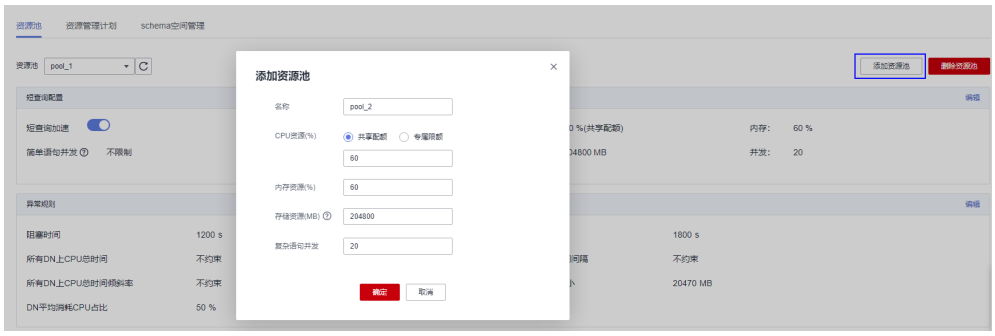
----结束

步骤三：创建资源池

步骤1 登录GaussDB(DWS) 管理控制台，在集群列表中单击集群名称，切换至“资源管理”页签。

步骤2 单击“添加资源池”创建资源池。参见**场景介绍**的模型分别创建报表业务资源池**pool_1**和核心交易资源池**pool_2**。



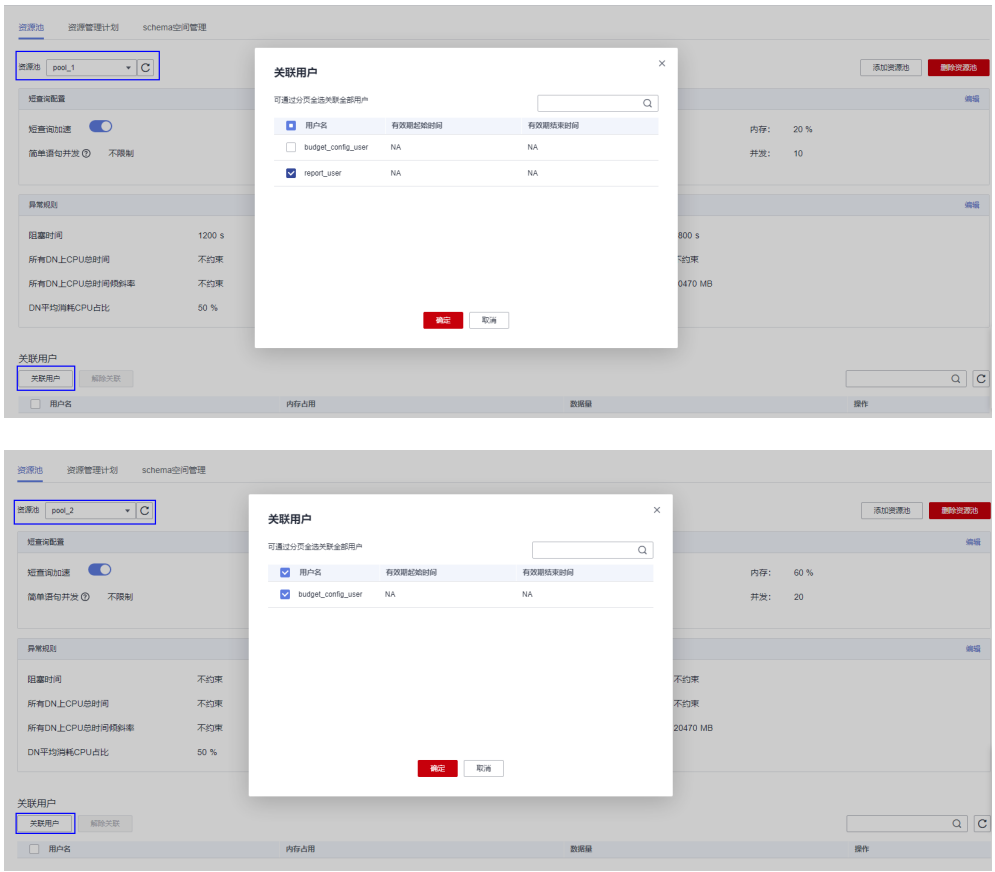


步骤3 修改异常规则。

1. 单击创建好的pool_1资源池。
2. 在异常规则中，修改“阻塞时间”和“执行所消耗时间”分别为1200s和1800s。
3. 单击“保存”。
4. 重复以上步骤，修改pool_2。

步骤4 关联用户。

1. 左侧单击“pool_1”资源池。
2. 单击“关联用户”右侧的“添加”。
3. 勾选报表业务对应的用户**report_user**，单击“确定”。
4. 重复以上步骤，将核心交易的用户**budget_config_user**添加入pool_2的资源池中。



----结束

步骤四：异常规则验证

步骤1 使用用户report_user登录数据库。

步骤2 执行如下命令查看用户report_user所属资源池。

```
SELECT usename,respool FROM pg_user WHERE usename = 'report_user';
```

```
gaussdb=> select usename,respool from pg_user where usename = 'report_user';
   usename   | respool
-------------+---------
 report_user | pool_1
(1 row)
```

查询显示用户report_user所属资源池为pool_1。

步骤3 校验资源池pool_1所绑定的异常规则。

```
SELECT respool_name,mem_percent,active_statements,except_rule FROM pg_resource_pool WHERE respool_name='pool_1';
```

```
gaussdb=> select respool_name,mem_percent,active_statements,except_rule from pg_resource_pool where respool_name='pool_1';
 respool_name | mem_percent | active_statements | except_rule
--------------+-------------+-------------------+-------------
 pool_1       |          20 |                20 | rule_1
(1 row)
```

确认资源池pool_1所绑定的是异常规则rule_1。

步骤4 查看当前用户异常规则的规则类型和阈值。

```
SELECT * FROM pg_except_rule WHERE name = 'rule_1';
```

```
gaussdb=> select * from pg_except_rule where name = 'rule_1';
  name   |     rule     | value
---------+--------------+-------
 rule_1 | action       | abort
 rule_1 | blocktime    | 1200
 rule_1 | elapsedtime | 1800
(3 rows)
```

查询显示rule_1中所绑定的规则为**步骤3**设置的“阻塞时间1200秒，运行时长1800秒”，则会终止查询。

须知

- PG_EXCEPT_RULE系统表存储关于异常规则的信息，该系统表仅8.2.0及以上集群版本支持。
- 同一条异常规则内的参数项相互之间为且的关系。

步骤5 执行作业，当运行时长超过“阻塞时间1200秒，运行时长1800秒”时，报错提示作业被取消并提示所触发的异常规则限制。

```
gaussdb=> insert into mytable select * from table1;
ERROR: canceling statement due to workload manager exception.
DETAIL: except rule [rule_1] is meet condition: rule [elapsedtime] is over limit. current value is: 1800. rule [blocktime] is over limit. current value is: 1200.
```

作业执行过程中，如果出现类似“ERROR: canceling statement due to workload manager exception.”的报错信息，表示该作业超过异常规则的规则阈值限制被终止。若规则设置合理，那么就需要考虑从业务角度进行语句优化，减少执行时间。

----结束

5.2 SQL 查询优秀实践

根据数据库的SQL执行机制以及大量的实践总结发现：通过一定的规则调整SQL语句，在保证结果正确的基础上，能够提高SQL执行效率。

- **使用union all代替union**

  union在合并两个集合时会执行去重操作，而union all则直接将两个结果集合并、不执行去重。执行去重会消耗大量的时间，因此，在一些实际应用场景中，如果通过业务逻辑已确认两个集合不存在重叠，可用union all替代union以便提升性能。

- **join列增加非空过滤条件**

  若join列上的NULL值较多，则可以加上is not null过滤条件，以实现数据的提前过滤，提高join效率。

- **not in转not exists**

  not in语句需要使用nestloop anti join来实现，而not exists则可以通过hash anti join来实现。在join列不存在null值的情况下，not exists和not in等价。因此在确保没有null值时，可以通过将not in转换为not exists，通过生成hash join来提升查询效率。

  如下所示，如果t2.d2字段中没有null值(t2.d2字段在表定义中not null)查询可以修改为

  ```
  SELECT * FROM t1 WHERE  NOT EXISTS (SELECT * FROM t2 WHERE t1.c1=t2.d2);
  ```

产生的计划如下：

图 5-1 not exists 执行计划

```
 id |                 operation
----+-------------------------------------------
  1 | ->  Streaming (type: GATHER)
  2 |     ->  Hash Right Anti Join (3, 5)
  3 |        ->  Streaming(type: REDISTRIBUTE)
  4 |           ->  Seq Scan on t2
  5 |        ->  Hash
  6 |           ->  Seq Scan on t1

Predicate Information (identified by plan id)
---------------------------------------------
  2 --Hash Right Anti Join (3, 5)
        Hash Cond: (t2.d2 = t1.c1)
(13 rows)
```

- **选择hashagg。**

  查询中GROUP BY语句如果生成了groupagg+sort的plan性能会比较差，可以通过加大work_mem的方法生成hashagg的plan，因为不用排序而提高性能。

- **尝试将函数替换为case语句。**

  GaussDB(DWS)函数调用性能较低，如果出现过多的函数调用导致性能下降很多，可以根据情况把可下推函数的函数改成CASE表达式。

- **避免对索引使用函数或表达式运算。**

  对索引使用函数或表达式运算会停止使用索引转而执行全表扫描。

- **尽量避免在where子句中使用!=或<>操作符、null值判断、or连接、参数隐式转换。**

- **对复杂SQL语句进行拆分。**

  对于过于复杂并且不易通过以上方法调整性能的SQL可以考虑拆分的方法，把SQL中某一部分拆分成独立的SQL并把执行结果存入临时表，拆分常见的场景包括但不限于：

  - 作业中多个SQL有同样的子查询，并且子查询数据量较大。
  - Plan cost计算不准，导致子查询hash bucket太小，比如实际数据1000W行，hash bucket只有1000。
  - 函数（如substr,to_number）导致大数据量子查询选择度计算不准。
  - 多DN环境下对大表做broadcast的子查询。

其他更多调优点，请参考**典型SQL调优点**。

5.3 分析正在执行的 SQL

在开发过程中，开发者常遇到SQL连接数超限、SQL查询时间过长、SQL查询阻塞等问题，您可以通过**PG_STAT_ACTIVITY**和**PGXC_THREAD_WAIT_STATUS**视图来分析和定位SQL问题，以下通过PG_STAT_ACTIVITY视图展示常用的一些定位思路。

表 5-1 部分 PG_STAT_ACTIVITY 字段

名称	类型	描述
usename	name	登录该后端的用户名。
client_addr	inet	连接到该后端的客户端的IP地址。 如果此字段是null，则表示通过服务器机器上UNIX套接字连接客户端或者这是内部进程，如autovacuum。
application_name	text	连接到该后端的应用名。
state	text	后端当前总体状态。取值如下： ● active：后台正在执行查询。 ● idle：后台正在等待新的客户端命令。 ● idle in transaction：后端在事务中，但事务中没有语句在执行。 ● idle in transaction (aborted)：后端在事务中，但事务中有语句执行失败。 ● fastpath function call：后端正在执行一个fast-path函数。 ● disabled：如果后端禁用track_activities，则报告此状态。 **说明** 普通用户只能查看到自己账户所对应的会话状态，即其他账户的state信息为空。
waiting	boolean	如果后端当前正等待锁则为t，否则为f。 ● t代表true。 ● f代表false。

名称	类型	描述
enqueue	text	语句当前排队状态。可能值是： ● waiting in global queue：表示语句在全局并发队列排队中，主要包含并发数超过单CN配置的max_active_statements。 ● waiting in respool queue：表示语句在资源池排队中，简单作业并发受限，主要是简单作业并发超过快车道并发上限max_dop。 ● waiting in ccn queue：表示作业在CCN排队中，包含全局内存排队和慢车道内存和并发排队，包含以下场景： 1. 全局可用内存超过上限，进行全局内存队列排队。 2. 资源池慢车道并发上限，即资源池并发超过active_statements上限。 3. 资源池慢车道内存上限，即资源池并发作业估算内存超过mem_percent计算的上限。 ● 空或no waiting queue：表示语句正在运行。
pid	bigint	后端线程ID。

查看连接信息

- 设置参数track_activities为on：
  ```
  SET track_activities = on;
  ```
  当此参数为on时，数据库系统才会收集当前活动查询的运行信息。
- 通过以下SQL就能确认当前的连接用户、连接地址、连接应用、状态、是否等待锁、排队状态以及线程id。
  ```
  SELECT usename,client_addr,application_name,state,waiting,enqueue,pid FROM PG_STAT_ACTIVITY WHERE DATNAME='数据库名称';
  ```
  回显如下：
  ```
   usename |  client_addr  | application_name | state  | waiting | enqueue |       pid
  ---------+---------------+------------------+--------+---------+---------+-----------------
   leo     | 192.168.0.133 | gsql             | idle   | f       |         | 139666091022080
   dbadmin | 192.168.0.133 | gsql             | active | f       |         | 139666212681472
   joe     | 192.168.0.133 |                  | idle   | f       |         | 139665671489280
  (3 rows)
  ```
- 中止某个会话连接（仅系统管理员有权限）：
  ```
  SELECT PG_TERMINATE_BACKEND(pid);
  ```

查看 SQL 运行信息

- 获取当前用户有权限查看的所有的SQL信息（若有管理员权限或预置角色权限可以显示和所有用户查询相关的信息）：
  ```
  SELECT usename,state,query FROM PG_STAT_ACTIVITY WHERE DATNAME='数据库名称';
  ```
  如果state为active，则query列表示当前执行的SQL语句，其他情况则表示为上一个查询语句；如果state字段显示为idle，则表明此连接处于空闲，等待用户输入命令。回显如下：

```
 usename | state  |                                query
---------+--------+-----------------------------------------------------------------------
 leo     | idle   | select * from joe.mytable;
 dbadmin | active | SELECT usename,state,query FROM PG_STAT_ACTIVITY WHERE
DATNAME='gaussdb';
 joe     | idle   | GRANT SELECT ON TABLE mytable to leo;
(3 rows)
```

- 查看当前正在运行（非idle）的SQL信息：

```
SELECT datname,usename,query FROM PG_STAT_ACTIVITY WHERE state != 'idle' ;
```

查看耗时较长的语句

- 查看当前运行中的耗时较长的SQL语句：

```
SELECT current_timestamp - query_start as runtime, datname, usename, query FROM
PG_STAT_ACTIVITY WHERE state != 'idle' order by 1 desc;
```

查询会返回按执行时间长短从大到小排列的查询语句列表。第一条结果就是当前系统中执行时间最长的查询语句。

```
    runtime     | datname  | usename |
query
-----------------+----------+---------
+-----------------------------------------------------------------------------------------------------------------------------------------------------
---------------------
 00:04:47.054958 | gaussdb  | leo     | insert into mytable1 select generate_series(1, 10000000);
 00:00:01.72789  | gaussdb  | dbadmin | SELECT current_timestamp - query_start as runtime, datname,
usename, query FROM PG_STAT_ACTIVITY WHERE state != 'idle' order by 1 desc;
(2 rows)
```

- 若当前系统较为繁忙，可以通过限制current_timestamp - query_start大于某一阈值来查看执行时间超过此阈值的查询语句。

```
SELECT query from PG_STAT_ACTIVITY WHERE current_timestamp - query_start > interval '2 days';
```

查看处于阻塞状态的语句

- 查看当前处于阻塞状态的查询语句：

```
SELECT pid, datname, usename, state, query FROM PG_STAT_ACTIVITY WHERE state <> 'idle' and
waiting=true;
```

执行以下语句结束到阻塞的SQL会话：

```
SELECT PG_TERMINATE_BACKEND(pid);
```

说明

- 大部分场景下，阻塞是因为系统内部锁而导致的，waiting字段才显示为true，此阻塞可在视图pg_stat_activity中体现。
- 在一些少数场景下，例如写文件、定时器等情况的查询阻塞，不会在视图pg_stat_activity中体现。

- 查看阻塞的查询语句及阻塞查询的表、模式信息

```
SELECT w.query as waiting_query,
w.pid as w_pid,
w.usename as w_user,
l.query as locking_query,
l.pid as l_pid,
l.usename as l_user,
t.schemaname || '.' || t.relname as tablename
from pg_stat_activity w join pg_locks l1 on w.pid = l1.pid
and not l1.granted join pg_locks l2 on l1.relation = l2.relation
and l2.granted join pg_stat_activity l on l2.pid = l.pid join pg_stat_user_tables t on l1.relation = t.relid
where w.waiting;
```

该查询返回会话ID、用户信息、查询状态，以及导致阻塞的表、模式信息。

查询到阻塞的表及模式信息后请根据会话ID结束会话。

```
SELECT PG_TERMINATE_BACKEND(pid);
```

显示类似如下信息，表示结束会话成功。

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

显示类似如下信息，表示用户正在尝试结束当前会话，此时仅会重连会话，而不是结束会话。

```
FATAL:  terminating connection due to administrator command
FATAL:  terminating connection due to administrator command
The connection to the server was lost. Attempting reset: Succeeded.
```

说明

gsql客户端使用PG_TERMINATE_BACKEND函数终止本会话后台线程时，客户端不会退出而是自动重连。

5.4 数据倾斜查询优秀实践

5.4.1 导入过程存储倾斜即时检测

导入过程中对DN导入行数进行统计，导入完成后计算倾斜率，超过一定阈值时，立即进行告警。倾斜率通过（DN导入行数最大值-DN导入行数最小值）/导入总行数计算。目前，只支持INSERT和COPY导入。

说明

必须设置**enable_stream_operator**=on，确保计划下发到DN，DN一次性返回导入行数，从而可以在CN计算倾斜率。

使用方法

1. 设置参数（表倾斜告警阈值**table_skewness_warning_threshold**和表倾斜告警最小行数**table_skewness_warning_rows**）。
   - 表倾斜告警阈值取值范围0~1，默认值为1，即关闭状态，取其他值时为开启状态。
   - 表倾斜告警最小行数取值范围0~2147483647，默认值为100,000。当导入总行数超过该值与导入DN数之积时，才可能触发告警，从而不会在小数据量导入的场景进行无意义的告警。

