Data Warehouse Service
Best Practices

Issue 03

Date 2020-12-15

HUAWEI

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


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

Notice

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

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

Contents

1 Excellent Practices for Table Design

1.1 Learning the Tutorial: Tuning Table Design

During database design, some key factors about table design will greatly affect the subsequent query performance of the database. Table design affects data storage as well. Scientific table design reduces I/O operations and minimizes memory usage, improving the query performance.

Overview describes how to select the sequence keys, distribution modes, and compression encoding for tables, and how to compare system performance before and after tuning.

The following sections summarize the key points associated with table design and provide suggestions for excellent practices.

1.2 Selecting a Storage Model

During database design, some key factors about table design will greatly affect the subsequent query performance of the database. Table design affects data storage as well. Scientific table design reduces I/O operations and minimizes memory usage, improving the query performance.

Selecting a model for table storage is the first step of table definition. Select a proper storage model for your service based on the following table.

Storage Model	Application Scenario
Row storage	Point query (simple index–based query that returns only a few records). Query involving many **INSERT**, **UPDATE**, and **DELETE** operations.
Column storage	Statistics analysis query, in which operations, such as group and join, are performed many times.

1.3 Selecting a Distribution Mode

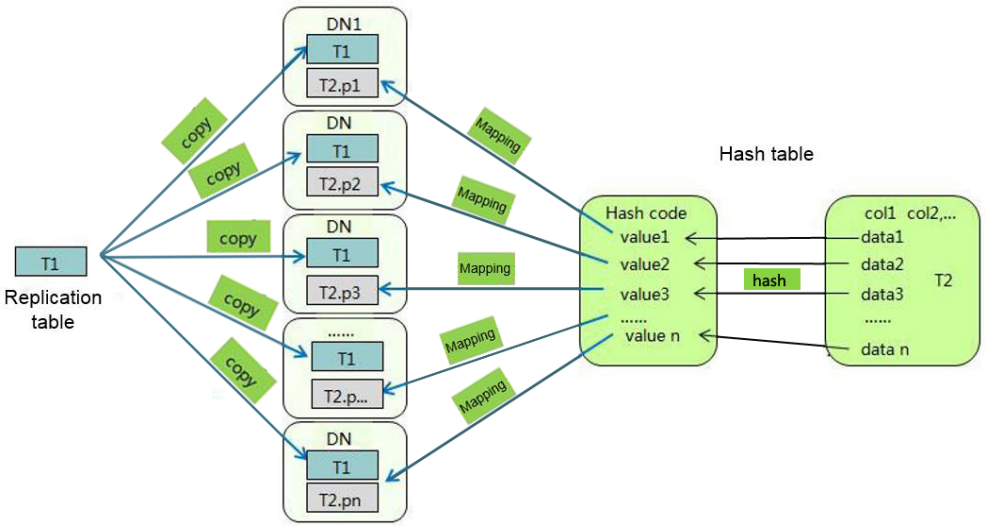
Replication is to copy full data in a table to every DN in a cluster. This is suitable for tables having small record sets. Full data in a table stored on each DN avoids data redistribution during the join operation. This reduces network costs and plan segment (each having a thread), but generates much redundant data. Generally, replication is only used for small dimension tables.

In a hash table, hash values are generated for one or more columns. You can obtain the storage location of a tuple based on the mapping between DNs and the hash values. In a hash table, I/O resources on each node can be used during I/O read/write, which greatly improves the read/write speed of a table. Generally, a table containing a large amount data is defined as a hash table.

Policy	Description	Application Scenario
Hash	Table data is distributed on all DNs in the cluster in hash mode.	Fact tables containing a large amount of data
Replication	Full data in a table is stored on each DN in the cluster.	Small tables and dimension tables.

As shown in **Figure 1-1**, T1 is a replication table and T2 is a hash table.

Figure 1-1 Replication table and hash table



1.4 Selecting a Distribution Column

The distribution column in a hash table must meet the following requirements, which are ranked by priority in descending order:

1. **The values of the distribution column should be discrete so that data can be evenly distributed on each DN.** For example, you are advised to select

the primary key of a table as the distribution column, and the ID card number as the distribution column in a personnel information table.

2. **Do not select the column where a constant filter exists.** For example, if a constant constraint (for example, zqdh= '000001') exists on the **zqdh** column in some queries on the **dwcjk** table, you are not advised to use **zqdh** as the distribution column.
3. **Select the join condition as the distribution column**, so that join tasks can be pushed down to DNs to execute, reducing the amount of data transferred between the DNs.

   For a hash table, an improper distribution key may cause data skew or poor I/O performance on certain DNs. Therefore, you need to check the table to ensure that data is evenly distributed on each DN. You can run the following SQL statements to check for data skew:

   ```
   select
   xc_node_id, count(1)
   from tablename
   group by xc_node_id
   order by xc_node_id desc;
   ```

   xc_node_id corresponds to a DN. Generally, **over 5% difference between the amount of data on different DNs is regarded as data skew. If the difference is over 10%, choose another distribution column.**
4. You are not advised to add a column as a distribution column, especially add a new column and use the SEQUENCE value to fill the column. This is because SEQUENCE may cause performance bottlenecks and unnecessary maintenance costs.

Multiple distribution columns can be selected in GaussDB(DWS) to evenly distribute data.

1.5 Using Partitioned Tables

Partitioning refers to splitting what is logically one large table into smaller physical pieces based on specific schemes. The table based on the logic is called a partitioned table, and a physical piece is called a partition. Data is stored on these smaller physical pieces, namely, partitions, instead of the larger logical partitioned table. A partitioned table has the following advantages over an ordinary table:

1. High query performance: The system queries only the concerned partitions rather than the whole table, so the query efficiency is improved.
2. High availability: If a partition in a partitioned table is faulty, data in the other partitions is still available.
3. Easy maintenance: To fix a partitioned table having a faulty partition, you simply need to fix the partition.

GaussDB(DWS) supports range partitioned tables.

Range-partitioned table: Data within a specific range is mapped onto each partition. The range is determined by the partition key specified during the partitioned table creation. The partition key is usually a date. For example, sales data is partitioned by month.

1.6 Using Partial Clustering

Partial Cluster Key is the column-store-based technology. It can minimize or maximize sparse indexes to quickly filter base tables. Partial cluster key can specify multiple columns, but you are advised to specify no more than two columns. Use the following principles to specify columns:

1. The selected columns must be restricted by simple expressions in base tables. Such constraints are usually represented by Col, Op, and Const. Col specifies the column name, Op specifies operators, (including =, >, >=, <=, and <) Const specifies constants.
2. Select columns that are frequently selected (to filter much more undesired data) in simple expressions.
3. List the less frequently selected columns following Col listed on the top.
4. List the columns of the enumerated type at the top.