   ```
   show table_skewness_warning_threshold;
   set table_skewness_warning_threshold = xxx;
   show table_skewness_warning_rows;
   set table_skewness_warning_rows = xxx;
   ```

2. 执行导入，使用INSERT或者COPY；
3. 发现并处理告警，告警信息包括表名、最小行数、最大行数、总行数、平均行数、倾斜率，以及提示信息（检查数据分布或者修改参数）。

   ```
   WARNING:  Skewness occurs, table name: xxx, min value: xxx, max value: xxx, sum value: xxx, avg value: xxx, skew ratio: xxx
   HINT:  Please check data distribution or modify warning threshold
   ```

5.4.2 快速定位查询存储倾斜的表

目前提供的倾斜查询接口有函数：**table_distribution(schemaname text, tablename text)** 、**table_distribution()** 以及视图**PGXC_GET_TABLE_SKEWNESS**，客户可以根据自身业务情况来选择使用。

场景一：磁盘满后快速定位存储倾斜的表

首先，通过**pg_stat_get_last_data_changed_time(oid)**函数查询出近期发生过数据变更的表，介于表的最后修改时间只在进行IUD操作的CN记录，要查询库内1天(间隔可在函数中调整)内被修改的所有表，可以使用如下封装函数：

```
CREATE OR REPLACE FUNCTION get_last_changed_table(OUT schemaname text, OUT relname text)
RETURNS setof record
AS $$
DECLARE
row_data record;
row_name record;
query_str text;
query_str_nodes text;
BEGIN
query_str_nodes := 'SELECT node_name FROM pgxc_node where node_type = ''C''';
FOR row_name IN EXECUTE(query_str_nodes) LOOP
query_str := 'EXECUTE DIRECT ON (' || row_name.node_name || ') ''SELECT b.nspname,a.relname FROM
pg_class a INNER JOIN pg_namespace b on a.relnamespace = b.oid where
pg_stat_get_last_data_changed_time(a.oid) BETWEEN current_timestamp - 1 AND current_timestamp;''';
FOR row_data IN EXECUTE(query_str) LOOP
schemaname = row_data.nspname;
relname = row_data.relname;
return next;
END LOOP;
END LOOP;
return;
END; $$
LANGUAGE plpgsql;
```

然后，通过**table_distribution(schemaname text, tablename text)**查询出表在各个DN占用的存储空间。

```
SELECT table_distribution(schemaname,relname) FROM get_last_changed_table();
```

场景二：常规数据倾斜巡检

- 在库中表个数少于1W的场景，直接使用倾斜视图查询当前库内所有表的数据倾斜情况。
  ```
  SELECT * FROM pgxc_get_table_skewness ORDER BY totalsize DESC;
  ```
- 在库中表个数非常多（至少大于1W）的场景，因**PGXC_GET_TABLE_SKEWNESS**涉及全库查并计算非常全面的倾斜字段，所以可能会花费比较长的时间（小时级），建议参考**PGXC_GET_TABLE_SKEWNESS**视图定义，直接使用table_distribution()函数自定义输出，减少输出列进行计算优化，例如：
  ```
  SELECT schemaname,tablename,max(dnsize) AS maxsize, min(dnsize) AS minsize
  FROM pg_catalog.pg_class c
  INNER JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
  INNER JOIN pg_catalog.table_distribution() s ON s.schemaname = n.nspname AND s.tablename =
  c.relname
  INNER JOIN pg_catalog.pgxc_class x ON c.oid = x.pcrelid AND x.pclocatortype = 'H'
  GROUP BY schemaname,tablename;
  ```

场景三：查询某个表的数据倾斜情况

执行以下SQL查询某个表的数据倾斜情况，其中table_name替换为实际的表名。

```
SELECT a.count,b.node_name FROM (SELECT count(*) AS count,xc_node_id FROM table_name GROUP BY
xc_node_id) a, pgxc_node b WHERE a.xc_node_id=b.node_id ORDER BY a.count desc;
```

返回如下类似信息。若各DN上数据分布差小于10%，表明数据分布均衡。若大于10%，则表示数据出现倾斜。

```
gaussdb=>SELECT a.count,b.node_name FROM (select count(*) as count,xc_node_id FROM staffs GROUP BY
xc_node_id) a, pgxc_node b WHERE a.xc_node_id=b.node_id ORDER BY a.count desc;
```

```
count | node_name
------+-----------
11010 | datanode4
10000 | datanode3
12001 | datanode2
 8995 | datanode1
10000 | datanode5
 7999 | datanode6
 9995 | datanode7
10000 | datanode8
(8 rows)
```

5.5 用户管理优秀实践

GaussDB(DWS)集群中，常用的用户分别是系统管理员和普通用户。本节简述了系统管理员和普通用户的权限，如何创建以及如何查询用户相关信息。

系统管理员

在启动GaussDB(DWS)集群时创建的用户dbadmin是系统管理员，其拥有系统的最高权限，能够执行所有的操作（表空间，表，索引，模式，函数，自定义视图的操作权限及系统表和系统视图的查看权限）。

要创建新的数据库管理员，则以管理员用户身份连接数据库，并使用带SYSADMIN选项的CREATE USER语句或 ALTER USER语句进行设置。

例如：

创建用户Jim为系统管理员。

```
CREATE USER Jim WITH SYSADMIN password '{Password}';
```

修改用户Tom为系统管理员。（ALTER USER时，要求用户已存在。）

```
ALTER USER Tom SYSADMIN;
```

普通用户

普通用户由SQL语句CREATE USER创建。不具有对表空间创建，修改，删除，分配权限，访问需要被赋权；仅对自己创建的表/模式/函数/自定义视图有所有权限，仅可以在自己的表上建索引，仅可查看部分系统表和系统视图。

数据库集群包含一个或多个已命名数据库。用户在整个集群范围内是共享的，但是其数据并不共享。

常见用户相关操作如下，此处使用的密码需要用户自定义：

1. 创建用户。
   ```
   CREATE USER Tom PASSWORD '{Password}';
   ```
2. 修改用户密码。

   将用户Tom的登录密码由password修改为newpassword。
   ```
   ALTER USER Tom IDENTIFIED BY 'newpassword' REPLACE '{Password}';
   ```
3. 给用户授权。
   - 要创建有“创建数据库”权限的用户，需要加CREATEDB关键字。
     ```
     CREATE USER Tom CREATEDB PASSWORD '{Password}';
     ```
   - 为用户追加CREATEROLE权限。
     ```
     ALTER USER Tom CREATEROLE;
     ```

4. 撤销权限。
```
REVOKE ALL PRIVILEGES FROM Tom;
```
5. 锁定或解锁用户。
   - 锁定Tom账户：
```
ALTER USER Tom ACCOUNT LOCK;
```
   - 解锁Tom用户：
```
ALTER USER Tom ACCOUNT UNLOCK;
```
6. 删除用户。
```
DROP USER Tom CASCADE;
```

用户信息查询

涉及用户、角色及权限相关的系统视图有ALL_USERS、PG_USER和PG_ROLES，系统表有PG_AUTHID和PG_AUTH_MEMBERS。

- ALL_USERS视图存储记录数据库中所有用户，但不对用户信息进行详细的描述。
- PG_USER视图存储用户信息，包含用户ID，是否可以创建数据库以及用户所在资源池等信息。
- PG_ROLES视图存储数据库角色的相关信息。
- PG_AUTHID系统表存储有关数据库认证标识符（角色）的信息，包含角色是否可以登录，创建数据库等信息。
- PG_AUTH_MEMBERS存储角色的成员关系，即某个角色组包含了哪些其他角色。

1. 通过PG_USER可以查看数据库中所有用户的列表，还可以查看用户ID（USESYSID）和用户权限。
```
SELECT * FROM pg_user;
 usename | usesysid | usecreatedb | usesuper | usecatupd | userepl |  passwd  | valbegin | valuntil |
respool    | parent | spacelimit | useconfig | nodegroup | tempspacelimit | spillspacelim
it
---------+----------+-------------+----------+-----------+---------+----------+----------+----------+-------------
+--------+------------+-----------+-----------+----------------+--------------
---
 Ruby    |       10 | t          | t       | t         | t       | ******** |          |          | default_pool |      0 |
|          |          |            |
 kim     |    21661 | f          | f       | f         | f       | ******** |          |          | default_pool |      0 |
|          |          |            |
 u3      |    22662 | f          | f       | f         | f       | ******** |          |          | default_pool |      0 |
|          |          |            |
 u1      |    22666 | f          | f       | f         | f       | ******** |          |          | default_pool |      0 |
|          |          |            |
 dbadmin |    16396 | f          | f       | f         | f       | ******** |          |          | default_pool |      0
|          |          |          |          |
 u5      |    58421 | f          | f       | f         | f       | ******** |          |          | default_pool |      0 |
|          |          |            |
(6 rows)
```
2. ALL_USERS视图存储记录数据库中所有用户，但不对用户信息进行详细的描述。
```
SELECT * FROM all_users;
 username | user_id
----------+---------
 Ruby     |      10
 manager  |   21649
 kim      |   21661
 u3       |   22662
 u1       |   22666
 u2       |   22802
 dbadmin  |   16396
 u5       |   58421
(8 rows)
```

3. 系统表PG_ROLES存储访问数据库角色的相关信息。

```
SELECT * FROM pg_roles;
 rolname | rolsuper | rolinherit | rolcreaterole | rolcreatedb | rolcatupdate | rolcanlogin | rolreplication |
rolauditadmin | rolsystemadmin | rolconnlimit | rolpassword | rolvalidbegin | rolv
aliduntil |  rolrespool  | rolparentid | roltabspace | rolconfig |  oid  | roluseft | rolkind | nodegroup |
roltempspace | rolspillspace
---------+----------+------------+---------------+-------------+--------------+-------------+-------------+----------------
+---------------+----------------+-------------+-------------+-----------+--------------+-----
----------+--------------+-------------+-------------+-----------+-------+----------+---------+-----------
+--------------+---------------
 Ruby    | t        | t          | t           | t           | t            | t           | t           | t
|          -1 | ********    |             |
        | default_pool |          0 |           |         |    10 | t       | n      |         |           |
 manager | f        | t          | f           | f           | f            | f           | f           | f           | f
|          -1 | ********    |             |
        | default_pool |          0 |           |         | 21649 | f       | n      |         |           |
 kim     | f        | t          | f           | f           | f            | t           | f           | f           | f
|          -1 | ********    |             |
        | default_pool |          0 |           |         | 21661 | f       | n      |         |           |
 u3      | f        | t          | f           | f           | f            | t           | f           | f           | f
|          -1 | ********    |             |
        | default_pool |          0 |           |         | 22662 | f       | n      |         |           |
 u1      | f        | t          | f           | f           | f            | t           | f           | f           | f
|          -1 | ********    |             |
        | default_pool |          0 |           |         | 22666 | f       | n      |         |           |
 u2      | f        | t          | f           | f           | f            | f           | f           | f           | f
|          -1 | ********    |             |
        | default_pool |          0 |           |         | 22802 | f       | n      |         |           |
 dbadmin | f        | t          | f           | f           | f            | t           | f           | f           | t
|          -1 | ********    |             |
        | default_pool |          0 |           |         | 16396 | f       | n      |         |           |
 u5      | f        | t          | f           | f           | f            | t           | f           | f           | f
|          -1 | ********    |             |
        | default_pool |          0 |           |         | 58421 | f       | n      |         |           |
(8 rows)
```

4. 要查看用户属性，可查询系统表PG_AUTHID，它存储有关数据库认证标识符（角色）的信息。一个集群中只有一份pg_authid，并非每个数据库一份。需要有系统管理员权限才可以访问此系统表。

```
SELECT * FROM pg_authid;
rolname  | rolsuper | rolinherit | rolcreaterole | rolcreatedb | rolcatupdate | rolcanlogin | rolreplication |
rolauditadmin | rolsystemadmin | rolconnlimit
|

rolpassword
                                      | rolvalidbegin | rolvaliduntil |  rolrespool  | roluseft | rolparentid |
roltabspace | rolkind | rolnodegroup | roltempspace | rolspillspace | rolexcpdata | rolauthinfo
----------+----------+------------+---------------+-------------+--------------+-------------+-------------+----------------
+---------------+----------------+--------------
+-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------
------------------+---------------+---------------+--------------+----------+-------------+-------------+---------
+--------------+--------------+---------------+-------------+-------------
 Ruby    | t        | t          | t           | t           | t            | t           | t           | t           | t
|          -1 |
sha256366f1e665be208e6015bc3c5795d13e4dc297a148dca6c60346018c80e5c04c9ba170384ce44609b
31baa741f09a3ea5bedc7dadb906286ca994067c3fbf672dc08c981929e326ca08c005d8df942994e146ed
3302af47000b36e9852b50e39dmd585de11aafebd90ec620b201fc36f07a5ecdficefade3a1456ec0aca9a0
ee01e3bf2971d1dbafd604e596149e2e2928be4060dec2bd8688776588b4cd8c64fd38f1b0beab1603129f
a396556ba8aa4c7d6e137a04623 |              |               | default_pool | t        |           0 |           |
n       |           0 |            |             |             |
 sysadmin | f        | t          | f           | f           | f            | t           | f           | f           | t
|          -1 |
sha256ecaa7f0ca4436143af43074f16cdd825783ad1a5d659fd94f5e2fa5124e7da44045ecf40bda1a9797
5fcf5920dca0c8be375be5c71b51cb1eeeba0851fb3648cfa49f55989f83fd9baf1a9d5853ce19125f4fc29a7
c709c095ed02d00638410dmd556d6e2dcc41594dc7ad8ee909ef81637ecdficefadefd7d9704ee06affef958
1cd6a50a546607f88891198e96a5e84e7e83dccf56c5cd20a500bbc5248e8ea51f0bca70c5a8dcf00953f8b
62c7a181368153abce760 |              |               | default_pool | f        |           0 |           | n
```

```
|            |            |            |            |
 Tom      | f       | t         | f           | t          | f           | t          | f            | f            | f
|          -1 |
sha256f43c4f52ac51e297bc4dbdbc751fcf05319c15681dbf5a9c5777d2edce45cb592a948b25457a728e9
9a3e0608592f33b0a4312eba6124936522304ba298caa2002a04578860fecb0286d7c7baec09365eafd049
b2b99f74f21a08864dd7d3f2amd515ee49f0b18ef8e7d0cd27d91ce2fa9decdficefade16bab5f05b6d7c86a
19ae6406cc59c437506c3f6187bfdf3eefc7a7c7033afa076361b255cc8b6ccb6e19d4767effaec654b3308cc
72cebb891d00a4a10362da |              |              | default_pool | f         |           0 |             | n
|            |            |            |            |
(3 rows)
```

用户资源查询

1. 查询所有用户的资源限额和资源使用情况。

   ```
   SELECT * FROM PG_TOTAL_USER_RESOURCE_INFO;
   ```

   例如，当前所有用户的资源使用情况如下：

   ```
    username | used_memory | total_memory | used_cpu | total_cpu | used_space | total_space |
   used_temp_space | total_temp_space | used_spill_space | total_spill_space | read_kbytes | write_kbytes |
   read_counts | write_counts | read_speed | write_speed
   ----------+-------------+--------------+----------+-----------+------------+-------------+-----------------
   +------------------+------------------+------------------+-------------+--------------+-------------+--------------
   +------------+-------------
   perfadm   |          0 |       17250 |        0 |         0 |          0 |          -1 |              0 |              -1
   |              0 |             -1 |          0 |           0 |          0 |           0 |          0 |           0
   usern     |          0 |       17250 |        0 |        48 |          0 |          -1 |              0 |              -1
   |              0 |             -1 |          0 |           0 |          0 |           0 |          0 |           0
   userg     |         34 |       15525 |    23.53 |        48 |          0 |          -1 |              0 |              -1 |
   814955731 |             -1 |     6111952 |     1145864 |      763994 |       143233 |      42678 |        8001
   userg1    |         34 |       13972 |    23.53 |        48 |          0 |          -1 |              0 |              -1 |
   814972419 |             -1 |     6111952 |     1145864 |      763994 |       143233 |      42710 |        8007
   (4 rows)
   ```

2. 查询具体某个用户的资源限额和资源使用情况。

   ```
   SELECT * FROM GS_WLM_USER_RESOURCE_INFO('username');
   ```

   例如，查询Tom的资源使用情况：

   ```
   SELECT * FROM GS_WLM_USER_RESOURCE_INFO('Tom');
   userid | used_memory | total_memory | used_cpu | total_cpu | used_space | total_space |
   used_temp_space | total_temp_space | used_spill_space | total_spill_space | read_kbytes | write_kbytes |
   read_counts | write_counts | read_speed | write_speed
   -------+-------------+--------------+----------+-----------+------------+-------------+-----------------
   +------------------+------------------+------------------+-------------+--------------+-------------+--------------
   +------------+-------------
    16523 |          18 |        2831 |        0 |        19 |          0 |          -1 |              0 |              -1
   |              0 |             -1 |          0 |           0 |          0 |           0 |          0 |           0
   (1 row)
   ```

3. 查询具体某个用户的IO资源使用情况。

   ```
   SELECT * FROM pg_user_iostat('username');
   ```

   例如，查询Tom的IO资源使用情况：

   ```
   SELECT * FROM pg_user_iostat('Tom');
   userid | min_curr_iops | max_curr_iops | min_peak_iops | max_peak_iops | io_limits | io_priority
   -------+---------------+---------------+---------------+---------------+-----------+-------------
    16523 |             0 |             0 |             0 |             0 |         0 | None
   (1 row)
   ```

5.6 查看表和数据库的信息

查询表信息

- 使用系统表pg_tables查询数据库所有表的信息。

  ```
  SELECT * FROM pg_tables;
  ```

- 使用gsql的\d+命令查询表结构。

  示例：先创建表customer_t1并插入数据。


```
CREATE TABLE customer_t1
(
    c_customer_sk             integer,
    c_customer_id             char(5),
    c_first_name              char(6),
    c_last_name               char(8)
)
with (orientation = column,compression=middle)
distribute by hash (c_last_name);
INSERT INTO customer_t1 (c_customer_sk, c_customer_id, c_first_name) VALUES
    (6885, 'map', 'Peter'),
    (4321, 'river', 'Lily'),
    (9527, 'world', 'James');
```

查询表结构。（若建表时不指定schema，则表的默认schema为public）

```
\d+ customer_t1;
                        Table "public.customer_t1"
   Column     |     Type     | Modifiers | Storage  | Stats target | Description
--------------+--------------+-----------+----------+--------------+-------------
 c_customer_sk | integer      |           | plain    |              |
 c_customer_id | character(5) |           | extended |              |
 c_first_name  | character(6) |           | extended |              |
 c_last_name   | character(8) |           | extended |              |
Has OIDs: no
Distribute By: HASH(c_last_name)
Location Nodes: ALL DATANODES
Options: orientation=column, compression=middle, colversion=2.0, enable_delta=false
```

说明

此处的Options在不同版本会有差异，对实际业务没有影响，仅作参考，实际以用户当前版本查询为准。

- 使用函数pg_get_tabledef查询表定义。

```
SELECT * FROM PG_GET_TABLEDEF('customer_t1');
                                pg_get_tabledef
--------------------------------------------------------------------------------
 SET search_path = tpchobs;                                                    +
 CREATE  TABLE customer_t1 (                                                   +
        c_customer_sk integer,                                                 +
        c_customer_id character(5),                                            +
        c_first_name character(6),                                             +
        c_last_name character(8)                                               +
 )                                                                             +
 WITH (orientation=column, compression=middle, colversion=2.0, enable_delta=false)+
 DISTRIBUTE BY HASH(c_last_name)                                               +
 TO GROUP group_version1;
(1 row)
```

- 执行如下命令查询表customer_t1的所有数据。

```
SELECT * FROM customer_t1;
 c_customer_sk | c_customer_id | c_first_name | c_last_name
---------------+---------------+--------------+-------------
          6885 | map           | Peter        |
          4321 | river         | Lily         |
          9527 | world         | James        |
(3 rows)
```

- 使用SELECT查询表customer_t1中某一字段的所有数据。

```
SELECT c_customer_sk FROM customer_t1;
 c_customer_sk
---------------
          6885
          4321
          9527
(3 rows)
```

- 查询表是否做过表分析，执行如下命令会返回每个表最近一次做analyze的时间，没有返回时间的则表示没有做过analyze。

```
SELECT pg_stat_get_last_analyze_time(oid),relname FROM pg_class where relkind='r';
```

查询public下的表做表分析的时间：

```
SELECT pg_stat_get_last_analyze_time(c.oid),c.relname FROM pg_class c LEFT JOIN pg_namespace n
ON c.relnamespace = n.oid WHERE c.relkind='r' AND n.nspname='public';
 pg_stat_get_last_analyze_time |       relname
-------------------------------+----------------------
 2022-05-17 07:48:26.923782+00 | warehouse_t19
 2022-05-17 07:48:26.964512+00 | emp
 2022-05-17 07:48:27.016709+00 | test_trigger_src_tbl
 2022-05-17 07:48:27.045385+00 | customer
 2022-05-17 07:48:27.062486+00 | warehouse_t1
 2022-05-17 07:48:27.114884+00 | customer_t1
 2022-05-17 07:48:27.172256+00 | product_info_input
 2022-05-17 07:48:27.197014+00 | tt1
 2022-05-17 07:48:27.212906+00 | timezone_test
(9 rows)
```

- 快速查到一张表的列信息，information_schema下的视图在数据库中对象较多时返回结果很慢，可以通过以下sql快速查询到一张或几张表的列信息：

```
SELECT /*+ nestloop(a c)*/ c.column_name, c.data_type, c.ordinal_position, pgd.description, pp.partkey,
c.is_nullable, c.column_default, c.character_maximum_length, c.numeric_precision,  c.numeric_scale,
c.datetime_precision, c.interval_type, c.udt_name from information_schema.columns as c left join
pg_namespace sp on sp.nspname = c.table_schema left join pg_class cla on cla.relname =
c.table_name and cla.relnamespace = sp.oid left join pg_catalog.pg_partition pp on (pp.parentid =
cla.oid and pp.parttype = 'r') left join pg_catalog.pg_description pgd on (pgd.objoid=cla.oid and
pgd.objsubid = c.ordinal_position)where c.table_name in ('tablename') and c.table_schema = 'public';
```

例如，快速查询表customer_t1的列信息：

```
SELECT /*+ nestloop(a c)*/ c.column_name, c.data_type, c.ordinal_position, pgd.description, pp.partkey,
c.is_nullable, c.column_default, c.character_maximum_length, c.numeric_precision, c.numeric_scale,
c.datetime_precision, c.interval_type, c.udt_name from information_schema.columns as c left join
pg_namespace sp on sp.nspname = c.table_schema left join pg_class cla on cla.relname =
c.table_name and cla.relnamespace = sp.oid left join pg_catalog.pg_partition pp on (pp.parentid =
cla.oid and pp.parttype = 'r') left join pg_catalog.pg_description pgd on (pgd.objoid=cla.oid and
pgd.objsubid = c.ordinal_position) where c.table_name in ('customer_t1') and c.table_schema = 'public';
  column_name  | data_type | ordinal_position | description | partkey | is_nullable | column_default |
character_maximum_length | numeric_precision | numeric_scale | datetime_precision | interval_type |
udt_name
---------------+-----------+------------------+-------------+---------+-------------+----------------
+--------------------------+-------------------+-------------
--+--------------------+---------------+----------
 c_last_name   | character |                4 |             |         | YES         |              |
|                  |
  |                  |               | bpchar
 c_first_name  | character |                3 |             |         | YES         |              |
|                  |
  |                  |               | bpchar
 c_customer_id | character |                2 |             |         | YES         |              |
|                  |
  |                  |               | bpchar
 c_customer_sk | integer   |                1 |             |         | YES         |              |
|               32 |
0 |                  |               | int4
(4 rows)
```

- 通过查询审计日志获取表定义。

  使用函数pgxc_query_audit可以查询所有CN节点的审计日志，其语法为：

  ```
  pgxc_query_audit(timestamptz startime,timestamptz endtime)
  ```

  查询审计多个对象名的记录：

  ```
  SET audit_object_name_format TO 'all';
  SELECT object_name,result,operation_type,command_text FROM pgxc_query_audit('2022-08-26
  8:00:00','2022-08-26 22:55:00') where command_text like '%student%';
  ```

查询表大小

- 查询表的总大小（包含表的索引和数据）。

  ```
  SELECT pg_size_pretty(pg_total_relation_size('<schemaname>.<tablename>'));
  ```

示例：

先在customer_t1创建索引：

```
CREATE INDEX index1 ON customer_t1 USING btree(c_customer_sk);
```

然后查询public模式下，customer_t1表的大小。

```
SELECT pg_size_pretty(pg_total_relation_size('public.customer_t1'));
 pg_size_pretty
----------------
 264 kB
(1 row)
```

- 查询表的数据大小（不包括索引）。

```
SELECT pg_size_pretty(pg_relation_size('<schemaname>.<tablename>'));
```

示例：查询public模式下，customer_t1表的大小。

```
SELECT pg_size_pretty(pg_relation_size('public.customer_t1'));
 pg_size_pretty
----------------
 208 kB
(1 row)
```

- 查询系统中所有表占用空间大小排行。

```
SELECT table_schema || '.' || table_name AS table_full_name, pg_size_pretty(pg_total_relation_size('"' ||
table_schema || '"."' || table_name || '"')) AS size FROM information_schema.tables
ORDER BY
pg_total_relation_size('"' || table_schema || '"."' || table_name || '"') DESC limit xx;
```

示例1：查询系统中所有表占用空间大小排行前15。

```
SELECT table_schema || '.' || table_name AS table_full_name, pg_size_pretty(pg_total_relation_size('"' ||
table_schema || '"."' || table_name || '"')) AS size FROM information_schema.tables
ORDER BY
pg_total_relation_size('"' || table_schema || '"."' || table_name || '"') DESC limit 15;
      table_full_name       |  size
----------------------------+---------
 pg_catalog.pg_attribute    | 2048 KB
 pg_catalog.pg_rewrite      | 1888 KB
 pg_catalog.pg_depend       | 1464 KB
 pg_catalog.pg_proc         | 1464 KB
 pg_catalog.pg_class        | 512 KB
 pg_catalog.pg_description | 504 KB
 pg_catalog.pg_collation    | 360 KB
 pg_catalog.pg_statistic    | 352 KB
 pg_catalog.pg_type         | 344 KB
 pg_catalog.pg_operator     | 224 KB
 pg_catalog.pg_amop         | 208 KB
 public.tt1                 | 160 KB
 pg_catalog.pg_amproc       | 120 KB
 pg_catalog.pg_index        | 120 KB
 pg_catalog.pg_constraint   | 112 KB
(15 rows)
```

示例2：查询public模式下所有表占用空间排行前20。

```
SELECT table_schema || '.' || table_name AS table_full_name, pg_size_pretty(pg_total_relation_size('"' ||
table_schema || '"."' || table_name || '"')) AS size FROM information_schema.tables where
table_schema='public'
ORDER BY
pg_total_relation_size('"' || table_schema || '"."' || table_name || '"') DESC limit 20;
       table_full_name        |  size
------------------------------+---------
 public.tt1                   | 160 KB
 public.product_info_input    | 112 KB
 public.customer_t1           | 96 KB
 public.warehouse_t19         | 48 KB
 public.emp                   | 32 KB
 public.customer              | 0 bytes
 public.test_trigger_src_tbl | 0 bytes
 public.warehouse_t1          | 0 bytes
(8 rows)
```

快速查询全库中所有表占用空间大小

8.1.3及以上集群版本在大集群大数据量（表数量大于1000）场景下，如果进行全库表查询，建议优先使用pgxc_wlm_table_distribution_skewness视图，该视图可以查到全库内的各表空间使用情况以及数据倾斜分布情况。其中，total_size和avg_size的单位为字节（bytes）。

```
SELECT *, pg_size_pretty(total_size) as tableSize FROM pgxc_wlm_table_distribution_skewness ORDER BY
total_size desc;
   schema_name    |                table_name                | total_size | avg_size  | max_percent |
min_percent | skew_percent | tablesize
------------------+------------------------------------------+------------+-----------+-------------
+-------------+--------------+-----------
 public           | history_tbs_test_row_1                   |  804347904 | 134057984 |       18.02 |       15.63
|        7.53 | 767 MB
 public           | history_tbs_test_row_3                   |  402096128 |  67016021 |       18.30 |       15.60
|        8.90 | 383 MB
 public           | history_tbs_test_row_2                   |  401743872 |  66957312 |       18.01 |       15.01
|        7.47 | 383 MB
 public           | i_history_tbs_test_1                     |  325263360 |  54210560 |       17.90 |       15.50
|        6.90 | 310 MB
```

查询结果显示history_tbs_test_row_1表占用空间最大，且数据有一定的倾斜。

⚠ 注意

1. 视图pgxc_wlm_table_distribution_skewness需要打开GUC参数use_workload_manager和enable_perm_space才能进行查询，在低版本查询全库时建议使用table_distribution()函数，如果仅查询某一张表的大小，推荐使用table_distribution(schemaname text, tablename text)函数。
2. 8.2.1及以上集群版本中，GaussDB(DWS)已支持pgxc_wlm_table_distribution_skewness视图，可直接查询。
3. 在8.1.3集群版本中，可使用如下定义创建视图后再进行查询：

```
CREATE OR REPLACE VIEW
pgxc_wlm_table_distribution_skewness AS
WITH skew AS
(
SELECT
schemaname,
tablename,
pg_catalog.sum(dnsize)
AS totalsize,
pg_catalog.avg(dnsize)
AS avgsize,
pg_catalog.max(dnsize)
AS maxsize,
pg_catalog.min(dnsize)
AS minsize,
(maxsize
- avgsize) * 100 AS skewsize
FROM
pg_catalog.gs_table_distribution()
GROUP
BY schemaname, tablename
)
SELECT
    schemaname AS schema_name,
    tablename AS table_name,
    totalsize AS total_size,
    avgsize::numeric(1000) AS avg_size,
    (
        CASE
            WHEN totalsize = 0 THEN 0.00
            ELSE (maxsize * 100 /
totalsize)::numeric(5, 2)
        END
    ) AS max_percent,
    (
        CASE
            WHEN totalsize = 0 THEN 0.00
            ELSE (minsize * 100 /
totalsize)::numeric(5, 2)
        END
    ) AS min_percent,
    (
        CASE
            WHEN totalsize = 0 THEN 0.00
            ELSE (skewsize /
maxsize)::numeric(5, 2)
        END
    ) AS skew_percent
FROM skew;
```

查询数据库

- 使用gsql的\l元命令查看数据库系统的数据库列表。

```
\l
                    List of databases
```

```
   Name    | Owner | Encoding  | Collate | Ctype | Access privileges
-----------+-------+-----------+---------+-------+-------------------
 gaussdb   | Ruby  | SQL_ASCII | C       | C     |
 template0 | Ruby  | SQL_ASCII | C       | C     | =c/Ruby          +
           |       |           |         |       | Ruby=CTc/Ruby
 template1 | Ruby  | SQL_ASCII | C       | C     | =c/Ruby          +
           |       |           |         |       | Ruby=CTc/Ruby
(3 rows)
```

说明

- 如果用户在数据库安装的时候没有指定LC_COLLATE、LC_CTYPE参数，则LC_COLLATE、LC_CTYPE参数的默认值为C。
- 如果用户在创建数据库时没有指定LC_COLLATE、LC_CTYPE参数，则默认使用模板数据库的排序顺序及字符分类。

  详细内容可参见**CREATE DATABASE参数说明**。

- 通过系统表pg_database查询数据库列表。

```
SELECT datname FROM pg_database;
  datname
-----------
 template1
 template0
 gaussdb
(3 rows)
```

查询数据库大小

查询数据库的大小。

```
select datname,pg_size_pretty(pg_database_size(datname)) from pg_database;
```

示例：

```
select datname,pg_size_pretty(pg_database_size(datname)) from pg_database;
  datname  | pg_size_pretty
-----------+----------------
 template1 | 61 MB
 template0 | 61 MB
 postgres  | 320 MB
(3 rows)
```

查询指定 SCHEMA 下的表大小及表对应索引的大小

```
SELECT
    t.tablename,
    indexname,
    c.reltuples AS num_rows,
    pg_size_pretty(pg_relation_size(quote_ident(t.tablename)::text)) AS table_size,
    pg_size_pretty(pg_relation_size(quote_ident(indexrelname)::text)) AS index_size,
    CASE WHEN indisunique THEN 'Y'
       ELSE 'N'
    END AS UNIQUE,
    idx_scan AS number_of_scans,
    idx_tup_read AS tuples_read,
    idx_tup_fetch AS tuples_fetched
FROM pg_tables t
LEFT OUTER JOIN pg_class c ON t.tablename=c.relname
LEFT OUTER JOIN
    ( SELECT c.relname AS ctablename, ipg.relname AS indexname, x.indnatts AS number_of_columns,
idx_scan, idx_tup_read, idx_tup_fetch, indexrelname, indisunique FROM pg_index x
           JOIN pg_class c ON c.oid = x.indrelid
           JOIN pg_class ipg ON ipg.oid = x.indexrelid
           JOIN pg_stat_all_indexes psai ON x.indexrelid = psai.indexrelid )
    AS foo
    ON t.tablename = foo.ctablename
WHERE t.schemaname='public'
ORDER BY 1,2;
```

5.7 数据库 SEQUENCE 优秀实践

sequence，也称作序列，是用来产生唯一整数的数据库对象。序列的值按照一定的规则自增/自减，一般常被用作主键。GaussDB(DWS)中，创建sequence时会同时创建一张同名的元数据表，用来记录sequence相关的信息，例如：

```
CREATE SEQUENCE seq_test;
CREATE SEQUENCE

SELECT * FROM seq_test;
 sequence_name | last_value | start_value | increment_by |      max_value      | min_value | cache_value |
log_cnt | is_cycled | is_called |  uuid
---------------+------------+-------------+--------------+---------------------+-----------+-------------+---------
+-----------+-----------+---------
 seq_test      |         -1 |           1 |            1 | 9223372036854775807 |         1 |           1 |       0 | f         |
f         | 1400050
(1 row)
```

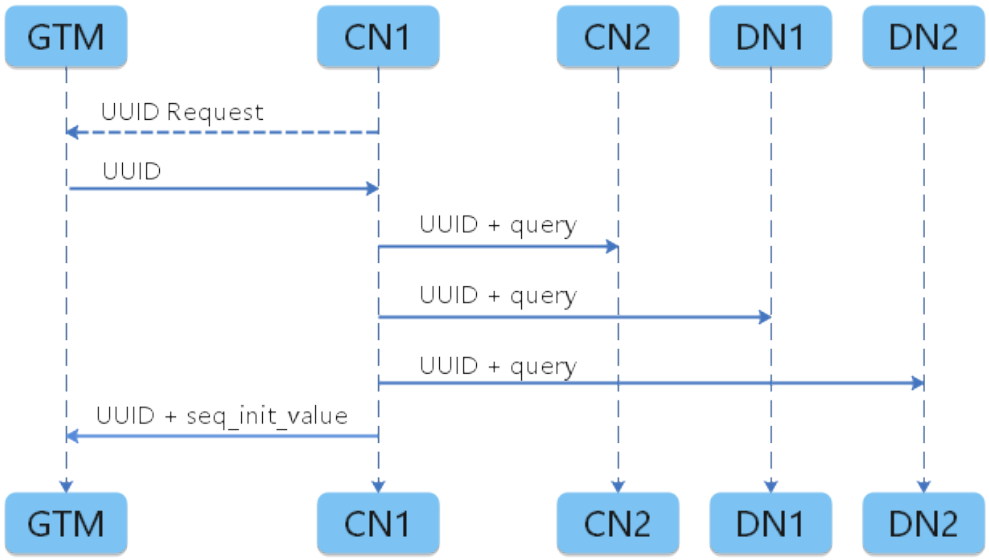
其中，

- sequence_name表示sequence的名称。
- last_value当前无意义。
- start_value表示sequence的初始值。
- increment_by表示sequence的步长。
- max_value表示sequence的最大值。
- min_value表示sequence最小值。
- cache_value表示为了快速获取下一个序列值而预先存储的sequence值个数（定义cache后不能保证sequence值的连续性，会产生空洞，造成序列号段浪费）。
- log_cnt表示WAL日志记录的sequence值个数，由于在GaussDB(DWS)中sequence是从GTM获取和管理，因此log_cnt无实际意义。
- is_cycled表示sequence在达到最小或最大值后是否循环继续。
- is_called表示该sequence是否已被调用（仅表示在当前实例是否被调用，例如在cn1上调用之后，cn1上该原数据表的值变为t，cn2上该字段仍为f）。
- uuid代表该sequence的唯一标识。

sequence 创建流程

GaussDB(DWS)中，GTM（Global Transaction Manager，即全局事务管理器）负责生成和维护全局事务ID、事务快照、sequence等需要全局唯一的信息。sequence在GaussDB(DWS)中的创建流程如下图所示：