1.7 Selecting a Data Type

Use the following principles to obtain efficient data types:

1. **Using the data type that can be efficiently executed**

   Generally, calculation of integers (including common comparison calculations, such as =, >, <, ≥, ≤, and ≠ and group by) is more efficient than that of strings and floating point numbers. For example, if you need to filter data in a column containing numeric data for a column-store table where point query is performed, the execution takes over 10s. However, the execution time is reduced to 1.8s when you change the data type from NUMERIC to INT.

2. **Using the data type of short length column**

   Using the data type with a shorter length reduces both the data file size and the memory used for computing, improving the I/O and computing performance. For example, use SMALLINT instead of INT, and INT instead of BIGINT.

3. **Using the same data type for associated columns**

   Use the same data type for associated columns. If columns having different data types are associated, the database must dynamically convert the different data types into the same ones for comparison. The conversion results in performance overheads.

2 Excellent Practices for Data Import

Importing Data from OBS in Parallel

- Splitting a data file into multiple files

  Importing a huge amount of data takes a long period of time and consumes many computing resources.

  To improve the performance of importing data from OBS, split a data file into multiple files as evenly as possible before importing it to OBS. The preferred number of split files is an integer multiple of the DN quantity.

- Verifying data files before and after an import

  When importing data from OBS, first import your files to your OBS bucket, and then verify that the bucket contains all the correct files, and only those files.

  After the import is complete, run the **SELECT** statement to verify that the required files have been imported.

- Ensuring no Chinese characters are contained in paths used for importing data to or exporting data from OBS.

Using GDS to Import Data

- Data skew causes the query performance to deteriorate. Data skew causes the query performance to deteriorate. Before importing all the data from a table containing over 10 million records, you are advised to import some of the data and check whether there is data skew and whether the distribution keys need to be changed. Troubleshoot the data skew if any. It is costly to address data skew and change the distribution keys after a large amount of data has been imported. For details, see **Checking for Data Skew**.
- To speed up the import, you are advised to split files and use multiple GDSs to import data in parallel. An import task can be split into multiple concurrent import tasks. If multiple import tasks use the same GDS, you can specify the **-t** parameter to enable GDS multi-thread concurrent import. To prevent physical I/O and network bottleneck, you are advised to mount GDSs to different physical disks and NICs.
- If the GDS I/O and NICs do not reach their physical bottlenecks, you can enable SMP on GaussDB(DWS) for acceleration. SMP will multiply the pressure on GDSs. Note that SMP adaptation is implemented based on the

GaussDB(DWS) CPU pressure rather than the GDS pressure. For details about SMP, see **Recommended Suggestions for SMP**.

- For the proper communication between GDSs and GaussDB(DWS), you are advised to use 10GE networks. 1GE networks cannot bear the high-speed data transmission, and as a result cannot ensure proper communication between GDSs and GaussDB(DWS). To maximize the import rate of a single file, ensure that a 10GE network is used and the data disk group I/O rate is greater than the upper limit of the GDS single-core processing capability (about 400 MB/s).
- Similar to the single-table import, ensure that the I/O rate is greater than the maximum network throughput in the concurrent import.
- It is recommended that the ratio of GDS quantity to DN quantity be in the range of 1:3 to 1:6.
- To improve the efficiency of importing data in batches to column-store partitioned tables, the data is buffered before being written into a disk. You can specify the number of buffers and the buffer size by setting **partition_mem_batch** and **partition_max_cache_size**, respectively. The smaller the values, the slower the batch import to column-store partitioned tables. The larger the values, the higher the memory consumption.

Using INSERT to Insert Multiple Rows

If the **COPY** statement cannot be used and you require SQL inserts, use a multi-row insert whenever possible. Data compression is inefficient when you add data of only one row or a few rows at a time.

Multi-row inserts improve performance by batching up a series of inserts. The following example inserts three rows into a three-column table using a single **INSERT** statement. This is still a small insert, shown simply to illustrate the syntax of a multi-row insert. For details about how to create a table, see **Creating a Table**.

To insert multiple rows of data to the table **customer_t1**, run the following statement:

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

For more details and examples, see **INSERT**.

Using the COPY Statement to Import Data

The **COPY** statement imports data from local and remote databases in parallel. **COPY** imports large amounts of data more efficiently than using **INSERT** statements.

For details about how to use the COPY statement, see **Data Import Using COPY FROM STDIN**.

Using a gsql Meta-Command to Import Data

The **\copy** command can be used to import data after you log in to a database through any **psql** client. Unlike the **COPY** statement, the **\copy** command reads from or writes into a file.

Data read or written using the **\copy** command is transferred through the connection between the server and the client and may not be efficient. The **COPY** statement is recommended when the amount of data is large.

For details about how to use the **\copy** command, see **Using a gsql Meta-Command to Import Data**.

NOTE

\copy only applies to small-batch data import with uniform formats but poor error tolerance capability. GDS or **COPY** is preferred for data import.

3 Excellent Practices for SQL Queries

Based on a large number of SQL execution mechanisms and practices, we can optimize SQL statements following certain rules to more quickly execute SQL statements and obtain correct results.

- Replacing **UNION** with **UNION ALL**

  UNION eliminates duplicate rows while merging two result sets but **UNION ALL** merges the two result sets without deduplication. Therefore, replace **UNION** with **UNION ALL** if you are sure that the two result sets do not contain duplicate rows based on the service logic.

- **Adding NOT NULL to the join column**

  If there are many **NULL** values in the **JOIN** columns, you can add the filter criterion **IS NOT NULL** to filter data in advance to improve the **JOIN** efficiency.

- Converting **NOT IN** to **NOT EXISTS**

  nestloop anti join must be used to implement **NOT IN**, and **hash anti join** is required for **NOT EXISTS**. If no NULL value exists in the **JOIN** column, **NOT IN** is equivalent to **NOT EXISTS**. Therefore, if you are sure that no **NULL** value exists, you can convert **NOT IN** to **NOT EXISTS** to generate **hash join** and to improve the query performance.

  As shown in the following figure, the **t2.d2** column does not contain null values (it is set to **NOT NULL**) and **NOT EXISTS** is used for the query.

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

  The generated execution plan is as follows:

Figure 3-1 NOT EXISTS execution plan

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

 Predicate Information (identified by plan id)
-----------------------------------------------
   2 --Hash Anti Join (3, 4)
         Hash Cond: (t1.c1 = t2.d2)
(2 rows)
```

- Use **hashagg**.

  If a plan involving groupAgg and SORT operations generated by the **GROUP BY** statement is poor in performance, you can set **work_mem** to a larger value to generate a **hashagg** plan, which does not require sorting and improves the performance.

- Replace functions with **CASE** statements

  The GaussDB(DWS) performance greatly deteriorates if a large number of functions are called. In this case, you can modify the pushdown functions to **CASE** statements.

- **Do not use functions or expressions for indexes.**

  Using functions or expressions for indexes stops indexing. Instead, it enables scanning on the full table.

- Do not use **!=** or **<>** operators, **NULL**, **OR**, or implicit parameter conversion in **WHERE** clauses.

- **Split complex SQL statements.**

  You can split an SQL statement into several ones and save the execution result to a temporary table if the SQL statement is too complex to be tuned using the solutions above, including but not limited to the following scenarios:

  – The same subquery is involved in multiple SQL statements of a task and the subquery contains large amounts of data.

  – Incorrect **Plan cost** causes a small hash bucket of subquery. For example, the actual number of rows is 10 million, but only 1000 rows are in hash bucket.

  – Functions such as **substr** and **to_number** cause incorrect measures for subqueries containing large amounts of data.

  – **BROADCAST** subqueries are performed on large tables in multi-DN environment.

For details about SQL optimization, see **Typical SQL Optimization Methods**.

4 Excellent Practices for Data Skew Queries

4.1 Real-Time Detection of Storage Skew During Data Import

During the import, the system collects statistics on the number of rows imported on each DN. After the import is complete, the system calculates the skew ratio. If the skew ratio exceeds the specified threshold, an alarm is generated immediately. The skew ratio is calculated as follows: Skew ratio = (Maximum number of rows imported on a DN – Minimum number of rows imported on a DN)/Number of imported rows. Currently, data can be imported only by running **INSERT** or **COPY**.

NOTE

enable_stream_operator must be set to **on** so that DNs can return the number of imported rows at a time when a plan is delivered to them. Then, the skew ratio is calculated on the CN based on the returned values.

Usage

1. Set parameters **table_skewness_warning_threshold** (threshold for triggering a table skew alarm) and **table_skewness_warning_rows** (minimum number of rows for triggering a table skew alarm).
   - The value of **table_skewness_warning_threshold** ranges from **0** to **1**. The default value is **1**, indicating that the alarm is disabled. Other values indicate that the alarm is enabled.
   - The value of **table_skewness_warning_rows** ranges from **0** to **2147483647**. The default value is **100,000**. The alarm is triggered only when the following condition is met: Total number of imported rows > Value of **table_skewness_warning_rows** x Number of DNs involving in the import.

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


2. Import data by running the **INSERT** or **COPY** statement.
3. Detect and handle alarms. The alarm information includes the table name, minimum number of rows, maximum number of rows, total number of rows, average number of rows, skew rate, and prompt information about data distribution or parameter modification.

```
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
```

4.2 Quickly Locating the Tables That Cause Data Skew

Currently, the **table_distribution(schemaname text, tablename text)** and **table_distribution()** functions as well as the **PGXC_GET_TABLE_SKEWNESS** view are provided to query for data skew. You can choose any of them as needed.

Scenario 1: Data Skew Caused by a Full Disk

First, use the **pg_stat_get_last_data_changed_time(oid)** function to query for the tables whose data is changed recently. The last change time of a table is recorded only on the CN where **INSERT**, **UPDATE**, and **DELETE** operations are performed. Therefore, you need to query for tables that are changed within the last day (the period can be changed in the function).

```
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;
```

Then, execute the **table_distribution(schemaname text, tablename text)** function to query for the storage space occupied by the tables on each DN.

```
SELECT table_distribution(schemaname,relname) FROM get_last_changed_table();
```

Scenario 2: Routine Data Skew Inspection

- If the number of tables in the database is less than 10,000, use the **PGXC_GET_TABLE_SKEWNESS** view to query for data skew of all tables in the database.

  ```
  SELECT * FROM pgxc_get_table_skewness ORDER BY totalsize DESC;
  ```

- If the number of tables in the database is no less than 10,000, you are advised to use the **table_distribution()** function instead of the

PGXC_GET_TABLE_SKEWNESS view because the view takes a longer time (hours) due to the query of the entire database for skew columns. When you use the **table_distribution()** function, you can define the output based on **PGXC_GET_TABLE_SKEWNESS**, optimizing the calculation and reducing the output columns. For example:

```
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;
```


5 Tutorial: Tuning Table Design

5.1 Overview

In this tutorial, you will learn how to optimize the design of your tables. You will start by creating tables without specifying their storage mode, distribution key, distribution mode, or compression mode. Load test data into these tables and test system performance. Then, follow excellent practices to create the tables again using new storage modes, distribution keys, distribution modes, and compression modes. Load the test data and test performance again. Compare the two test results to find out how table design affects the storage space, and the loading and query performance of the tables.

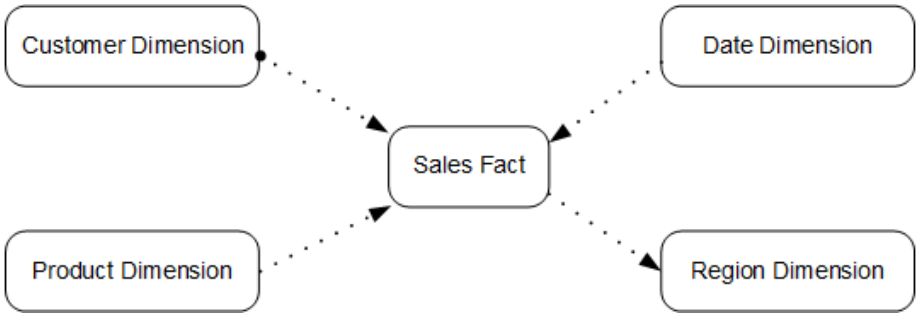
Estimated time: 60 minutes

5.2 Table Schemas

The most common types of data warehouse schemas are star and snowflake schemas. Consider service and performance requirements when you choose a schema for your tables.

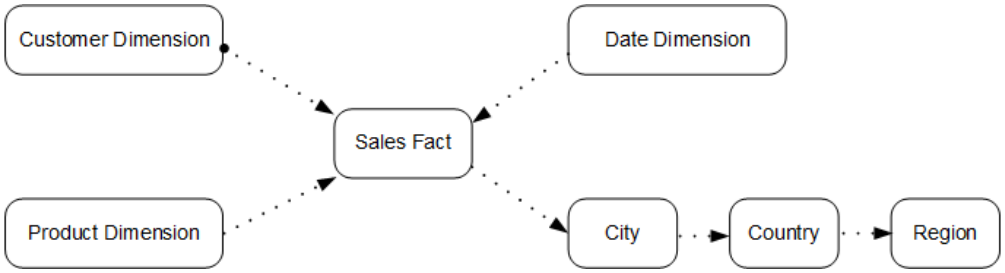
- In the star schema, a central fact table contains the core data for the database and several dimension tables provide descriptive attribute information for the fact table. The primary key of a dimension table associates a foreign key in a fact table, as shown in **Figure 5-1**.
  - All facts must have the same granularity.
  - Different dimensions are not associated.

Figure 5-1 Star schema



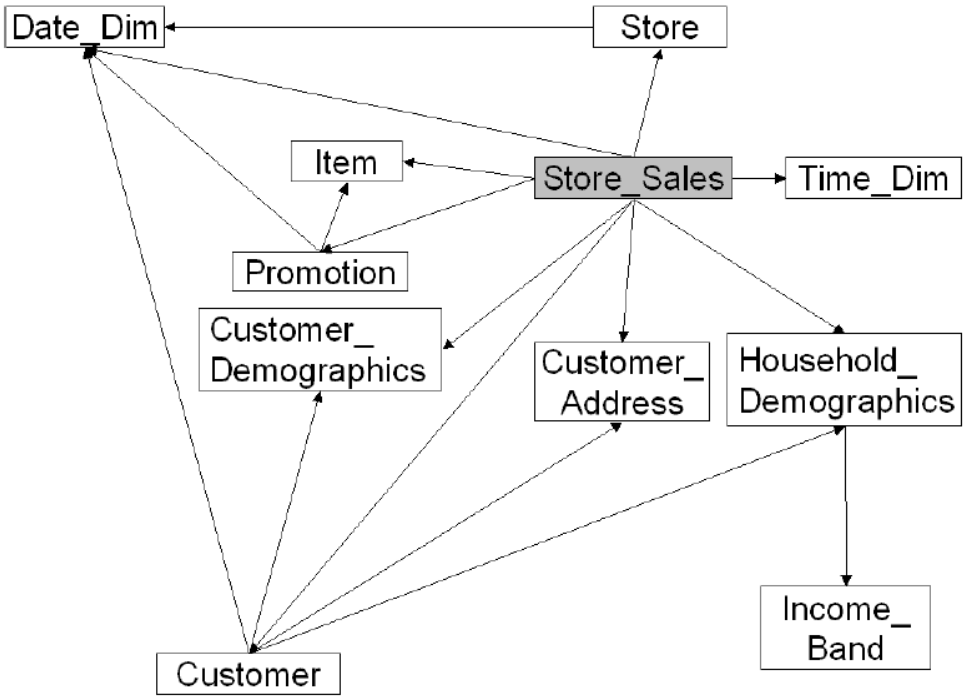
- The snowflake schema is developed based on the star schema. In this schema, each dimension can be associated with multiple dimensions and split into tables of different granularities based on the dimension level, as shown in **Figure 5-2**.
  - This schema reduces the amount of data in dimension tables and facilitates join queries.
  - This schema has more dimension tables that need to be maintained than the star schema does.

Figure 5-2 Snowflake schema



This tutorial verifies performance using the Store Sales (SS) model of TPC-DS. The model uses the snowflake schema. **Figure 5-3** illustrates its structure.

Figure 5-3 TPC-DS Store Sales ER-Diagram



For details about the **store_sales** fact table and dimension tables in the model, see the official document of TPC-DS at **http://www.tpc.org/tpc_documents_current_versions/current_specifications5.asp**.

5.3 Step 1: Creating an Initial Table and Loading Sample Data

Create a group of tables without specifying their storage modes, distribution keys, distribution modes, or compression modes. Load sample data into these tables.

Step 1 (Optional) Create a cluster.

If a cluster is available, skip this step. For details about how to create a cluster, see **Getting Started**. Then, follow the instructions provided in **Service Overview** to connect to the cluster using an SQL client and test the connection.

This tutorial uses an 8-node cluster as an example. You can also use a four-node cluster to perform the test.

Step 2 Create an SS test table **store_sales**.

NOTE

Before you create this table, delete existing SS tables first (if any) using the **DROP TABLE** command. For example, to delete the **store_sales** table, run the following command:

```
DROP TABLE store_sales;
```

Do not set the storage mode, distribution key, distribution mode, or compression mode when you create this table.

Run the **CREATE TABLE** command to create the 11 tables in **Figure 5-3**. This section only provides the syntax for creating the **store_sales** table. To create all tables, copy the syntax in **Creating an Initial Table**.

```
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)
) ;
```

Step 3 Load sample data into these tables.

An OBS bucket provides sample data used for this tutorial. The bucket can be read by all authenticated cloud users. Perform the following substeps to load the sample data:

1. Create a foreign table for each table.

   GaussDB(DWS) uses the Foreign Data Wrapper (FDW) provided by PostgreSQL to import data in parallel. To use FDW, you need to create FDW tables, also called foreign tables. This section only provides the syntax for creating the **obs_from_store_sales_001** foreign table corresponding to the **store_sales** table. To create all foreign tables, copy the syntax in **Creating a Foreign Table**.

   - The columns of the foreign tables must be the same as that of the corresponding ordinary table. In this example, **store_sales** and **obs_from_store_sales_ 001** should have the same columns.
   - The foreign table syntax obtains the sample data used for this tutorial from the OBS bucket. To load other sample data, modify **SERVER gsmpp_server OPTIONS** as needed. For details, see **About Parallel Data Import from OBS**.

   ```
   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.cn-north-1.myhuaweicloud.comstore_sales/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. Set **ACCESS_KEY** and **SECRET_ACCESS_KEY** parameters as needed in the foreign table creation statement, and run this statement in a client tool to create a foreign table.

   For the values of **ACCESS_KEY** and **SECRET_ACCESS_KEY**, see **Creating Access Keys (AK and SK)** of this document.

3. Import data.

   Create the **insert.sql** script containing the following statements and execute it:

   ```
   \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
   ```

   Information similar to the following is displayed:

   ```
   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. Calculate the total time spent in creating the 11 tables. The result will be recorded as the loading time in the benchmark table in **Step 1** in the next section.
5. Run the following command to verify that each table is loaded correctly and records lines into the table:

```
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;
```

The number of rows in each SS table is as follows:

Table name	Number of Rows
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

Step 4 Run the **ANALYZE** command to update statistics.

```
ANALYZE;
```

If **ANALYZE** is returned, the execution is successful.