图 5-2 sequence 创建流程



具体过程为：

1. 接受SQL命令的CN从GTM申请UUID。
2. GTM返回一个UUID。
3. CN将获取的UUID与用户创建的sequenceName绑定。
4. CN将绑定关系下发到其他节点上，其他节点同步创建sequence元数据表。
5. CN将UUID和sequence的startID发送到GTM端，在GTM行进行永久保存。

因此，sequence的维护和申请实际是在GTM上完成的。**当申请nextval，每个执行nextval调用的实例会根据该sequence的UUID到GTM上申请序列值，每次申请的序列值范围与cache有关，只有当cache消耗完之后才会继续到GTM上申请。因此，增大sequence的cache有利于减少CN/DN与GTM通信的次数。**

创建 sequence 的两种方式

方式一：使用CREATE SEQUENCE语句创建序列，在新建的表中通过nextval调用。

```
CREATE SEQUENCE seq_test increment by 1 minvalue 1 no maxvalue start with 1;
CREATE SEQUENCE

CREATE TABLE table_1(id int not null default nextval('seq_test'), name text);
CREATE TABLE
```

方式二：建表时使用serial类型，会自动创建一个sequence，并且会将该列的默认值设置为nextval。

```
CREATE TABLE mytable(a int, b serial) distribute by hash(a);
NOTICE:  CREATE TABLE will create implicit sequence "mytable_b_seq" for serial column "mytable.b"
CREATE TABLE

 \d+ mytable
                                    Table "dbadmin.mytable"
 Column |  Type   |                      Modifiers                      | Storage | Stats target | Description
--------+---------+-----------------------------------------------------+---------+--------------+-------------
 a      | integer |                                                     | plain   |              |
 b      | integer | not null default nextval('mytable_b_seq'::regclass) | plain   |              |
Has OIDs: no
Distribute By: HASH(a)
```



```
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no
```

本示例中会自动创建一个名为mytable_b_seq的sequence。严格来讲serial类型不是真正的类型，只是为在表中设置唯一标识而存在的概念，在创建时会同时创建一个sequence，并与该列相关联。

等同于下列的操作语句：

```
CREATE TABLE mytable01(a int, b int) distribute by hash(a);
CREATE TABLE

CREATE SEQUENCE mytable01_b_seq owned by mytable.b;
CREATE SEQUENCE

ALTER SEQUENCE mytable01_b_seq owner to u1;  --u1为mytable01表的属主，如果当前用户即为属主，可不执行此语句。
ALTER SEQUENCE

ALTER TABLE mytable01 alter b set default nextval('mytable01_b_seq'), alter b set not null;
ALTER TABLE

\d+ mytable01
                                    Table "dbadmin.mytable01"
 Column |  Type   |                      Modifiers                       | Storage | Stats target | Description
--------+---------+------------------------------------------------------+---------+--------------+-------------
 a      | integer |                                                      | plain   |              |
 b      | integer | not null default nextval('mytable01_b_seq'::regclass) | plain   |              |
Has OIDs: no
Distribute By: HASH(a)
Location Nodes: ALL DATANODES
Options: orientation=row, compression=no
```

sequence 在业务中的常见用法

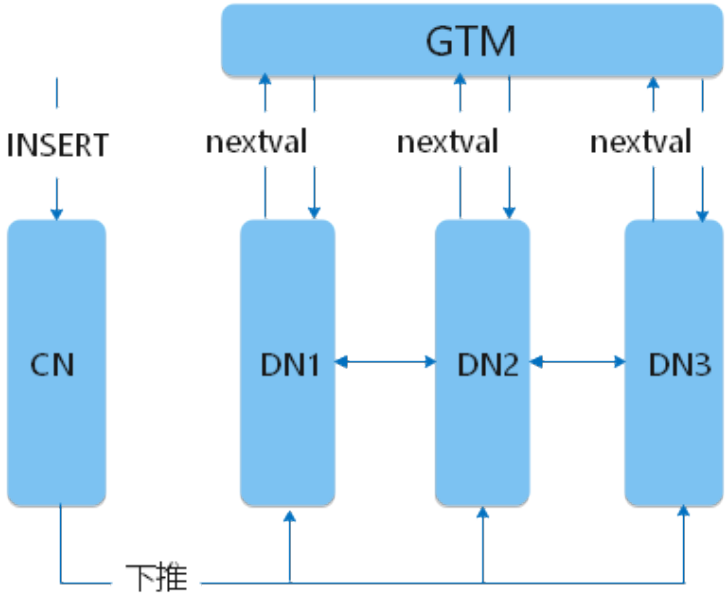
sequence在业务中常被用作在导入时生成主键或唯一列，常见于数据迁移场景。不同的迁移工具或业务导入场景使用的入库方法不同，常见的方法主要可以分为**copy**和**insert**。对于seqeunce来讲，这两种场景在处理时略有差别。

- **场景一：insert下推场景**

  ```
  CREATE TABLE test1(a int, b serial) distribute by hash(a);
  NOTICE:  CREATE TABLE will create implicit sequence "test1_b_seq" for serial column "test1.b"
  CREATE TABLE

  CREATE TABLE test2(a int) distribute by hash(a);
  CREATE TABLE

  EXPLAIN VERBOSE INSERT INTO test1(a) SELECT a FROM test2;
                                      QUERY PLAN
  ------------------------------------------------------------------------------------------------
   id |              operation              | E-rows | E-distinct | E-memory | E-width | E-costs
  ----+-------------------------------------+--------+------------+----------+---------+---------
    1 | ->  Streaming (type: GATHER)        |      1 |            |          |       4 | 16.34
    2 |    ->  Insert on dbadmin.test1      |     30 |            |          |       4 | 16.22
    3 |       ->  Seq Scan on dbadmin.test2 |     30 |            | 1MB      |       4 | 14.21

             RunTime Analyze Information
  ------------------------------------------------------
         "dbadmin.test2" runtime: 9.586ms, sync stats

       Targetlist Information (identified by plan id)
  ------------------------------------------------------------
    1 --Streaming (type: GATHER)
          Node/s: All datanodes
    3 --Seq Scan on dbadmin.test2
          Output: test2.a, nextval('test1_b_seq'::regclass)
          Distribute Key: test2.a
  ```

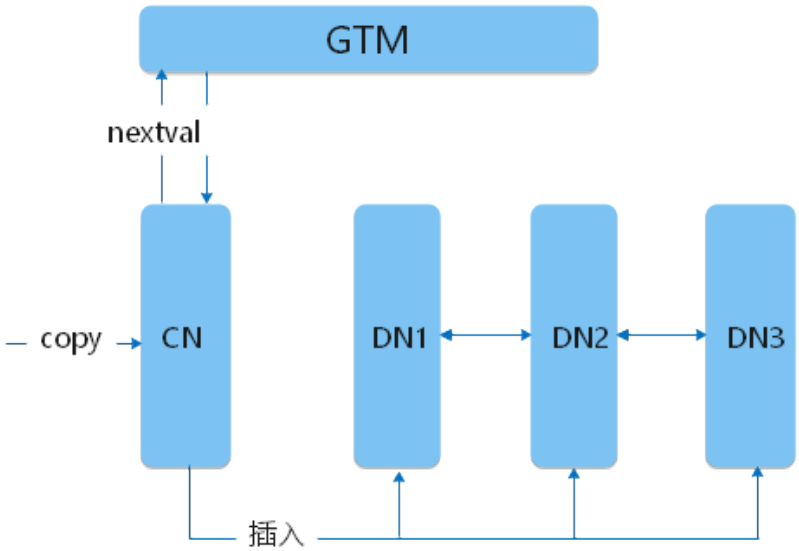
```
 ====== Query Summary =====
---------------------------------
System available mem: 1351680KB
Query Max mem: 1351680KB
Query estimated mem: 1024KB
Parser runtime: 0.076 ms
Planner runtime: 12.666 ms
Unique SQL Id: 831364267
(26 rows)
```

由于nextval在INSERT场景下可以下推到DN执行，因此，不管是使用default值的nextval，还是显示调用nextval，nextval都会被下推到DN执行，在上例的执行计划中也能看出，nextval的调用在sequence层，说明是在DN执行的。此时，DN直接向GTM申请序列值，且各DN并行执行，因此效率相对较高。



- **场景二：copy场景**

  在业务开发过程中，入库方式除了INSERT外，还有COPY入库的场景。常用于将文件内容COPY入库、使用CopyManager接口入库等。此外，CDM数据同步工具，其实现方式也是通过COPY的方式批量入库。在COPY入库过程中，如果COPY的目标表使用了默认值，且默认值为nextval，处理过程如下图：



COPY场景中，由CN负责向GTM申请序列值，因此，当sequence的cache值较小，CN会频繁和GTM建联并申请nextval，出现性能瓶颈。**sequence相关的典型优化场景**将针对此种场景说明业务上的性能表现并提供优化方法。

sequence 相关的典型优化场景

业务场景：某业务场景中使用CDM数据同步工具进行数据迁移，从源端入库目标端GaussDB(DWS)。导入速率与经验值相差较大，业务将CDM并发从1调整为5，同步速率仍无法提升。查看语句执行情况，除COPY入库外，其余业务均正常执行，无性能瓶颈，且观察无资源瓶颈，因此初步判断为该业务自身存在瓶颈，查看该表COPY相关的作业等待视图情况：

```
=# select * from pgxc_thread_wait_status where query_id in (2172986873834471 30,2172986873834757 07,2172986873839563 17,2172986873839623 87,2172986873840266 48) order by query_id;
  node_name   | db_name | thread_name |      query_id       |       tid       | lwtid | ptid | tlevel | smpid |     wait_status      |     wait_event
--------------+---------+-------------+---------------------+-----------------+-------+------+--------+-------+----------------------+---------------------
 dn_6001_6002 | kfpt    | cn_5003     | 2172986873834471 30 | 281459614074928 |  2490718 |      |      0 |     0 | wait cmd             |
 dn_6005_6006 | kfpt    | cn_5003     | 2172986873834471 30 | 281459123161136 |   211948 |      |      0 |     0 | wait cmd             |
 dn_6003_6004 | kfpt    | cn_5003     | 2172986873834471 30 | 281460010731568 |  3913471 |      |      0 |     0 | wait cmd             |
 dn_6007_6008 | kfpt    | cn_5003     | 2172986873834471 30 | 281459064444976 |  2974031 |      |      0 |     0 | wait cmd             |
 cn_5003      | kfpt    |             | 2172986873834471 30 | 281454976845872 |   211939 |      |      0 |     0 | gtm_get_sequence_val |
 cn_5003      | kfpt    |             | 2172986873834757 07 | 281454960064560 |   211941 |      |      0 |     0 | acquire_lwlock       | PgStatObjectLock
 dn_6001_6002 | kfpt    | cn_5003     | 2172986873834757 07 | 281459882559536 |  2490701 |      |      0 |     0 | wait cmd             |
 dn_6005_6006 | kfpt    | cn_5003     | 2172986873834757 07 | 281460079614000 |   211844 |      |      0 |     0 | wait cmd             |
 dn_6003_6004 | kfpt    | cn_5003     | 2172986873834757 07 | 281458634766384 |  3913447 |      |      0 |     0 | wait cmd             |
 dn_6007_6008 | kfpt    | cn_5003     | 2172986873834757 07 | 281460012533808 |  2973982 |      |      0 |     0 | wait cmd             |
 dn_6003_6004 | kfpt    | cn_5003     | 2172986873839563 17 | 281458383046704 |  3913477 |      |      0 |     0 | wait cmd             |
 cn_5003      | kfpt    |             | 2172986873839563 17 | 281454926501936 |   211943 |      |      0 |     0 | acquire_lwlock       | PgStatObjectLock
 dn_6005_6006 | kfpt    | cn_5003     | 2172986873839563 17 | 281458955372592 |   211952 |      |      0 |     0 | wait cmd             |
 dn_6001_6002 | kfpt    | cn_5003     | 2172986873839563 17 | 281459345590320 |  2490722 |      |      0 |     0 | wait cmd             |
 dn_6007_6008 | kfpt    | cn_5003     | 2172986873839563 17 | 281458762401840 |  2974037 |      |      0 |     0 | wait cmd             |
 dn_6005_6006 | kfpt    | cn_5003     | 2172986873839623 87 | 281459139942448 |   211949 |      |      0 |     0 | wait cmd             |
 cn_5003      | kfpt    |             | 2172986873839623 87 | 281454943283248 |   211942 |      |      0 |     0 | acquire_lwlock       | PgStatObjectLock
 dn_6003_6004 | kfpt    | cn_5003     | 2172986873839623 87 | 281458433390640 |  3913473 |      |      0 |     0 | wait cmd             |
 dn_6001_6002 | kfpt    | cn_5003     | 2172986873839623 87 | 281459412715568 |  2490719 |      |      0 |     0 | wait cmd             |
 dn_6007_6008 | kfpt    | cn_5003     | 2172986873839623 87 | 281458795964464 |  2974033 |      |      0 |     0 | wait cmd             |
 dn_6001_6002 | kfpt    | cn_5003     | 2172986873840266 48 | 281459395934256 |  2490721 |      |      0 |     0 | wait cmd             |
 dn_6005_6006 | kfpt    | cn_5003     | 2172986873840266 48 | 281459008598512 |   211951 |      |      0 |     0 | wait cmd             |
 dn_6007_6008 | kfpt    | cn_5003     | 2172986873840266 48 | 281458779183152 |  2974036 |      |      0 |     0 | wait cmd             |
 cn_5003      | kfpt    |             | 2172986873840266 48 | 281454993627184 |   211940 |      |      0 |     0 | acquire_lwlock       | PgStatObjectLock
 dn_6003_6004 | kfpt    | cn_5003     | 2172986873840266 48 | 281458399828016 |  3913476 |      |      0 |     0 | wait cmd             |
(25 rows)
```

如上图所示，由于CDM作业执行了5个并发，因此在活跃视图中可以看到5个COPY语句，根据这5个COPY语句对应的query_id查看等待视图情况。查看到这5个COPY中，同一时刻，仅有1个COPY在向GTM申请序列值，其余的COPY在等待轻量级锁。因此，即使作业中开启了5并发在运行，实际效果较1并发并没有带来明显提升。

问题原因：目标表在建表时使用了serial类型，默认创建的sequence的cache为1，导致在并发COPY入库时，CN频繁与GTM建连，且多个并发之间存在轻量锁争抢，导致数据同步效率低。

解决方案：此种场景下可以调大sequence的cache值，防止频繁GTM建联带来的瓶颈。本业务场景示例中，业务每次同步的数据量在10万左右，根据业务评估，将cache值修改为10000（实际使用时应根据业务设置合理的cache值，既能保证快速访问，又不会造成序列号浪费）。

8.2.1.100及以上集群版本中支持使用ALTER SEQUENCE的方式修改cache值。

8.2.1及之前低版本集群中GaussDB(DWS)不支持通过ALTER SEQUENCE的方式修改cache值，可以通过如下方式修改已有sequence的cache值，以mytable表为例：

步骤1 解除当前sequence与目标表的关联关系

```
ALTER SEQUENCE mytable_b_seq owned by none;
ALTER TABLE mytable alter b drop default;
```

步骤2 记录当前seqeunce值，作为新建sequence的start value。

```
SELECT nextval('mytable_b_seq');
```

删除sequence。

```
DROP SEQUENCE mytable_b_seq;
```

步骤3 新建seqeunce并绑定目标表，xxx替换为上一步查到的nextval值。

```
CREATE SEQUENCE mytable_b_seq START with xxx cache 10000 owned by mytable.b;
ALTER SEQUENCE mytable_b_seq owner to u1;--u1为mytable表的属主，如果当前用户即为属主，可不执行此语句。
ALTER TABLE mytable alter b set default nextval('mytable_b_seq');
```

----结束

6 模拟数据分析

6.1 交通卡口通行车辆分析

本实践将演示交通卡口车辆通行分析，将加载8.9亿条交通卡口车辆通行模拟数据到数据仓库单个数据库表中，并进行车辆精确查询和车辆模糊查询，展示GaussDB(DWS)对于历史详单数据的高性能查询能力。

说明

GaussDB(DWS) 已预先将样例数据上传到OBS桶的“traffic-data”文件夹中，并给所有华为云用户赋予了该OBS桶的只读访问权限。

操作流程

本实践预计时长40分钟，基本流程如下：

1. **准备工作**
2. **步骤一：创建集群**
3. **步骤二：使用Data Studio连接集群**
4. **步骤三：导入交通卡口样例数据**
5. **步骤四：车辆分析**

支持区域

当前已上传OBS数据的区域如**表6-1**所示。

表 6-1 区域和 OBS 桶名

区域	OBS桶名
华北-北京一	dws-demo-cn-north-1
华北-北京二	dws-demo-cn-north-2
华北-北京四	dws-demo-cn-north-4
华北-乌兰察布一	dws-demo-cn-north-9

区域	OBS桶名
华东-上海一	dws-demo-cn-east-3
华东-上海二	dws-demo-cn-east-2
华南-广州	dws-demo-cn-south-1
华南-广州友好	dws-demo-cn-south-4
中国-香港	dws-demo-ap-southeast-1
亚太-新加坡	dws-demo-ap-southeast-3
亚太-曼谷	dws-demo-ap-southeast-2
拉美-圣地亚哥	dws-demo-la-south-2
非洲-约翰内斯堡	dws-demo-af-south-1
拉美-墨西哥城一	dws-demo-na-mexico-1
拉美-墨西哥城二	dws-demo-la-north-2
莫斯科二	dws-demo-ru-northwest-2
拉美-圣保罗一	dws-demo-sa-brazil-1