```
ANALYZE
```

The **ANALYZE** statement collects statistics about table content in databases, which will be stored in the **PG_STATISTIC** system catalog. Then, the query optimizer uses the statistics to work out the most efficient execution plan.

After executing batch insertions and deletions, you are advised to run the **ANALYZE** statement on the table or the entire library to update statistics.

----End

5.4 Step 2: Testing System Performance of the Initial Table and Establishing a Baseline

Before and after tuning table structures, test and record the following information to compare differences in system performance:

- Load time
- Storage space occupied by tables
- Query performance

The examples in this tutorial are based on a dws.d2.xlarge cluster consisting of eight nodes. Even if you use the same cluster configuration, the results will be different. System performance is affected by many factors. No two systems will perform in the same way.

Model	dws.d2.xlarge VM
CPU	4*CPU E5-2680 v2 @ 2.80GHZ
Memory	32 GB
Network	1 GB
Disk	1.63 TB
Number of Nodes	8

Record the results using the following benchmark table.

Benchmark	Before	After
Loading time (11 tables)	341584 ms	
Occupied storage space		
Store_Sales		

Benchmark	Before	After
Date_Dim		
Store		
Item		
Time_Dim		
Promotion		
Customer_Demographics		
Customer_Address		
Household_Demographics		
Customer		
Income_Band		
Total storage space		
Query execution time		
Query 1		
Query 2		
Query 3		
Total execution time		

Perform the following steps to test the system performance before tuning to establish a benchmark:

Step 1 Enter the cumulative load time for all the 11 tables in the benchmarks table in the **Before** column.

Step 2 Record the storage space usage of each table.

Determine how much disk space is used for each table using the **pg_size_pretty** function and record the results in base tables.

```
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);
```

The following information is displayed:

```
        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)
```

Step 3 Test query performance.

Run the following queries and record the time spent on each query. The execution durations of the same query can be different, depending on the OS cache during execution. You are advised to perform several rounds of tests and select a group with average values.

```
\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;
```

----End

After the preceding statistics are collected, the benchmark table is as follows:

Benchmark	Before	After
Loading time (11 tables)	341584 ms	
Occupied storage space		
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	
Total storage space	42 GB	
Query execution time		
Query 1	14552.05 ms	
Query 2	27952.36 ms	
Query 3	17721.15 ms	
Total execution time	60225.56 ms	

5.5 Step 3: Selecting Storage and Compression Modes

Selecting a Storage Mode

GaussDB(DWS) supports hybrid row-column storage. You can create row- or column-store tables as needed in your business scenarios.

Generally, if a table contains many columns (called a wide table) and its query involves only a few columns, column storage is recommended. If a table contains only a few columns and a query includes most of the fields, row storage is recommended.

Storage Mode	Scenario
Row storage	Point query returned with a few records. This is a simple index-based query. In this scenario, data is frequently added, modified, and deleted.
Column storage	Statistics analysis queries where tables are frequently grouped and joined. Ad hoc queries where query conditions are uncertain and no index can be determined for row-store tables.

Sample tables used in this tutorial are typical multi-column TPC-DS tables where many statistical analysis queries are performed. Therefore, the column storage mode is recommended.

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

Selecting a Compression Level

In scenarios where I/O is large (much data is read and written) and CPU is sufficient (little data is computed), select a high compression ratio. In scenarios where I/O is small and CPU is insufficient, select a low compression ratio. Based on this principle, you are advised to select different compression ratios and test and compare the results to select the optimal compression ratio as required. Specify a compressions ratio using the **COMPRESSION** parameter. The supported values are as follows:

- The valid value of column-store tables is **YES**, **NO**, **LOW**, **MIDDLE**, or **HIGH**, and the default value is **LOW**.
- The valid values of row-store tables are **YES** and **NO**, and the default is **NO**.

The service scenarios applicable to each compression level are described in the following table.

Compression Level	Application Scenario
LOW	The system CPU usage is high and the disk storage space is sufficient.
MIDDLE	The system CPU usage is moderate and the disk storage space is insufficient.
HIGH	The system CPU usage is low and the disk storage space is insufficient.

No compression ratio is specified in **Step 1: Creating an Initial Table and Loading Sample Data**, and the low compression ratio is selected by GaussDB(DWS) by default. Specify **COMPRESSION** to **MIDDLE**, and compare the result to that when **COMPRESSION** is set to **LOW**.

The following is an example of selecting a storage mode and the **MIDDLE** compression ratio for a table.

```
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);
```

5.6 Step 4: Selecting Distribution Modes

GaussDB(DWS) uses a massively parallel processing (MPP) system of the shared-nothing architecture. The MPP performs horizontal partitioning to store tuples in service data tables on all DNs using proper distribution policies. Currently, user tables can be distributed in Replication or Hash mode.

- Replication: stores a full table on each DN. Full data in a table stored on each DN avoids data redistribution during the join operation. This reduces network costs and plan segment (each having a thread), but generates much redundant data. Generally, replication is only used for small dimension tables.
- Hash: A distribute key must be specified for a user table. If a record is inserted, the system performs hash computing based on values in the distribute column and then stores data on the related DN. In a hash table, I/O resources on each node can be used during I/O read/write, which greatly improves the read/write speed of a table. Generally, a large table (containing over 1 million records) is defined as a hash table.

Based on table sizes provided in **Step 2: Testing System Performance of the Initial Table and Establishing a Baseline**, set the distribution mode as follows.

Table Name	Number of Rows	Distribution Mode
Store_Sales	287997024	Hash
Date_Dim	73049	Replication
Store	402	Replication

Table Name	Number of Rows	Distribution Mode
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

5.7 Step 5: Selecting a Distribution Key

If your table is distributed using hash, select a proper distribution key to prevent data skew or poor I/O performance on certain DNs.

You are advised to select a distribution key for each hash table based on the following rules:

1. **The values of the distribution column should be discrete so that data can be evenly distributed on each DN.** For example, select the primary key of a hash table as a distribution column or the ID card number as the distribution column in an employee information table.
2. **Do not select the column where a constant filter exists.** For example, if a constant constraint (for example, zqdh= '000001') exists on the **zqdh** column in some queries on the **dwcjk** table, you are not advised to use **zqdh** as the distribution column.
3. **Select the join conditions in a query as distribution keys.** In this way, Join tasks can be pushed down to DNs for execution, and the communication data between DNs is reduced.

Based on the excellent practice rules, select the primary key of each table as the hash table distribution key.

Table Name	Number of Records	Distribution Mode	Distribution Key
Store_Sales	287997024	Hash	ss_item_sk
Date_Dim	73049	Replication	-
Store	402	Replication	-
Item	204000	Replication	-

Table Name	Number of Records	Distribution Mode	Distribution Key
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	-

5.8 Step 6: Creating Another Table and Loading Data

After selecting a storage mode, compression level, distribution mode, and distribution column for each table, use these attributes to create tables and reload data. Compare the system performance before and after the table recreation.

Step 1 Delete the tables created before.

```
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;
```

Step 2 Create tables and specify storage and distribution modes for them.

Only the syntax for recreating the **store_sales** table is provided for simplicity. To recreate all the other tables, copy the syntax in **Creating a Secondary Table After Design Tuning**.

```
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);
```

Step 3 **Load sample data into these tables.**

Step 4 Record the loading time in the benchmark tables.

Benchmark	Before	After
Loading time (11 tables)	341584 ms	257241 ms
Occupied storage space		
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	
Total storage space	42 GB	
Query execution time		
Query 1	14552.05 ms	

Benchmark	Before	After
Query 2	27952.36 ms	
Query 3	17721.15 ms	
Total execution time	60225.56 ms	

Step 5 Run the **ANALYZE** command to update statistics.

```
ANALYZE;
```

If **ANALYZE** is returned, the execution is successful.

```
ANALYZE
```

Step 6 Check for data skew.

For a hash table, an improper distribution key may cause data skew or poor I/O performance on certain DNs. Therefore, you need to check the table to ensure that data is evenly distributed on each DN. You can run the following SQL statements to check for data skew:

```
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;
```

xc_node_id corresponds to a DN. Generally, **over 5% difference between the amount of data on different DNs is regarded as data skew. If the difference is over 10%, choose another distribution column.** Multiple distribution columns can be selected on GaussDB(DWS) to distribute data evenly.

----End

5.9 Step 7: Testing System Performance in the New Table

After recreating the test data set with the selected storage modes, compression levels, distribution modes, and distribution columns, you will retest the system performance.

Step 1 Record the storage space usage of each table.

Determine how much disk space is used for each table using the **pg_size_pretty** function and record the results in base tables.

```
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)
```

Step 2 Test the query performance and record the performance data in the benchmark table.

Execute the following queries again and record the time spent on each query.

```
\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;
```

The following benchmark table shows the validation results of the cluster used in this tutorial. Your results may vary based on a number of factors, but the relative results should be similar. The execution durations of queries having the same table structure can be different, depending on the OS cache during execution. You are advised to perform several rounds of tests and select a group with average values.

Benchmark	Before	After
Loading time (11 tables)	341584 ms	257241 ms
Occupied storage space		
Store_Sales	42 GB	14 GB
Date_Dim	11 MB	27 MB
Store	232 KB	4352 KB
Item	110 MB	259 MB
Time_Dim	11 MB	14 MB
Promotion	256 KB	3200 KB
Customer_Demographics	171 MB	11 MB
Customer_Address	170 MB	27 MB
Household_Demographics	504 KB	1280 KB
Customer	441 MB	111 MB
Income_Band	88 KB	896 KB
Total storage space	42 GB	15 GB
Query execution time		
Query 1	14552.05 ms	1783.353 ms
Query 2	27952.36 ms	14247.803 ms
Query 3	17721.15 ms	11441.659 ms
Total execution time	60225.56 ms	27472.815 ms

Step 3 If you have higher expectations for the performance after the table design, you can run the **EXPLAIN PERFORMANCE** command to view the execution plan for tuning.

For more details about execution plans and query tuning, see **SQL Execution Plan** and **Query Performance Tuning Overview**.

----End

5.10 Step 8: Evaluating the Results

Compare the loading time, storage space usage, and query execution time before and after the table tuning.

The following table shows the example results of the cluster used in this tutorial. Your results will be different, but should show similar improvement.

Benchmark	Before	After	Change	Percentage (%)
Loading time (11 tables)	341584 ms	257241 ms	-84343 ms	-24.7%
Occupied storage space			-	-
Store_Sales	42 GB	14 GB	-28 GB	-66.7%
Date_Dim	11 MB	27 MB	16 MB	145.5%
Store	232 KB	4352 KB	4120 KB	1775.9%
Item	110 MB	259 MB	149 MB	1354.5%
Time_Dim	11 MB	14 MB	13 MB	118.2%
Promotion	256 KB	3200 KB	2944 KB	1150%
Customer_Demographics	171 MB	11 MB	-160 MB	-93.6
Customer_Address	170 MB	27 MB	-143 MB	-84.1%
Household_Demographics	504 KB	1280 KB	704 KB	139.7%
Customer	441 MB	111 MB	-330 MB	-74.8%
Income_Band	88 KB	896 KB	808 KB	918.2%
Total storage space	42 GB	15 GB	-27 GB	-64.3%
Query execution time			-	-
Query 1	14552.05 ms	1783.353 ms	-12768.697 ms	-87.7%
Query 2	27952.36 ms	14247.803 ms	-13704.557 ms	-49.0%
Query 3	17721.15 ms	11441.659 ms	-6279.491 ms	-35.4%
Total execution time	60225.56 ms	27472.815 ms	-32752.745 ms	-54.4%

- The loading time was reduced by 24.7%.

  The distribution mode has obvious impact on loading data. The hash distribution mode improves the loading efficiency. The replication distribution mode reduces the loading efficiency. When the CPU and I/O are sufficient, the compression level has little impact on the loading efficiency. Typically, the efficiency of loading a column-store table is higher than that of a row-store table.

- The storage usage space was reduced by 64.3%.

  The compression level, column storage, and hash distribution can save the storage space. A replication table increases the storage usage, but reduces the network overhead. Using the replication mode for small tables is a positive way to use small space for performance.

- The query performance (speed) increased by 54.4%, indicating that the query time decreased by 54.4%.

  The query performance is improved by optimizing storage modes, distribution modes, and distribution columns. In a statistical analysis query on multi-column tables, column storage can improve query performance. In a hash table, I/O resources on each node can be used during I/O read/write, which improves the read/write speed of a table.

  Often, query performance can be improved further by rewriting queries and configuring workload management (WLM). For details, see **Query Performance Tuning Overview** and **Resource Load Management Overview**.

5.11 Step 9: Cleaning Up Resources

After completing this tutorial, delete the cluster by following the instructions provided in **Service Overview**.

If you want to keep the cluster, but delete the storage space used by the SS tables, run the following commands:

```
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;
```

5.12 Summary

In this tutorial, you learned how to optimize the design of your tables. Based on these steps, you can further use the excellent practices in **Tutorial: Tuning Table Design** to improve the distribution of tables to achieve the desired effects on data loading, storage, and query.

5.13 Appendix: Table Creation Syntax

5.13.1 Usage

This section provides SQL test statements used in this tutorial. You are advised to copy the SQL statements in each section and save them as an .sql file. For example, create a file named **create_table_fir.sql** file and paste the SQL statements in section **Creating an Initial Table** to the file. Executing the file on an

SQL client tool is efficient, and the total elapsed time of test cases is easy to calculate. Execute the **.sql** file using **gsql** as follows:

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

Replace the italic parts in the example with actual values in GaussDB(DWS). For example:

```
gsql -d postgres -h 10.10.0.1 -U dbadmin -p 8000 -W Bigdata@123 -f create_table_fir.sql
```

5.13.2 Creating an Initial Table

This section contains the table creation syntax used when you create a table for the first time in this tutorial. Tables are created without specifying their storage modes, distribution keys, distribution modes, or compression modes.