准备工作

- 已注册账号，且在使用GaussDB(DWS) 前检查账号状态，账号不能处于欠费或冻结状态。
- 获取此账号的“AK/SK”。

步骤一：创建集群

步骤1 登录管理控制台。

步骤2 在“服务列表”中，选择“大数据 > 数据仓库服务 GaussDB(DWS)”。

步骤3 左侧导航栏单击“集群管理”，进入页面后，单击右上角的“创建数据仓库集群”按钮。

步骤4 参见表6-2进行基础配置。

表 6-2 基础配置

参数名称	配置方式
区域	选择“华北-北京四”。 **说明** 本指导以“华北-北京四”为例进行介绍，如果您需要选择其他区域进行操作，请确保所有操作均在同一区域进行。
可用分区	可用区2

参数名称	配置方式
产品类型	标准数仓
计算类型	弹性云服务器
存储类型	SSD云盘
CPU架构	X86
节点规格	dws2.m6.4xlarge.8（16 vCPU | 128GB | 2000GB SSD） **说明** 如规格售罄，可选择其他可用区或规格。
热数据存储	100GB / 节点
节点数量	3

步骤5 信息核对无误，单击“下一步：网络配置”，参见表6-3进行网络配置。

表 6-3 网络配置

参数名称	配置方式
虚拟私有云	vpc-default
子网	subnet-default(192.168.0.0/24)
安全组	自动创建安全组
公网访问	现在购买
宽带	1Mbit/s
弹性负载均衡	暂不使用

步骤6 信息核对无误，单击“下一步：高级配置”，参见表6-4进行网络配置。

表 6-4 高级配置

参数名称	配置方式
集群名称	dws-demo
集群版本	使用推荐版本，例如8.1.3.311
管理员用户	dbadmin
管理员密码	-
确认密码	-
数据库端口	8000
企业项目	default

参数名称	配置方式
高级配置	默认配置

步骤7 单击“下一步：确认配置”，确认无误后，单击“立即购买”

步骤8 等待约6分钟，待集群创建成功后，单击集群名称前面的，弹出集群信息，记录下“公网访问地址”。

----结束

步骤二：使用 Data Studio 连接集群

步骤1 请确保客户端主机已安装JDK 1.8.0以上版本，并进入“此电脑 > 属性 > 高级系统设置 > 环境变量”设置JAVA_HOME（例如C:\Program Files\Java\jdk1.8.0_191），并在变量path中添加“;%JAVA_HOME%\bin”。

步骤2 在GaussDB(DWS) 控制台的“连接管理”页面，下载Data Studio客户端。

步骤3 解压下载的Data Studio软件包，进入解压目录后，双击Data Studio.exe启动客户端。

步骤4 在Data Studio主菜单中选择“文件 > 新建连接”，并在弹出框中参照表6-5所示配置。

表 6-5 Data Studio 软件配置

参数名称	配置方式
数据库类型	GaussDB(DWS)
名称	dws-demo
主机	dws-demov.dws.huaweicloud.com 与步骤一：创建集群查询到的“公网访问地址”一致。
端口	8000
数据库	gaussdb
用户名	dbadmin
密码	-

参数名称	配置方式
启用SSL	不启用

步骤5 单击“确定”。

----结束

步骤三：导入交通卡口样例数据

使用SQL客户端工具连接到集群后，就可以在SQL客户端工具中，执行以下步骤导入交通卡口车辆通行的样例数据并执行查询。

步骤1 执行以下语句，创建traffic数据库。

```
CREATE DATABASE traffic encoding 'utf8' template template0;
```

步骤2 执行以下步骤切换为连接新建的数据库。

1. 在Data Studio客户端的**“对象浏览器”**窗口，右键单击数据库连接名称，在弹出菜单中单击“刷新”，刷新后就可以看到新建的数据库。
2. 右键单击“traffic”数据库名称，在弹出菜单中单击“打开连接”。
3. 右键单击“traffic”数据库名称，在弹出菜单中单击“打开新的终端”，即可打开连接到指定数据库的SQL命令窗口，后面的步骤，请全部在该命令窗口中执行。

步骤3 执行以下语句，创建用于存储卡口车辆信息的数据库表。

```
CREATE SCHEMA traffic_data;
SET current_schema= traffic_data;
DROP TABLE if exists GCJL;
CREATE TABLE GCJL
(
      kkbh   VARCHAR(20),
      hphm   VARCHAR(20),
      gcsj   DATE ,
      cplx   VARCHAR(8),
      cllx   VARCHAR(8),
      csys   VARCHAR(8)
)
with (orientation = column, COMPRESSION=MIDDLE)
distribute by hash(hphm);
```

步骤4 创建外表。外表用于识别和关联OBS上的源数据。

须知

- 其中，*<obs_bucket_name>*代表OBS桶名，仅支持部分区域，当前支持的区域和对应的OBS桶名请参见**支持区域**。GaussDB(DWS) 集群不支持跨区域访问OBS桶数据。
- 本实践以“华北-北京四”地区为例，可填入dws-demo-cn-north-4，<Access_Key_Id>和<Secret_Access_Key>替换为实际值，在**准备工作**获取。
- 认证用的AK和SK硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全。
- 创建外表如果提示“ERROR: schema "*xxx*" does not exist Position”，则说明schema不存在，请先参照上一步创建schema。

```
CREATE SCHEMA tpchobs;
SET current_schema = 'tpchobs';
DROP FOREIGN table if exists GCJL_OBS;
CREATE FOREIGN TABLE GCJL_OBS
(
    like traffic_data.GCJL
)
SERVER gsmpp_server
OPTIONS (
    encoding 'utf8',
    location 'obs://<obs_bucket_name>/traffic-data/gcxx',
    format 'text',
    delimiter ',',
    access_key '<Access_Key_Id>',
    secret_access_key '<Secret_Access_Key>',
    chunksize '64',
    IGNORE_EXTRA_DATA 'on'
);
```

步骤5 执行以下语句，将数据从外表导入到数据库表中。

```
INSERT INTO traffic_data.GCJL SELECT * FROM tpchobs.GCJL_OBS;
```

导入数据需要一些时间，请耐心等待。

----结束

步骤四：车辆分析

1. **执行Analyze**

   用于收集与数据库中普通表内容相关的统计信息，统计结果存储在系统表PG_STATISTIC中。执行计划生成器会使用这些统计数据，以生成最有效的查询执行计划。

   执行以下语句生成表统计信息：

   ```
   ANALYZE;
   ```

2. **查询数据表中的数据量**

   执行如下语句，可以查看已加载的数据条数。

   ```
   SET current_schema= traffic_data;
   SELECT count(*) FROM traffic_data.gcjl;
   ```

3. **车辆精确查询**

   执行以下语句，指定车牌号码和时间段查询车辆行驶路线。GaussDB(DWS) 在应对点查时秒级响应。

   ```
   SET current_schema= traffic_data;
   SELECT hphm, kkbh, gcsj
   FROM traffic_data.gcjl
   where hphm =  'YD38641'
   and gcsj between '2016-01-06' and '2016-01-07'
   order by gcsj desc;
   ```

4. **车辆模糊查询**

   执行以下语句，指定车牌号码和时间段查询车辆行驶路线，GaussDB(DWS) 在应对模糊查询时秒级响应。

   ```
   SET current_schema= traffic_data;
   SELECT hphm, kkbh, gcsj
   FROM traffic_data.gcjl
   where hphm like  'YA23F%'
   and kkbh in('508', '1125', '2120')
   and gcsj between '2016-01-01' and '2016-01-07'
   order by hphm,gcsj desc;
   ```

6.2 某公司供应链需求分析

本实践将演示从OBS加载样例数据集到GaussDB(DWS) 集群中并查询数据的流程，从而向您展示GaussDB(DWS) 在数据分析场景中的多表分析与主题分析。

说明

GaussDB(DWS) 已经预先生成了1GB的TPC-H-1x的标准数据集，已将数据集上传到了OBS桶的tpch文件夹中，并且已赋予所有华为云用户该OBS桶的只读访问权限，用户可以方便的进行导入。

操作流程

本实践预计时长60分钟，基本流程如下：

1. **准备工作**
2. **步骤一：导入公司样例数据**
3. **步骤二：多表分析与主题分析**

支持区域

当前已上传OBS数据的区域如**表6-6**所示。

表 6-6 区域和 OBS 桶名

区域	OBS桶名
华北-北京一	dws-demo-cn-north-1
华北-北京二	dws-demo-cn-north-2
华北-北京四	dws-demo-cn-north-4
华北-乌兰察布一	dws-demo-cn-north-9
华东-上海一	dws-demo-cn-east-3
华东-上海二	dws-demo-cn-east-2
华南-广州	dws-demo-cn-south-1
华南-广州友好	dws-demo-cn-south-4
中国-香港	dws-demo-ap-southeast-1
亚太-新加坡	dws-demo-ap-southeast-3
亚太-曼谷	dws-demo-ap-southeast-2
拉美-圣地亚哥	dws-demo-la-south-2
非洲-约翰内斯堡	dws-demo-af-south-1
拉美-墨西哥城一	dws-demo-na-mexico-1

区域	OBS桶名
拉美-墨西哥城二	dws-demo-la-north-2
莫斯科二	dws-demo-ru-northwest-2
拉美-圣保罗一	dws-demo-sa-brazil-1

场景描述

了解GaussDB(DWS)的基本功能和数据导入，对某公司与供应商的订单数据分析，分析维度如下：

1. 分析某地区供应商为公司带来的收入，通过该统计信息可用于决策在给定的区域是否需要建立一个当地分配中心。
2. 分析零件/供货商关系，可以获得能够以指定的贡献条件供应零件的供货商数量，通过该统计信息可用于决策在订单量大，任务紧急时，是否有充足的供货商。
3. 分析小订单收入损失，通过查询得知如果没有小量订单，平均年收入将损失多少。筛选出比平均供货量的20%还低的小批量订单，如果这些订单不再对外供货，由此计算平均一年的损失。

准备工作

- 已注册账号，且在使用GaussDB(DWS) 前检查账号状态，账号不能处于欠费或冻结状态。
- 获取此账号的“AK/SK”。
- 已创建集群，并已使用Data Studio连接集群，参见**交通卡口通行车辆分析**。

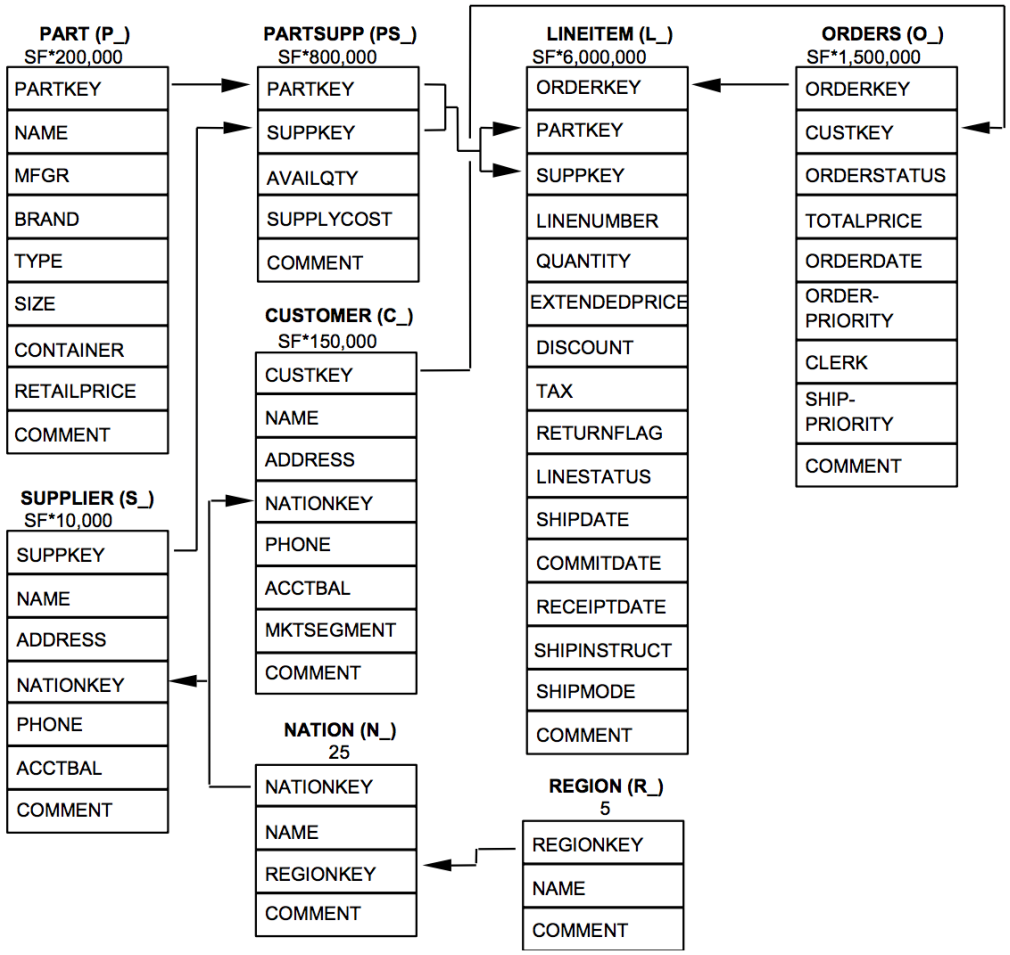
步骤一：导入公司样例数据

使用SQL客户端工具连接到集群后，就可以在SQL客户端工具中，执行以下步骤导入TPC-H样例数据并执行查询。

步骤1 创建数据库表。

TPC-H样例包含8张数据库表，其关联关系如**图6-1**所示。

图 6-1 TPC-H 数据表



复制并执行下列表创建语句，在gaussdb数据库中创建对应的数据表。

```
CREATE SCHEMA tpch;
SET current_schema = tpch;

DROP TABLE if exists region;
CREATE TABLE REGION
(
        R_REGIONKEY  INT NOT NULL ,
        R_NAME       CHAR(25) NOT NULL ,
        R_COMMENT    VARCHAR(152)
)
with (orientation = column, COMPRESSION=MIDDLE)
distribute by replication;

DROP TABLE if exists nation;
CREATE TABLE NATION
(
        N_NATIONKEY  INT NOT NULL,
        N_NAME       CHAR(25) NOT NULL,
        N_REGIONKEY  INT NOT NULL,
        N_COMMENT    VARCHAR(152)
)
with (orientation = column,COMPRESSION=MIDDLE)
distribute by replication;

DROP TABLE if exists supplier;
CREATE TABLE SUPPLIER
```



```
(
	S_SUPPKEY     BIGINT NOT NULL,
	S_NAME        CHAR(25) NOT NULL,
	S_ADDRESS     VARCHAR(40) NOT NULL,
	S_NATIONKEY   INT NOT NULL,
	S_PHONE       CHAR(15) NOT NULL,
	S_ACCTBAL     DECIMAL(15,2) NOT NULL,
	S_COMMENT     VARCHAR(101) NOT NULL
)
with (orientation = column,COMPRESSION=MIDDLE)
distribute by hash(S_SUPPKEY);

DROP TABLE if exists customer;
CREATE TABLE CUSTOMER
(
	C_CUSTKEY     BIGINT NOT NULL,
	C_NAME        VARCHAR(25) NOT NULL,
	C_ADDRESS     VARCHAR(40) NOT NULL,
	C_NATIONKEY   INT NOT NULL,
	C_PHONE       CHAR(15) NOT NULL,
	C_ACCTBAL     DECIMAL(15,2)   NOT NULL,
	C_MKTSEGMENT  CHAR(10) NOT NULL,
	C_COMMENT     VARCHAR(117) NOT NULL
)
with (orientation = column,COMPRESSION=MIDDLE)
distribute by hash(C_CUSTKEY);

DROP TABLE if exists part;
CREATE TABLE PART
(
	P_PARTKEY     BIGINT NOT NULL,
	P_NAME        VARCHAR(55) NOT NULL,
	P_MFGR        CHAR(25) NOT NULL,
	P_BRAND       CHAR(10) NOT NULL,
	P_TYPE        VARCHAR(25) NOT NULL,
	P_SIZE        BIGINT NOT NULL,
	P_CONTAINER   CHAR(10) NOT NULL,
	P_RETAILPRICE DECIMAL(15,2) NOT NULL,
	P_COMMENT     VARCHAR(23) NOT NULL
)
with (orientation = column,COMPRESSION=MIDDLE)
distribute by hash(P_PARTKEY);

DROP TABLE if exists partsupp;
CREATE TABLE PARTSUPP
(
	PS_PARTKEY     BIGINT NOT NULL,
	PS_SUPPKEY     BIGINT NOT NULL,
	PS_AVAILQTY    BIGINT NOT NULL,
	PS_SUPPLYCOST  DECIMAL(15,2)  NOT NULL,
	PS_COMMENT     VARCHAR(199) NOT NULL
)
with (orientation = column,COMPRESSION=MIDDLE)
distribute by hash(PS_PARTKEY);

DROP TABLE if exists orders;
CREATE TABLE ORDERS
(
	O_ORDERKEY       BIGINT NOT NULL,
	O_CUSTKEY        BIGINT NOT NULL,
	O_ORDERSTATUS    CHAR(1) NOT NULL,
	O_TOTALPRICE     DECIMAL(15,2) NOT NULL,
	O_ORDERDATE      DATE NOT NULL ,
	O_ORDERPRIORITY  CHAR(15) NOT NULL,
	O_CLERK          CHAR(15) NOT NULL ,
	O_SHIPPRIORITY   BIGINT NOT NULL,
	O_COMMENT        VARCHAR(79) NOT NULL
)
with (orientation = column,COMPRESSION=MIDDLE)
```

```
distribute by hash(O_ORDERKEY);

DROP TABLE if exists lineitem;
CREATE TABLE LINEITEM
(
        L_ORDERKEY    BIGINT NOT NULL,
        L_PARTKEY     BIGINT NOT NULL,
        L_SUPPKEY     BIGINT NOT NULL,
        L_LINENUMBER  BIGINT NOT NULL,
        L_QUANTITY    DECIMAL(15,2) NOT NULL,
        L_EXTENDEDPRICE  DECIMAL(15,2) NOT NULL,
        L_DISCOUNT    DECIMAL(15,2) NOT NULL,
        L_TAX         DECIMAL(15,2) NOT NULL,
        L_RETURNFLAG  CHAR(1) NOT NULL,
        L_LINESTATUS  CHAR(1) NOT NULL,
        L_SHIPDATE    DATE NOT NULL,
        L_COMMITDATE  DATE NOT NULL ,
        L_RECEIPTDATE DATE NOT NULL,
        L_SHIPINSTRUCT CHAR(25) NOT NULL,
        L_SHIPMODE     CHAR(10) NOT NULL,
        L_COMMENT      VARCHAR(44) NOT NULL
)
with (orientation = column,COMPRESSION=MIDDLE)
distribute by hash(L_ORDERKEY);
```

步骤2 创建外表。外表用于识别和关联OBS上的源数据。

须知

- 其中，*<obs_bucket_name>*代表OBS桶名，仅支持部分区域，当前支持的区域和对应的OBS桶名请参见**支持区域**。GaussDB(DWS) 集群不支持跨区域访问OBS桶数据。
- 本实践以“华北-北京四”地区为例，可填入dws-demo-cn-north-4，<Access_Key_Id>和<Secret_Access_Key>替换为实际值，在**准备工作**获取。
- 认证用的AK和SK硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全。
- 创建外表如果提示“ERROR: schema "*xxx*" does not exist Position”，则说明schema不存在，请先参照上一步创建schema。

```
CREATE SCHEMA tpchobs;
SET current_schema='tpchobs';
DROP FOREIGN table if exists region;
CREATE FOREIGN TABLE REGION
(
        like tpch.region
)
SERVER gsmpp_server
OPTIONS (
        encoding 'utf8',
        location 'obs://<obs_bucket_name>/tpch/region.tbl',
        format 'text',
        delimiter '|',
        access_key '<Access_Key_Id>',
        secret_access_key '<Secret_Access_Key>',
        chunksize '64',
        IGNORE_EXTRA_DATA 'on'
);

DROP FOREIGN table if exists nation;
CREATE FOREIGN TABLE NATION
(
        like tpch.nation
```

```
)
SERVER gsmpp_server
OPTIONS (
        encoding 'utf8',
        location 'obs://<obs_bucket_name>/tpch/nation.tbl',
        format 'text',
        delimiter '|',
        access_key '<Access_Key_Id>',
        secret_access_key '<Secret_Access_Key>',
        chunksize '64',
        IGNORE_EXTRA_DATA 'on'
);

DROP FOREIGN table if exists supplier;
CREATE FOREIGN TABLE SUPPLIER
(
      like tpch.supplier
)
SERVER gsmpp_server
OPTIONS (
      encoding 'utf8',
      location 'obs://<obs_bucket_name>/tpch/supplier.tbl',
      format 'text',
      delimiter '|',
      access_key '<Access_Key_Id>',
      secret_access_key '<Secret_Access_Key>',
      chunksize '64',
      IGNORE_EXTRA_DATA 'on'
);

DROP FOREIGN table if exists customer;
CREATE FOREIGN TABLE CUSTOMER
(
      like tpch.customer
)
SERVER gsmpp_server
OPTIONS (
      encoding 'utf8',
      location 'obs://<obs_bucket_name>/tpch/customer.tbl',
      format 'text',
      delimiter '|',
      access_key '<Access_Key_Id>',
      secret_access_key '<Secret_Access_Key>',
      chunksize '64',
      IGNORE_EXTRA_DATA 'on'
);
DROP FOREIGN table if exists part;
CREATE FOREIGN TABLE PART
(
      like tpch.part

)
SERVER gsmpp_server
OPTIONS (
      encoding 'utf8',
      location 'obs://<obs_bucket_name>/tpch/part.tbl',
      format 'text',
      delimiter '|',
      access_key '<Access_Key_Id>',
      secret_access_key '<Secret_Access_Key>',
      chunksize '64',
      IGNORE_EXTRA_DATA 'on'
);
DROP FOREIGN table if exists partsupp;
CREATE FOREIGN TABLE PARTSUPP
(
      like tpch.partsupp
)
SERVER gsmpp_server
```



```
OPTIONS (
        encoding 'utf8',
        location 'obs://<obs_bucket_name>/tpch/partsupp.tbl',
        format 'text',
        delimiter '|',
        access_key '<Access_Key_Id>',
        secret_access_key '<Secret_Access_Key>',
        chunksize '64',
        IGNORE_EXTRA_DATA 'on'
);
DROP FOREIGN table if exists orders;
CREATE FOREIGN TABLE ORDERS
(
        like tpch.orders
)
SERVER gsmpp_server
OPTIONS (
        encoding 'utf8',
        location 'obs://<obs_bucket_name>/tpch/orders.tbl',
        format 'text',
        delimiter '|',
        access_key '<Access_Key_Id>',
        secret_access_key '<Secret_Access_Key>',
        chunksize '64',
        IGNORE_EXTRA_DATA 'on'
);
DROP FOREIGN table if exists lineitem;
CREATE FOREIGN TABLE LINEITEM
(
        like tpch.lineitem
)
SERVER gsmpp_server
OPTIONS (
        encoding 'utf8',
        location 'obs://<obs_bucket_name>/tpch/lineitem.tbl',
        format 'text',
        delimiter '|',
        access_key '<Access_Key_Id>',
        secret_access_key '<Secret_Access_Key>',
        chunksize '64',
        IGNORE_EXTRA_DATA 'on'
);
```

步骤3 复制并执行以下语句，将外表数据导入到对应的数据库表中。

将OBS外表的数据通过insert命令导入GaussDB(DWS) 的数据库表中，数据库内核对应的操作为OBS数据高速并发导入GaussDB(DWS) 。

```
INSERT INTO tpch.lineitem SELECT * FROM tpchobs.lineitem;
INSERT INTO tpch.part SELECT * FROM tpchobs.part;
INSERT INTO tpch.partsupp SELECT * FROM tpchobs.partsupp;
INSERT INTO tpch.customer SELECT * FROM tpchobs.customer;
INSERT INTO tpch.supplier SELECT * FROM tpchobs.supplier;
INSERT INTO tpch.nation SELECT * FROM tpchobs.nation;
INSERT INTO tpch.region SELECT * FROM tpchobs.region;
INSERT INTO tpch.orders SELECT * FROM tpchobs.orders;
```

导入数据需要约10分钟，请耐心等待。

----结束

步骤二：多表分析与主题分析

以下以TPC-H标准查询为例，演示在GaussDB(DWS) 中进行的基本数据查询。

在进行数据查询之前，请先执行“Analyze”命令生成与数据库表相关的统计信息。统计信息存储在系统表PG_STATISTIC中，执行计划生成器会使用这些统计数据，以生成最有效的查询执行计划。

查询示例如下：

- **某地区供货商为公司带来的收入查询（TPCH-Q5）**

  通过执行TPCH-Q5查询语句，可以查询到通过某个地区零件供货商获得的收入（收入按sum(l_extendedprice * (1 - l_discount))计算）统计信息。该统计信息可用于决策在给定的区域是否需要建立一个当地分配中心。

  复制并执行以下TPCH-Q5语句进行查询。该语句的特点是：带有分组、排序、聚集操作并存的多表连接查询操作。

```
SET current_schema='tpch';
SELECT
n_name,
sum(l_extendedprice * (1 - l_discount)) as revenue
FROM
customer,
orders,
lineitem,
supplier,
nation,
region
where
c_custkey = o_custkey
and l_orderkey = o_orderkey
and l_suppkey = s_suppkey
and c_nationkey = s_nationkey
and s_nationkey = n_nationkey
and n_regionkey = r_regionkey
and r_name = 'ASIA'
and o_orderdate >= '1994-01-01'::date
and o_orderdate < '1994-01-01'::date + interval '1 year'
group by
n_name
order by
revenue desc;
```

- **零件/供货商关系查询（TPCH-Q16）**

  通过执行TPCH-Q16查询语句，可以获得能够以指定的贡献条件供应零件的供货商数量。该信息可用于决策在订单量大，任务紧急时，是否有充足的供货商。

  复制并执行以下TPCH-Q16语句进行查询，该语句的特点是：带有分组、排序、聚集、去重、NOT IN子查询操作并存的多表连接操作。

```
SET current_schema='tpch';
SELECT
p_brand,
p_type,
p_size,
count(distinct ps_suppkey) as supplier_cnt
FROM
partsupp,
part
where
p_partkey = ps_partkey
and p_brand <> 'Brand#45'
and p_type not like 'MEDIUM POLISHED%'
and p_size in (49, 14, 23, 45, 19, 3, 36, 9)
and ps_suppkey not in (
        select
        s_suppkey
        from
        supplier
        where
        s_comment like '%Customer%Complaints%'
)
group by
p_brand,
p_type,
```



```
p_size
order by
supplier_cnt desc,
p_brand,
p_type,
p_size
limit 100;
```

- **小订单收入损失查询（TPCH-Q17）**

  通过查询得知如果没有小量订单，平均年收入将损失多少。筛选出比平均供货量的20%还低的小批量订单，如果这些订单不再对外供货，由此计算平均一年的损失。

  复制并执行以下TPCH-Q17语句进行查询，该语句的特点是：带有聚集、聚集子查询操作并存的两表连接操作。

```
SET current_schema='tpch';
SELECT
sum(l_extendedprice) / 7.0 as avg_yearly
FROM
lineitem,
part
where
p_partkey = l_partkey
and p_brand = 'Brand#23'
and p_container = 'MED BOX'
and l_quantity < (
        select 0.2 * avg(l_quantity)
        from lineitem
        where l_partkey = p_partkey
);
```

6.3 零售业百货公司经营状况分析

零售业百货公司样例简介

本实践将演示以下场景：从OBS加载各个零售商场每日经营的业务数据到数据仓库对应的表中，然后对商铺营业额、客流信息、月度销售排行、月度客流转化率、月度租售比、销售坪效等KPI信息进行汇总和查询。本示例旨在展示在零售业场景中GaussDB(DWS) 数据仓库的多维度查询分析的能力。

说明

GaussDB(DWS) 已预先将样例数据上传到OBS桶的“retail-data”文件夹中，并给所有华为云用户赋予了该OBS桶的只读访问权限。

操作流程

本实践预计时长60分钟，基本流程如下：

1. **准备工作**
2. **步骤一：导入零售业百货公司样例数据**
3. **步骤二：经营状况分析**

支持区域

当前已上传OBS数据的区域如**表6-7**所示。

表 6-7 区域和 OBS 桶名

区域	OBS桶名
华北-北京一	dws-demo-cn-north-1
华北-北京二	dws-demo-cn-north-2
华北-北京四	dws-demo-cn-north-4
华北-乌兰察布一	dws-demo-cn-north-9
华东-上海一	dws-demo-cn-east-3
华东-上海二	dws-demo-cn-east-2
华南-广州	dws-demo-cn-south-1
华南-广州友好	dws-demo-cn-south-4
中国-香港	dws-demo-ap-southeast-1
亚太-新加坡	dws-demo-ap-southeast-3
亚太-曼谷	dws-demo-ap-southeast-2
拉美-圣地亚哥	dws-demo-la-south-2
非洲-约翰内斯堡	dws-demo-af-south-1
拉美-墨西哥城一	dws-demo-na-mexico-1
拉美-墨西哥城二	dws-demo-la-north-2
莫斯科二	dws-demo-ru-northwest-2
拉美-圣保罗一	dws-demo-sa-brazil-1

准备工作

- 已注册账号，账号不能处于欠费或冻结状态。
- 获取此账号的“AK/SK”。
- 已创建集群，并已使用Data Studio连接集群，参见**步骤一：创建集群**和**步骤二：使用Data Studio连接集群**。

步骤一：导入零售业百货公司样例数据

使用SQL客户端工具连接到集群后，就可以在SQL客户端工具中，执行以下步骤导入零售业百货公司样例数据并执行查询。

步骤1 执行以下语句，创建retail数据库。

```
CREATE DATABASE retail encoding 'utf8' template template0;
```

步骤2 执行以下步骤切换为连接新建的数据库。

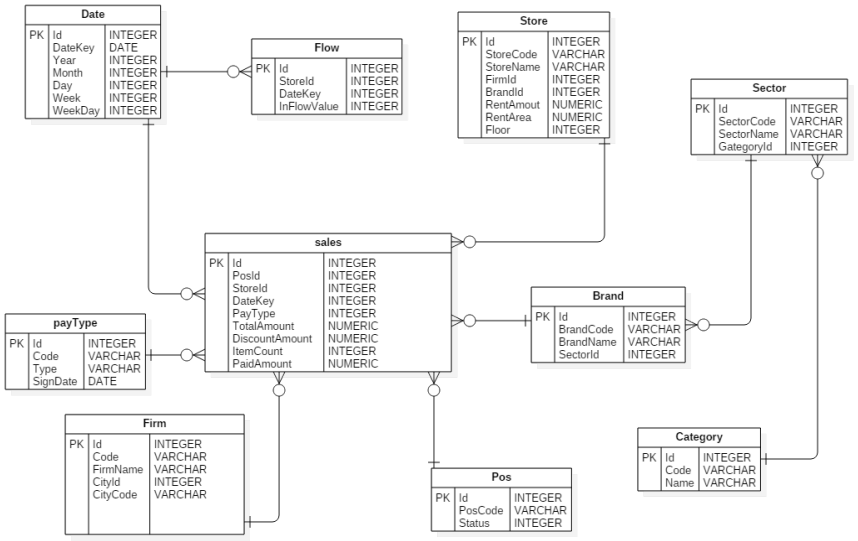
1. 在Data Studio客户端的**“对象浏览器”**窗口，右键单击数据库连接名称，在弹出菜单中单击“刷新”，刷新后就可以看到新建的数据库。

2. 右键单击“retail”数据库名称，在弹出菜单中单击“打开连接”。
3. 右键单击“retail”数据库名称，在弹出菜单中单击“打开新的终端”，即可打开连接到指定数据库的SQL命令窗口，后面的步骤，请全部在该命令窗口中执行。

步骤3 创建数据库表。

样例数据包含10张数据库表，其关联关系如图6-2所示。

图 6-2 百货公司样例数据表



复制并执行以下语句，创建零售业百货公司信息数据库表。

```
CREATE SCHEMA retail_data;
SET current_schema='retail_data';

DROP TABLE IF EXISTS STORE;
CREATE TABLE STORE (
        ID INT,
        STORECODE VARCHAR(10),
        STORENAME VARCHAR(100),
        FIRMID INT,
        FLOOR INT,
        BRANDID INT,
        RENTAMOUNT NUMERIC(18,2),
        RENTAREA NUMERIC(18,2)
)
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY REPLICATION;

DROP TABLE IF EXISTS POS;
CREATE TABLE POS(
        ID INT,
        POSCODE VARCHAR(20),
        STATUS INT,
        MODIFICATIONDATE DATE
)
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY REPLICATION;

DROP TABLE IF EXISTS BRAND;
CREATE TABLE BRAND (
        ID INT,
        BRANDCODE VARCHAR(10),
        BRANDNAME VARCHAR(100),
        SECTORID INT
)
```



```
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY REPLICATION;

DROP TABLE IF EXISTS SECTOR;
CREATE TABLE SECTOR(
	ID INT,
	SECTORCODE VARCHAR(10),
	SECTORNAME VARCHAR(20),
	CATEGORYID INT
)
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY REPLICATION;

DROP TABLE IF EXISTS CATEGORY;
CREATE TABLE CATEGORY(
	ID INT,
	CODE VARCHAR(10),
	NAME VARCHAR(20)
)
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY REPLICATION;

DROP TABLE IF EXISTS FIRM;
CREATE TABLE FIRM(
	ID INT,
	CODE VARCHAR(4),
	NAME VARCHAR(40),
	CITYID INT,
	CITYNAME VARCHAR(10),
	CITYCODE VARCHAR(20)
)
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY REPLICATION;

DROP TABLE IF EXISTS DATE;
CREATE TABLE DATE(
	ID INT,
	DATEKEY DATE,
	YEAR INT,
	MONTH INT,
	DAY INT,
	WEEK INT,
	WEEKDAY INT
)
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY REPLICATION;

DROP TABLE IF EXISTS PAYTYPE;
CREATE TABLE PAYTYPE(
	ID INT,
	CODE VARCHAR(10),
	TYPE VARCHAR(10),
	SIGNDATE DATE
)
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY REPLICATION;

DROP TABLE IF EXISTS SALES;
CREATE TABLE SALES(
	 ID INT,
	 POSID INT,
	 STOREID INT,
	 DATEKEY INT,
	 PAYTYPE INT,
	 TOTALAMOUNT NUMERIC(18,2),
	 DISCOUNTAMOUNT NUMERIC(18,2),
	 ITEMCOUNT INT,
	 PAIDAMOUNT NUMERIC(18,2)
)
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY HASH(ID);

DROP TABLE IF EXISTS FLOW;
CREATE TABLE FLOW (
	ID INT,
	STOREID INT,
```

```
    DATEKEY INT,
    INFLOWVALUE INT
)
WITH (ORIENTATION = COLUMN, COMPRESSION=MIDDLE) DISTRIBUTE BY HASH(ID);
```