```
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
) ;
```

5.13.3 Creating a Secondary Table After Design Tuning

This section contains the syntax of creating a secondary table after the storage modes, compression levels, distribution modes, and distribution columns are selected in this tutorial.

```
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;
```

5.13.4 Creating a Foreign Table

This section contains the syntax of foreign tables for obtaining sample data used in this tutorial. The sample data is stored in an OBS bucket accessible to all authenticated cloud users. During runtime, you can replace **ACCESS_KEY** and **SECRET_ACCESS_KEY** with your own credentials in this example.

```
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.cn-north-1.myhuaweicloud.comstore_sales/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.cn-north-1.myhuaweicloud.comdate_dim/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.cn-north-1.myhuaweicloud.comstore/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.cn-north-1.myhuaweicloud.comitem/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.cn-north-1.myhuaweicloud.comtime_dim/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 (
LOCATIONobs.cn-north-1.myhuaweicloud.compromotion/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.cn-north-1.myhuaweicloud.comcustomer_demographics/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.cn-north-1.myhuaweicloud.comcustomer_address/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.cn-north-1.myhuaweicloud.comhousehold_demographics/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.cn-north-1.myhuaweicloud.comcustomer/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.cn-north-1.myhuaweicloud.comincome_band/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;
```


6 Tutorial: Importing Data from OBS to a Cluster

6.1 Overview

This tutorial demonstrates how to upload sample data to OBS and import OBS data into a table in GaussDB(DWS). This helps you quickly learn about how to import data from OBS to GaussDB(DWS).

In this tutorial, you will:

- Generate data files in CSV format.
- Create OBS buckets and upload data files to the buckets.
- Create a foreign table to import data from the OBS bucket to GaussDB(DWS) clusters.
- Start GaussDB(DWS), create a table, and import data from OBS to the table.
- Analyze import errors based on the information in the error table and correct these errors.

Estimated time: 30 minutes

6.2 Step 1: Uploading Data to OBS

Before importing data from OBS to a cluster, prepare source data files and upload these files to OBS. If the data files have been stored on OBS, perform only **Step 2** in **Uploading Data to OBS**.

Preparing Data Files

You can import data files in TEXT, CSV, ORC, or CARBONDATA format from to GaussDB(DWS). This tutorial uses data in CSV format as an example. The method is the same for TEXT and FIXED data except that the parameter settings of foreign tables are different. For details, see **About Parallel Data Import from OBS**.

To demonstrate how to import multiple files, this tutorial uses the following three CSV data files as an example. Generally, the source data files are exported from a database. In this tutorial, the CSV source data files are manually created.

- Data file **product_info0.csv**

  The file contains the following data:

  ```
  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.
  ```

- Data file **product_info1.csv**

  The file contains the following data:

  ```
  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.
  ```

- Data file **product_info2.csv**

  The file contains the following data:

  ```
  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."
  ```

CSV is short for Comma Separated Values. A .csv file is similar to a .txt or .doc file. It can also be considered a text file containing records, which are separated into columns by commas (,) or tabs. The column sequence in each record is the same. In Windows, .csv files can be opened in different applications, such as Notepad, Excel, and Notepad++.

The following describes how to generate a CSV file in Windows:

Step 1 Create a text file and open it in Notepad++. Copy the sample data into it. Then, check the total number of rows and check whether the data of rows is correctly separated.

Step 2 Choose **Format** > **Encode in UTF-8 without BOM**.

Step 3 Choose **File** > **Save as**.

Step 4 In the displayed dialog box, enter the file name and click **Save**.

To identify the file type, use the file name extension .csv when entering the file name.

----End

Uploading Data to OBS

Step 1 Store the three CSV source data files in the OBS bucket.

1. Log in to the OBS management console.

Click **Service List** and choose **Object Storage Service** to open the OBS management console. Alternatively, you can access **https://storage.huaweicloud.com** to log in to the OBS management console.

2. Create a bucket.

   For details about how to create a bucket, see **Creating a Bucket** in the *Object Storage Service Console Operation Guide*.

   For example, create two buckets named **mybucket** and **mybucket02**.

3. Create a folder.

   For details, see **Creating a Folder** in the *Object Storage Service Console Operation Guide*

   For example:

   - Create a folder named **input_data** in the **mybucket** OBS bucket.
   - Create a folder named **input_data** in the **mybucket02** OBS bucket.

4. Upload the files.

   For details, see **Uploading an Object** in the *Object Storage Service Console Operation Guide*.

   For example:

   - Upload the following data files to the **input_data** folder in the **mybucket** OBS bucket:

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

   - Upload the following data file to the **input_data** folder in the **mybucket02** OBS bucket:

     ```
     product_info2.csv
     ```

Step 2 Grant the OBS bucket read permission for the user who will import data.

When importing data from OBS to a cluster, the user must have the read permission for the OBS buckets where the source data files are located. You can configure the ACL for the OBS buckets to grant the read permission to a specific user.

For details, see **Configuring a Bucket ACL** in the *Object Storage Service Console Operation Guide*.

----End

6.3 Step 2: Creating a Foreign Table

Step 1 Connect to the GaussDB(DWS) database.

Step 2 Create a foreign table to store the three CSV data files that have been uploaded to OBS in **Step 1: Uploading Data to OBS**. This table is used to transfer data to ordinary tables in GaussDB(DWS).

The following foreign table and its parameter settings are provided as an example.

Parameters defined in a foreign table are used to identify data formats and set the error tolerance for data import. In this tutorial, only key parameter settings are provided. For more information, see **CREATE FOREIGN TABLE (for OBS Import and Export)**.