步骤4 创建外表。外表用于识别和关联OBS上的源数据。

须知

- 其中，*<obs_bucket_name>*代表OBS桶名，仅支持部分区域，当前支持的区域和对应的OBS桶名请参见**支持区域**。GaussDB(DWS) 集群不支持跨区域访问OBS桶数据。
- 本实践以“华北-北京四”地区为例，可填入dws-demo-cn-north-4，<Access_Key_Id>和<Secret_Access_Key>替换为实际值，在**准备工作**获取。
- 认证用的AK和SK硬编码到代码中或者明文存储都有很大的安全风险，建议在配置文件或者环境变量中密文存放，使用时解密，确保安全。
- 创建外表如果提示“ERROR: schema "*xxx*" does not exist Position”，则说明schema不存在，请先参照上一步创建schema。

```
CREATE SCHEMA retail_obs_data;
SET current_schema='retail_obs_data';
DROP FOREIGN table if exists SALES_OBS;
CREATE FOREIGN TABLE SALES_OBS
(
    like retail_data.SALES
)
SERVER gsmpp_server
OPTIONS (
    encoding 'utf8',
    location 'obs://<obs_bucket_name>/retail-data/sales',
    format 'csv',
    delimiter ',',
    access_key '<Access_Key_Id>',
    secret_access_key '<Secret_Access_Key>',
    chunksize '64',
    IGNORE_EXTRA_DATA 'on',
    header 'on'
);

DROP FOREIGN table if exists FLOW_OBS;
CREATE FOREIGN TABLE FLOW_OBS
(
    like retail_data.flow
)
SERVER gsmpp_server
OPTIONS (
    encoding 'utf8',
    location 'obs://<obs_bucket_name>/retail-data/flow',
    format 'csv',
    delimiter ',',
    access_key '<Access_Key_Id>',
    secret_access_key '<Secret_Access_Key>',
    chunksize '64',
    IGNORE_EXTRA_DATA 'on',
    header 'on'
);

DROP FOREIGN table if exists BRAND_OBS;
CREATE FOREIGN TABLE BRAND_OBS
(
    like retail_data.brand
)
```



```
SERVER gsmpp_server
OPTIONS (
        encoding 'utf8',
        location 'obs://<obs_bucket_name>/retail-data/brand',
        format 'csv',
        delimiter ',',
        access_key '<Access_Key_Id>',
        secret_access_key '<Secret_Access_Key>',
        chunksize '64',
        IGNORE_EXTRA_DATA 'on',
        header 'on'
);


DROP FOREIGN table if exists CATEGORY_OBS;
CREATE FOREIGN TABLE CATEGORY_OBS
(
      like retail_data.category
)
SERVER gsmpp_server
OPTIONS (
        encoding 'utf8',
        location 'obs://<obs_bucket_name>/retail-data/category',
        format 'csv',
        delimiter ',',
        access_key '<Access_Key_Id>',
        secret_access_key '<Secret_Access_Key>',
        chunksize '64',
        IGNORE_EXTRA_DATA 'on',
        header 'on'
);

DROP FOREIGN table if exists DATE_OBS;
CREATE FOREIGN TABLE DATE_OBS
(
       like retail_data.date
)
SERVER gsmpp_server
OPTIONS (
        encoding 'utf8',
        location 'obs://<obs_bucket_name>/retail-data/date',
        format 'csv',
        delimiter ',',
        access_key '<Access_Key_Id>',
        secret_access_key '<Secret_Access_Key>',
        chunksize '64',
        IGNORE_EXTRA_DATA 'on',
        header 'on'
);

DROP FOREIGN table if exists FIRM_OBS;
CREATE FOREIGN TABLE FIRM_OBS
(
       like retail_data.firm
)
SERVER gsmpp_server
OPTIONS (
        encoding 'utf8',
        location 'obs://<obs_bucket_name>/retail-data/firm',
        format 'csv',
        delimiter ',',
        access_key '<Access_Key_Id>',
        secret_access_key '<Secret_Access_Key>',
        chunksize '64',
        IGNORE_EXTRA_DATA 'on',
        header 'on'
);
```



```
DROP FOREIGN table if exists PAYTYPE_OBS;
CREATE FOREIGN TABLE PAYTYPE_OBS
(
    like retail_data.paytype
)
SERVER gsmpp_server
OPTIONS (
    encoding 'utf8',
    location 'obs://<obs_bucket_name>/retail-data/paytype',
    format 'csv',
    delimiter ',',
    access_key '<Access_Key_Id>',
    secret_access_key '<Secret_Access_Key>',
    chunksize '64',
    IGNORE_EXTRA_DATA 'on',
    header 'on'
);


DROP FOREIGN table if exists POS_OBS;
CREATE FOREIGN TABLE POS_OBS
(
    like retail_data.pos
)
SERVER gsmpp_server
OPTIONS (
    encoding 'utf8',
    location 'obs://<obs_bucket_name>/retail-data/pos',
    format 'csv',
    delimiter ',',
    access_key '<Access_Key_Id>',
    secret_access_key '<Secret_Access_Key>',
    chunksize '64',
    IGNORE_EXTRA_DATA 'on',
    header 'on'
);

DROP FOREIGN table if exists SECTOR_OBS;
CREATE FOREIGN TABLE SECTOR_OBS
(
    like retail_data.sector
)
SERVER gsmpp_server
OPTIONS (
    encoding 'utf8',
    location 'obs://<obs_bucket_name>/retail-data/sector',
    format 'csv',
    delimiter ',',
    access_key '<Access_Key_Id>',
    secret_access_key '<Secret_Access_Key>',
    chunksize '64',
    IGNORE_EXTRA_DATA 'on',
    header 'on'
);


DROP FOREIGN table if exists STORE_OBS;
CREATE FOREIGN TABLE STORE_OBS
(
     like retail_data.store
)
SERVER gsmpp_server
OPTIONS (
     encoding 'utf8',
     location 'obs://<obs_bucket_name>/retail-data/store',
     format 'csv',
     delimiter ',',
     access_key '<Access_Key_Id>',
     secret_access_key '<Secret_Access_Key>',
```



```
        chunksize '64',
        IGNORE_EXTRA_DATA 'on',
        header 'on'
);
```

步骤5 复制并执行以下语句，导入外表数据到集群。

```
INSERT INTO retail_data.store SELECT * FROM retail_obs_data.STORE_OBS;
INSERT INTO retail_data.sector SELECT * FROM retail_obs_data.SECTOR_OBS;
INSERT INTO retail_data.paytype SELECT * FROM retail_obs_data.PAYTYPE_OBS;
INSERT INTO retail_data.firm SELECT * FROM retail_obs_data.FIRM_OBS;
INSERT INTO retail_data.flow SELECT * FROM retail_obs_data.FLOW_OBS;
INSERT INTO retail_data.category SELECT * FROM retail_obs_data.CATEGORY_OBS;
INSERT INTO retail_data.date SELECT * FROM retail_obs_data.DATE_OBS;
INSERT INTO retail_data.pos SELECT * FROM retail_obs_data.POS_OBS;
INSERT INTO retail_data.brand SELECT * FROM retail_obs_data.BRAND_OBS;
INSERT INTO retail_data.sales SELECT * FROM retail_obs_data.SALES_OBS;
```

导入数据需要一些时间，请耐心等待。

步骤6 复制并执行以下语句，创建视图v_sales_flow_details。

```
SET current_schema='retail_data';
CREATE VIEW v_sales_flow_details AS
SELECT
FIRM.ID FIRMID, FIRM.NAME FIRNAME, FIRM. CITYCODE,
CATEGORY.ID CATEGORYID, CATEGORY.NAME CATEGORYNAME,
SECTOR.ID SECTORID, SECTOR.SECTORNAME,
BRAND.ID BRANDID, BRAND.BRANDNAME,
STORE.ID STOREID, STORE.STORENAME, STORE.RENTAMOUNT, STORE.RENTAREA,
DATE.DATEKEY, SALES.TOTALAMOUNT, DISCOUNTAMOUNT, ITEMCOUNT, PAIDAMOUNT, INFLOWVALUE
FROM SALES
INNER JOIN STORE ON SALES.STOREID = STORE.ID
INNER JOIN FIRM ON STORE.FIRMID = FIRM.ID
INNER JOIN BRAND ON STORE.BRANDID = BRAND.ID
INNER JOIN SECTOR ON BRAND.SECTORID = SECTOR.ID
INNER JOIN CATEGORY ON SECTOR.CATEGORYID = CATEGORY.ID
INNER JOIN DATE ON SALES.DATEKEY = DATE.ID
INNER JOIN FLOW ON FLOW.DATEKEY = DATE.ID AND FLOW.STOREID = STORE.ID;
```

----结束

步骤二：经营状况分析

以下以零售百货公司标准查询为例，演示在GaussDB(DWS) 中进行的基本数据查询。

在进行数据查询之前，请先执行“Analyze”命令生成与数据库表相关的统计信息。统计信息存储在系统表PG_STATISTIC中，执行计划生成器会使用这些统计数据，以生成最有效的查询执行计划。

查询示例如下：

- **查询各商铺的月度营业额**

  复制并执行以下语句查询各商铺的月度营业额。

  ```
  SET current_schema='retail_data';
  SELECT DATE_TRUNC('month',datekey)
  AT TIME ZONE 'UTC' AS __timestamp,
  SUM(paidamount)
  AS sum__paidamount
  FROM v_sales_flow_details
  GROUP BY DATE_TRUNC('month',datekey) AT TIME ZONE 'UTC'
  ORDER BY SUM(paidamount) DESC;
  ```

- **查询各门店营收及租售比状况**

  复制并执行以下语句进行营收及租售比状况查询。

  ```
  SET current_schema='retail_data';
  SELECT firname AS firname,
  ```



```
storename AS storename,
SUM(paidamount)
AS sum__paidamount,
AVG(RENTAMOUNT)/SUM(PAIDAMOUNT)
AS rentamount_sales_rate
FROM v_sales_flow_details
GROUP BY firname, storename
ORDER BY SUM(paidamount) DESC;
```

- **各城市营业汇总分析**

  复制并执行以下语句进行汇总分析查询。

```
SET current_schema='retail_data';
SELECT citycode AS citycode,
SUM(paidamount)
AS sum__paidamount
FROM v_sales_flow_details
GROUP BY citycode
ORDER BY SUM(paidamount) DESC;
```

- **各门店租售比和客流转化率对比分析**

```
SET current_schema='retail_data';
SELECT brandname AS brandname,
firname AS firname,
SUM(PAIDAMOUNT)/AVG(RENTAREA) AS sales_rentarea_rate,
SUM(ITEMCOUNT)/SUM(INFLOWVALUE) AS poscount_flow_rate,
AVG(RENTAMOUNT)/SUM(PAIDAMOUNT) AS rentamount_sales_rate
FROM v_sales_flow_details
GROUP BY brandname,  firname
ORDER BY sales_rentarea_rate DESC;
```

- **品牌业态分析**

```
SET current_schema='retail_data';
SELECT categoryname AS categoryname,
brandname AS brandname,
SUM(paidamount) AS sum__paidamount
FROM v_sales_flow_details
GROUP BY categoryname,
brandname
ORDER BY sum__paidamount DESC;
```

- **查询各品牌每日营业状况**

```
SET current_schema='retail_data';
SELECT brandname AS brandname,
DATE_TRUNC('day', datekey) AT TIME ZONE 'UTC' AS __timestamp,
SUM(paidamount) AS sum__paidamount
FROM v_sales_flow_details
WHERE datekey >= '2016-01-01 00:00:00'
AND datekey <= '2016-01-30 00:00:00'
GROUP BY brandname,
DATE_TRUNC('day', datekey) AT TIME ZONE 'UTC'
ORDER BY sum__paidamount ASC
LIMIT 50000;
```

7 安全管理

7.1 基于角色的权限管理(RBAC)

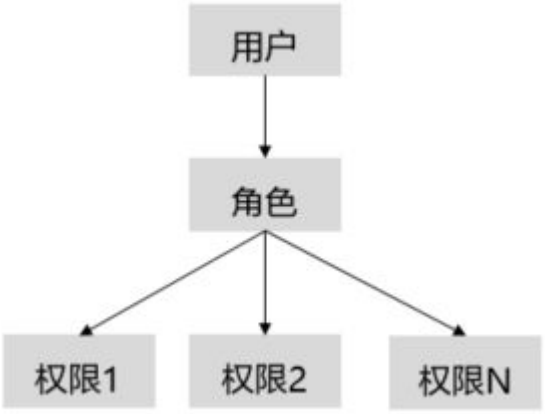
什么是基于角色的用户管理?

- 基于角色的用户管理（Role-Based Access Control，简称RBAC）是通过为角色赋予权限，使用户成为适当的角色而获取相应角色的权限。
- 角色是一组权限的抽象。
- 使用RBAC可以极大简化对权限的管理。

什么是 RBAC 模型?

为角色赋予适当的权限。

指定用户为相应的角色。

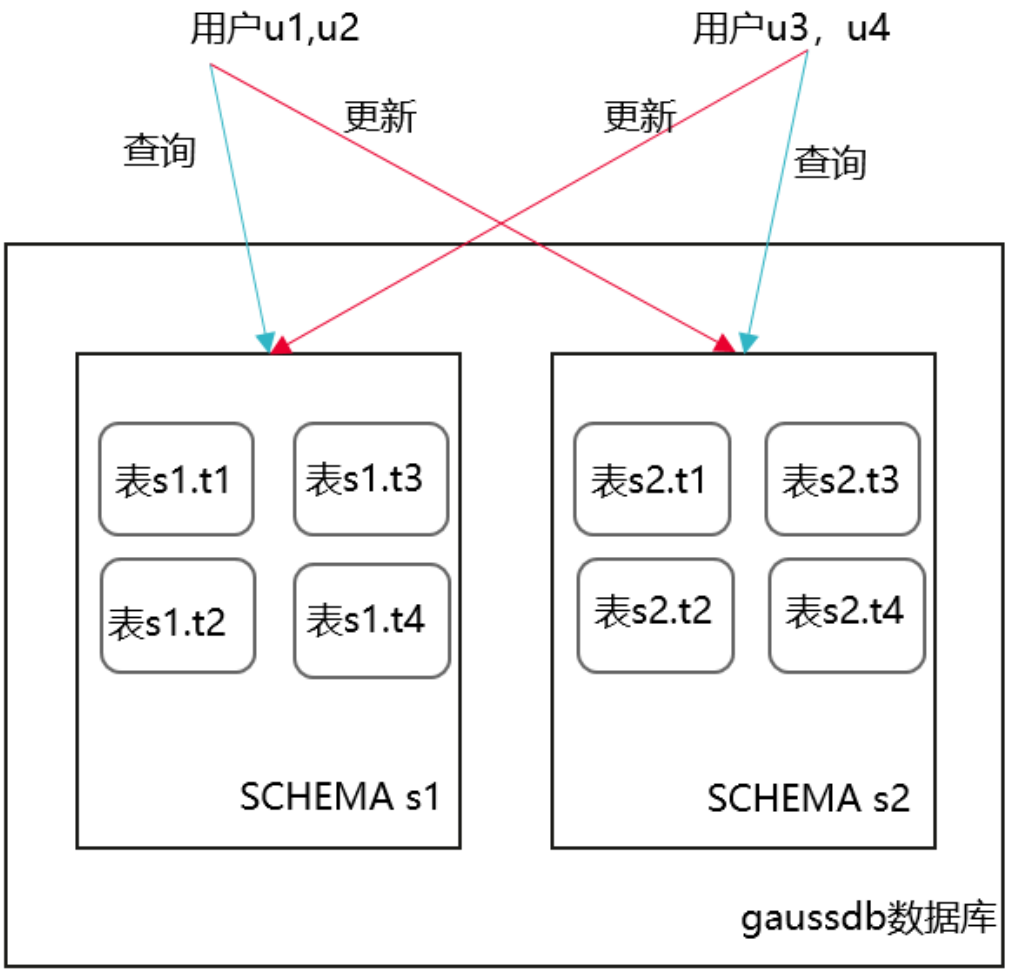


场景介绍

假设有两个SCHEMA：s1, s2。

有两组用户：

- 一组用户包括u1, u2，可以在s1中查询所有表，在s2中更新所有表。
- 另一组用户包括u3, u4，可以在s2中查询所有表，在s1中更新所有表。



授予权限操作步骤

步骤1 使用系统管理员dbadmin连接GaussDB(DWS)数据库。

步骤2 复制以下语句在窗口1中执行，创建本用例的SCHEMA s1和s2，用户u1~u4。

说明

示例中*{password}*请替换成实际密码。

```
CREATE SCHEMA s1;
CREATE SCHEMA s2;
CREATE USER u1 PASSWORD '{password}';
CREATE USER u2 PASSWORD '{password}';
CREATE USER u3 PASSWORD '{password}';
CREATE USER u4 PASSWORD '{password}';
```

步骤3 复制以下语句在窗口1中执行，创建对应的s1.t1，s2.t1表。

```
CREATE TABLE s1.t1 (c1 int, c2 int);
CREATE TABLE s2.t1 (c1 int, c2 int);
```

步骤4 复制以下语句在窗口1中执行，为表插入数据。

```
INSERT INTO s1.t1 VALUES (1,2);
INSERT INTO s2.t1 VALUES (1,2);
```


步骤5 复制以下语句在窗口1中执行，创建4个角色。分别对应s1的查询权限、s1的更新权限、s2的查询权限、s2的更新权限。

```
CREATE ROLE rs1_select PASSWORD disable;  --s1的查询权限
CREATE ROLE rs1_update PASSWORD disable;  --s1的更新权限
CREATE ROLE rs2_select PASSWORD disable;  --s2的查询权限
CREATE ROLE rs2_update PASSWORD disable;  --s2的更新权限
```

步骤6 复制以下语句在窗口1中执行，将SCHEMA s1和s2的访问权限先授予这些角色。

```
GRANT USAGE ON SCHEMA s1, s2 TO rs1_select, rs1_update,rs2_select, rs2_update;
```

步骤7 复制以下语句在窗口1中执行，将具体的权限授予这些角色。

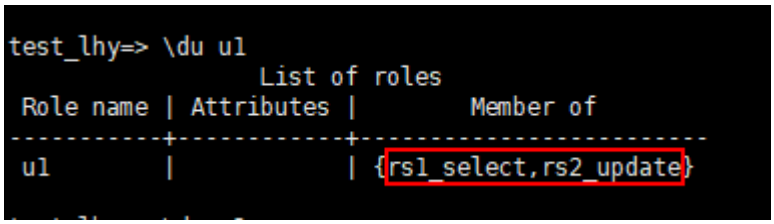
```
GRANT SELECT ON ALL TABLES IN SCHEMA s1 TO rs1_select; --将s1下的所有表的查询权限授予角色rs1_select
GRANT SELECT,UPDATE ON ALL TABLES IN SCHEMA s1 TO rs1_update;  --将s1下的所有表的查询、更新权限授予角色rs1_update
GRANT SELECT ON ALL TABLES IN SCHEMA s2 TO rs2_select;  --将s2下的所有表的查询权限授予角色rs2_select
GRANT SELECT,UPDATE ON ALL TABLES IN SCHEMA s2 TO rs2_update;  --将s2下的所有表的查询、更新权限授予角色rs2_update
```

步骤8 复制以下语句在窗口1中执行，将对应的角色授予对应的用户，实现将一组权限授予用户。

```
GRANT rs1_select, rs2_update TO u1, u2;  --u1，u2可以对s1的查询权限、对s2的更新权限。
GRANT rs2_select, rs1_update TO u3, u4;  --u3，u4可以对s2的查询权限、对s1的更新权限。
```

步骤9 复制以下语句在窗口1中执行，可以查看指定用户绑定的角色。

```
\du u1;
```

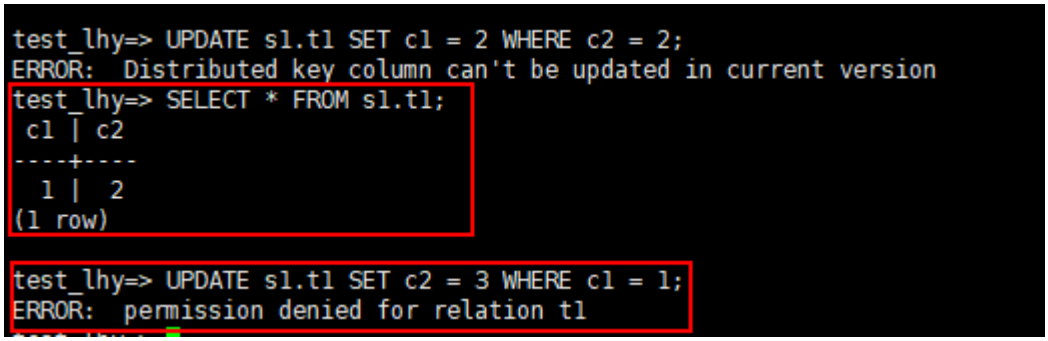


步骤10 重新打开一个会话窗口2，以用户u1连接DWS数据库。

gsql -d gaussdb -h *<DWS的公网IP>* **-U u1 -p 8000 -r -W** *{password}*;

步骤11 复制以下语句在窗口2中执行，验证用户u1对s1.t1有查询权限而没有更新权限。

```
SELECT * FROM s1.t1;
UPDATE s1.t1 SET c2 = 3 WHERE c1 = 1;
```



步骤12 复制以下语句在窗口2中执行，验证用户u1对s2.t1有更新权限。

```
SELECT * FROM s2.t1;
UPDATE s2.t1 SET c2 = 3 WHERE c1 = 1;
```



```
test_lhy=> SELECT * FROM s2.t1;
 c1 | c2
----+----
  1 |  2
(1 row)

test_lhy=> UPDATE s2.t1 SET c2 = 3 WHERE c1 = 1;
UPDATE 1
```

----结束

7.2 实现数据列的加解密

数据加密作为有效防止未授权访问和防护数据泄露的技术，在各种信息系统中广泛使用。作为信息系统的核心，GaussDB(DWS)数仓也提供数据加密功能，包括透明加密和使用SQL函数加密。本章节主要讨论SQL函数加密。

说明

GaussDB(DWS)目前不支持从Oracle、Teradata和MySQL加密后到DWS解密。Oracle、Teradata和MySQL与DWS加解密有区别，需要非加密数据迁移到DWS后在DWS侧进行加解密。

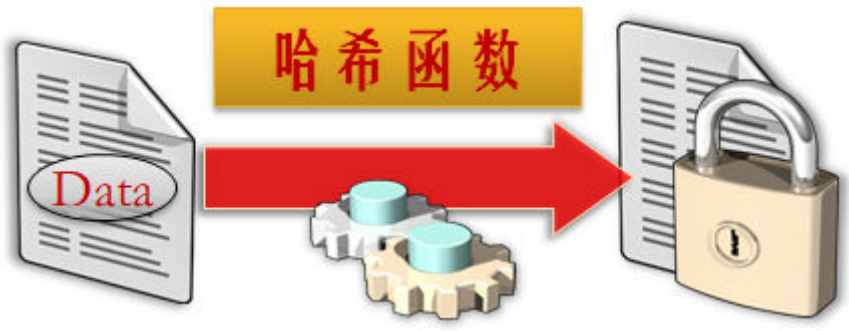
技术背景

- 哈希函数

  哈希函数又称为摘要算法，对于数据data，Hash函数会生成固定长度的数据，即Hash(data)=result。这个过程是不可逆的，即Hash函数不存在反函数，无法由result得到data。在不应保存明文场景（比如口令password属于敏感信息），系统管理员用户也不应该知道用户的明文口令，就应该使用哈希算法存储口令的单向哈希值。

  实际使用中会加入盐值和迭代次数，避免相同口令生成相同的哈希值，以防止彩虹表攻击。

  图 7-1 哈希函数

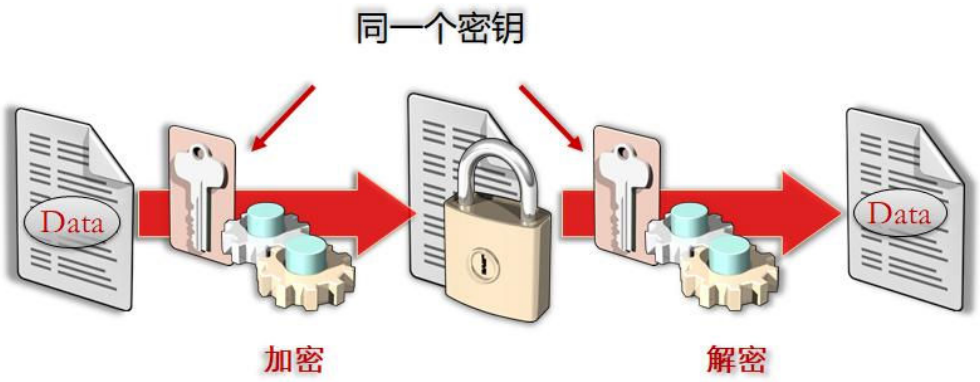
- 对称密码算法

  对称密码算法使用相同的密钥来加密和解密数据。对称密码算法分为分组密码算法和流密码算法。

  分组密码算法将明文分成固定长度的分组，用密钥对每个分组加密。由于分组长度固定，当明文长度不是分组长度的整数倍时，会对明文做填充处理。由于填充的存在，分组密码算法得到的密文长度会大于明文长度。

流加密算法是指加密和解密双方使用相同伪随机加密数据流作为密钥，明文数据依次与密钥数据流顺次对应加密，得到密文数据流。实践中数据通常是一个位（bit）并用异或（xor）操作加密。流密码算法不需要填充，得到的密文长度等于明文长度。

图 7-2 对称密码算法



技术实现

GaussDB(DWS)主要提供了哈希函数和对称密码算法来实现对数据列的加解密。哈希函数支持sha256，sha384，sha512和国密sm3。对称密码算法支持aes128，aes192，aes256和国密sm4。

- 哈希函数
  - md5(string)

    将string使用MD5加密，并以16进制数作为返回值。MD5的安全性较低，不建议使用。
  - gs_hash(hashstr, hashmethod)

    以hashmethod算法对hashstr字符串进行信息摘要，返回信息摘要字符串。支持的hashmethod：sha256, sha384, sha512, sm3。
- 对称密码算法
  - gs_encrypt(encryptstr, keystr, cryptotype, cryptomode, hashmethod)

    采用cryptotype和cryptomode组成的加密算法以及hashmethod指定的HMAC算法，以keystr为密钥对encryptstr字符串进行加密，返回加密后的字符串。
  - gs_decrypt(decryptstr, keystr，cryptotype, cryptomode, hashmethod)

    采用cryptotype和cryptomode组成的加密算法以及hashmethod指定的HMAC算法，以keystr为密钥对decryptstr字符串进行解密，返回解密后的字符串。解密使用的keystr必须保证与加密时使用的keystr一致才能正常解密。
  - gs_encrypt_aes128(encryptstr,keystr)

    以keystr为密钥对encryptstr字符串进行加密，返回加密后的字符串。keystr的长度范围为1~16字节。
  - gs_decrypt_aes128(decryptstr,keystr)

    以keystr为密钥对decryptstr字符串进行解密，返回解密后的字符串。解密使用的keystr必须保证与加密时使用的keystr一致才能正常解密。keystr不得为空。

有关函数的更多内容，请参见**使用函数加解密**。

应用示例

步骤1 连接数据库。

具体步骤参见**使用gsql命令行客户端连接集群**。

步骤2 创建表student，有id，name和score三个属性。使用哈希函数加密保存name，使用对称密码算法保存score。

```
CREATE TABLE student (id int, name text, score text, subject text);
INSERT INTO student VALUES (1, gs_hash('alice', 'sha256'), gs_encrypt('95', '12345', 'aes128', 'cbc',
'sha256'),gs_encrypt_aes128('math', '1234'));
INSERT INTO student VALUES (2, gs_hash('bob', 'sha256'), gs_encrypt('92', '12345', 'aes128', 'cbc',
'sha256'),gs_encrypt_aes128('english', '1234'));
INSERT INTO student VALUES (3, gs_hash('peter', 'sha256'), gs_encrypt('98', '12345', 'aes128', 'cbc',
'sha256'),gs_encrypt_aes128('science', '1234'));
```

步骤3 不使用密钥查询表student，通过查询结果可知：没有密钥的用户即使拥有了select权限也无法看到name和score这两列加密数据。

```
select * from student;
 id |                                name                                |
score                                                |
                          subject
----+--------------------------------------------------------------------
+-----------------------------------------------------------------------------------------------------------------------
+-----------
-------------------------------------------------------------------------------
  1 | 2bd806c97f0e00af1a1fc3328fa763a9269723c8db8fac4f93af71db186d6e90 | AAAAAAAAAABAuUC3VQ
+MvPCDAaTUySl1e2gGLr4/ATdCUjTEvova3cb/Ba3ZKqIn1yNVGEFBvJnTq/3sLF4//
Gm8qG7AyfNbbqdW3aYErLVpbE/QWFX9Ig== | aFEWQR2gkj
iu6sfsAad+dHzfFDHePZ6xd44zyekh+qVFlh9FODZ0DoaFAJXctwUsiqaiitTxW8cCSEaNjS/E7Ke1ruY=
  2 | 81b637d8fcd2c6da6359e6963113a1170de795e4b725b84d1e0b4cfd9ec58ce9 | AAAAAAAAAABAuUC3VQ
+MvPCDAaTUySl1taXxAoDqE793hgyCJvC0ESdAX5Mtgdq2LXI1f5ZxraQ73WIJVtIBX8oe3gTDxoXGlHbHht4kzM
4U8dOwr5rjgg== | aFEWQR2gkj
iu6sfsAad+dM8tPTDo/Pds6ZmqdmjGiKxf39+Wzx5NoQ6c8FrzihnRzgc0fycWSu5YGWNOKYWhRsE84Ac=
  3 | 026ad9b14a7453b7488daa0c6acbc258b1506f52c441c7c465474c1a564394ff |
AAAAAAAAAACnyusORPeApqMUgh56ucQu3uso/
Llw5MbPFMkOXuspEzhhnc9vErwOFe6cuGtx8muEyHCX7V5yXs+8FxhNh3n5L3419LDWJJLY2O4merHpSg== |
zomphRfHV4
H32hTtgkio1PyrobVO8N+hN7kAKwtygKP2E7Aaf1vsjmtLHcL88jyeJNe1lxe0fAvodzPJAxAuV3UJN4M=
(3 rows)
```

步骤4 使用密钥查询表student，通过查询结果可知：拥有密钥的用户通过使用gs_encrypt对应的解密函数gs_decrypt解密后，可以查看加密数据。

```
select id, gs_decrypt(score, '12345', 'aes128', 'cbc', 'sha256'),gs_decrypt_aes128(subject, '1234') from student;
 id | gs_decrypt | gs_decrypt_aes128
----+------------+-------------------
  1 | 95         | math
  2 | 92         | english
  3 | 98         | science
(3 rows)
```

----结束

7.3 通过视图管控数据权限

本章节介绍如何通过视图实现给不同的用户授予查询同一表中不同数据的权限，提供数据的权限管理和安全性。

场景

dbadmin用户连接集群后，创建示例表customer：

```
CREATE TABLE customer (id bigserial NOT NULL, province_id bigint NOT NULL, user_info varchar, primary key (id)) DISTRIBUTE BY HASH(id);
```

向示例表customer插入测试数据：

```
INSERT INTO customer(province_id,user_info) VALUES (1,'Alice'),(1,'Jack'),(2,'Jack'),(3,'Matu');
INSERT 0 4
```

查询示例表customer：

```
SELECT * FROM customer;
 id | province_id | user_info
----+-------------+-----------
  3 |           2 | Jack
  1 |           1 | Alice
  2 |           1 | Jack
  4 |           3 | Matu
(4 rows)
```

需求：要求用户u1仅能查看省份1（即province_id=1）的数据，而u2仅能查看省份2（即province_id=2）的数据。

实现方式

通过创建视图实现上述场景中的需求，具体操作步骤如下：

步骤1 dbadmin用户连接集群后，在dbadmin模式下为省份1和省份2分别创建视图v1和视图v2。

使用CREATE VIEW语句创建查询省份1数据的视图v1：

```
CREATE VIEW v1 AS
   SELECT * FROM customer WHERE province_id=1;
```

使用CREATE VIEW语句创建查询省份2数据的视图v2：

```
CREATE VIEW v2 AS
   SELECT * FROM customer WHERE province_id=2;
```

步骤2 创建用户u1和u2。

```
CREATE USER u1 PASSWORD '*********';
CREATE USER u2 PASSWORD '*********';
```

步骤3 使用GRANT语句将对应的数据查询权限授予目标用户。

授予u1和u2对应视图schema的权限。

```
GRANT USAGE ON schema dbadmin TO u1,u2;
```

授予u1通过v1视图查询省份1数据的权限：

```
GRANT SELECT ON v1 TO u1;
```

授予u2通过v2视图查询省份2数据的权限：

```
GRANT SELECT ON v2 TO u2;
```

----结束

查询结果验证

- 切换到u1账号连接集群。


```
SET ROLE u1 PASSWORD '*********';
```

查询v1视图，u1仅能查询到视图v1数据。

```
SELECT * FROM dbadmin.v1;
 id | province_id | user_info
----+-------------+-----------
  1 |           1 | Alice
  2 |           1 | Jack
(2 rows)
```

若u1试图查询视图v2中的数据，则会返回如下报错：

```
SELECT * FROM dbadmin.v2;
ERROR:  SELECT permission denied to user "u1" for relation "dbadmin.v2"
```

结果显示用户u1仅能查看省份1（即province_id=1）的数据。

- 使用u2账号连接集群。

```
SET ROLE u2 PASSWORD '*********';
```

查询v2视图，u2仅能查询到视图v2数据。

```
SELECT * FROM dbadmin.v2;
 id | province_id | user_info
----+-------------+-----------
  3 |           2 | Jack
(1 row)
```

若u2试图查询视图v1中的数据，则会返回如下报错：

```
SELECT * FROM dbadmin.v1;
ERROR:  SELECT permission denied to user "u2" for relation "dbadmin.v1"
```

结果显示用户u2仅能查看省份2（即province_id=2）的数据。

7.4 只读用户配置权限

背景信息

如果您需要对华为云上的GaussDB(DWS)资源，为企业中的员工设置不同的访问权限，以达到不同员工之间的权限隔离，您可以使用统一身份认证服务（Identity and Access Management，简称IAM）进行精细的权限管理。该服务提供用户身份认证、权限分配、访问控制等功能，可以帮助您安全的控制云资源的访问。通过IAM，您可以在云账号中给员工创建IAM用户，并授权控制他们对云资源的访问范围。

- **场景一**：您的员工中有负责软件开发的人员，您希望他们拥有GaussDB(DWS)的使用权限，但是不希望他们拥有删除集群等高危操作的权限，那么您可以使用IAM为开发人员创建用户，通过授予仅能使用GaussDB(DWS)，但是不允许删除集群的权限，控制他们对GaussDB(DWS)资源的使用范围。
- **场景二**：您希望您的员工只有GaussDB(DWS)的资源使用权限，不希望拥有其他云资源的权限，以防止资源滥用。例如只开通GaussDB(DWS)的操作权限，不能使用其他云服务。

通过IAM权限控制，有效达到云资源访问控制，避免云资源误操作。本文将指导如何配置只读权限的IAM用户。

教程一：IAM 项目视图下的只读操作

步骤1 **创建用户组并授权**。

使用华为云账号登录**IAM控制台**，创建用户组，并授予数据仓库服务的只读权限“DWS ReadOnlyAccess”。

步骤2 **创建用户并加入用户组**。

在IAM控制台创建用户，并将其加入步骤**步骤1**中创建的用户组。

步骤3 **用户登录**并验证权限。

使用新创建的用户登录控制台，切换至授权区域，验证权限：

- 在“服务列表”中选择数据仓库服务，进入DWS主界面，单击右上角“创建数据仓库集群”，尝试创建数据仓库集群，如果无法创建（假设当前权限仅包含DWS ReadOnlyAccess），表示“DWS ReadOnlyAccess”已生效。
- 在“服务列表”中选择除数据仓库服务之外（假设当前策略仅包含DWS ReadOnlyAccess）的任一服务，若提示权限不足，表示“DWS ReadOnlyAccess”已生效。

----结束

教程二：企业项目下的只读操作

步骤1 **创建用户组并授权**。

使用华为云账号登录**IAM控制台**，创建用户组，并授予数据仓库服务的只读权限“DWS ReadOnlyAccess”。

说明

- 企业项目视图下，跟资源无关的只读操作细粒度权限依旧会提示无权限访问。如事件、告警等相关接口的细粒度。

步骤2 配置IAM项目视图下相关的事件与告警等只读权限。

1. 创建如下自定义策略readonly_event_alarm：

```
{
    "Version": "1.1",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "dws:alarm*:list*",
                "dws:cluster*:list*",
                "dws:dms*:get*",
                "dws:event*:list*"
            ]
        }
    ]
}
```

2. 登录**IAM控制台**，**创建用户组**并授权刚创建的自定义策略：

步骤3 **创建用户并加入用户组**。

在IAM控制台创建用户，并将其加入步骤**步骤1**中创建的用户组。

步骤4 用户登录并验证权限。

使用新创建的用户登录控制台，切换至授权区域，验证权限：

- 在“服务列表”中选择数据仓库服务，进入DWS主界面，单击右上角“创建数据仓库集群”，尝试创建数据仓库集群，如果无法创建（假设当前权限仅包含DWS ReadOnlyAccess），表示“DWS ReadOnlyAccess”已生效。
- 在“服务列表”中选择除数据仓库服务之外（假设当前策略仅包含DWS ReadOnlyAccess）的任一服务，若提示权限不足，表示“DWS ReadOnlyAccess”已生效。

----结束