```
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';
```

If the following information is displayed, the foreign table has been created:

```
CREATE FOREIGN TABLE
```

Parameter settings in the foreign table are as follows:

- **SERVER**: It is always set to **gsmpp_server**. You do not need to change it.
- **LOCATION**: The OBS path consists of **obs://**, a bucket name, and a file path. Example: **obs://<bucket_name>/<file_path>**.

  NOTE

  Ensure no Chinese characters are contained in paths used for importing data to or exporting data from OBS.

  Multiple paths (if any) are separated by vertical bars (|). For example:

  ```
  LOCATION 'obs://mybucket/input_data/product_info  |  obs://mybucket02/input_data/product_info',
  ```

  The following two files having the specified prefix in the **mybucket** folder will be imported:

  ```
  mybucket/input_data/product_info0.csv
  mybucket/input_data/product_info1.csv
  ```

- FORMAT

  It is set to **CSV** because data to be imported in this tutorial is in CSV format. If the data is in other formats, set this parameter accordingly.

- DELIMITER

  It is set to **,** because the data in source data files is separated by commas (,).

- ENCODING

  It is set to **UTF-8**.

- HEADER

This parameter specifies whether a data file contains a header. It is valid only for data files in CSV format. The first line of data files in **Preparing Data Files** is not a header. Therefore, it is set to **false**.

- ACCESS_KEY and SECRET_ACCESS_KEY

  These parameters specify the AK and SK used to access OBS by a user. Replace them with the actual AK and SK. To obtain access keys, log in to the management console, move the cursor over the username in the upper right corner, and click **My Credentials**. Then choose **Access Keys** in the navigation tree on the left. On the **Access Keys** page, you can view the existing AKs or click **Create Access Key** to create an AK/SK pair.

- FILL_MISSING_FIELDS

  This parameter specifies how to handle the problem that the last column of a row in a source data file is lost during data import. The default value is **false** or **off**. This parameter is set to **true** in this tutorial.

  - **true**/**on**: The last column is set to **NULL**. No error is reported.
  - **false**/**off**: Error "missing data for column "tt"" is reported.

    For example, the last column **product_comment_content** of the second record in the **product_info2.csv** source data file is lost. If **FILL_MISSING_FIELDS** is set to **false** or **off**, information similar to the following will be displayed in the error table during data import:

    ```
    missing data for column "product_comment_content"
    ```

- IGNORE_EXTRA_DATA

  This parameter specifies whether to ignore excessive columns when the number of columns in a source data file exceeds that defined in the foreign table. The default value is **false** or **off**. This parameter is set to **true** in this tutorial.

  - **true**/**on**: The excessive columns of a row are ignored. No error is reported.
  - **false**/**off**: Error "extra data after last expected column" is reported.

    For example, the number of columns in the third record in the **product_info2.csv** source data file is greater than that defined for the foreign table. If **IGNORE_EXTRA_DATA** is set to **false** or **off**, information similar to the following will be displayed in the error table during data import:

    ```
    extra data after last expected column
    ```

- READ ONLY

  Syntax defined in a foreign table can be used for both importing data to the GaussDB(DWS) cluster and for exporting data from the cluster. To import data to the cluster, use **READ ONLY** for the foreign table. To export data, use **WRITE ONLY**.

- LOG INTO

  This parameter specifies the error table that records data format error information during import. You only need to specify the table name, and do not need to create it in advance. GaussDB(DWS) automatically creates the table when a foreign table is created and automatically deletes it when the foreign table is deleted.

- PER NODE REJECT LIMIT

This parameter specifies the allowed number of data format errors on each DN during data import. If the number of errors exceeds the specified value on any DN, data import fails, an error is reported, and the system exits data import.

It is set to **unlimited** in this tutorial, indicating that all data format errors during import can be recorded.

----End

6.4 Step 3: Importing Data

Step 1 Create a table named **product_info** in the GaussDB(DWS) database to store the data imported from OBS.

```
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);
```

Step 2 Run **INSERT** to import data from OBS to the target table **product_info** through the foreign table **product_info_ext**.

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

- If information similar to the following is displayed, the data has been imported. Query the error table for data format errors. For details, see **Step 4: Analyzing and Handling Import Errors**.

  ```
  INSERT 0 20
  ```

- If there are import errors, troubleshoot by following instructions provided in **Step 4: Analyzing and Handling Import Errors** and try again.

Step 3 Run **SELECT** to view the data imported to the target table **product_info** in GaussDB(DWS).

```
SELECT * FROM product_info;
```

If the query result displays data specified in **Preparing Data Files**, the import is successful. The following information is displayed at the end of the query result:

```
(20 rows)
```

----End

6.5 Step 4: Analyzing and Handling Import Errors

This section describes how to handle the data format errors that occurred during data import.

Step 1 Query error information in the error table.

```
SELECT * FROM product_info_err ;
```

Step 2 Handle errors if any.

In this tutorial, there should be no error information in the error table.

Alternatively, you can change **FILL_MISSING_FIELDS** and **IGNORE_EXTRA_DATA** in the foreign table created in **Step 2: Creating a Foreign Table** to **false**, and import the data again. Then, query the error table. In this case, you will find the following data format error information:

- The last column **product_comment_content** of the second record in the **product_info2.csv** source data file is lost.
- The number of columns in the third record in the **product_info2.csv** source data file is greater than that defined for the foreign table.

```
postgres=# select * from product_info_err;
 nodeid |        begintime        |          filename          | rownum |                                    rawrecord                                    |                  detail
--------+-------------------------+----------------------------+--------+---------------------------------------------------------------------------------+-------------------------------------------
      0 | 2018-08-31 15:57:32.221265+08 | /input_data/product_info2.csv |      2 | 98,"FKQB-I",2017-09-15,"B",*2017 new shoes men",*red",*M",4345,2017-09-18,5473              | missing data for column "product_comment_content"
      0 | 2018-08-31 15:57:32.221265+08 | /input_data/product_info2.csv |      3 | 50,"DMQY-K",2017-09-21,"A",*2017 pants men",*red",*37",28,2017-09-25,58,"good","good","good" | extra data after last expected column
(2 rows)
```

For details about error tables and troubleshooting, see **Handling Import Errors**.

----End

6.6 Step 5: Improving Query Efficiency After Data Import

After data is imported, run the **ANALYZE** statement to generate table statistics. The execution plan generator uses the statistics to generate the most efficient query execution plan.

If a large number of rows were updated or deleted during import, run **VACUUM FULL** before **ANALYZE**. A large number of update and delete operations generate a lot of disk page fragments, reducing the query efficiency. **VACUUM FULL** can restore disk page fragments and return them to the OS.

Step 1 Run **VACUUM FULL** on the **product_info** table.

```
VACUUM FULL product_info
VACUUM
```

Step 2 Update statistics of the **product_info** table.

```
ANALYZE product_info;
ANALYZE
```

----End

6.7 Step 6: Cleaning Up Resources

After completing operations in this tutorial, if you no longer need to use the resources created during the operations, you can delete them to avoid resource waste or quota occupation.

Deleting the Foreign Table and Target Table

Step 1 If you have performed queries after importing data, run the following statement to delete the target table:

```
DROP TABLE product_info;
```

If the following output is displayed, the index has been deleted:

```
DROP TABLE
```

Step 2 Run the following statement to delete the foreign table:

```
DROP FOREIGN TABLE product_info_ext;
```

If the following output is displayed, the foreign table has been deleted:

```
DROP FOREIGN TABLE
```

----End

7 Tutorial: Using GDS to Import Data from a Remote Server

7.1 Overview

This tutorial demonstrates how to use General Data Service (GDS) to import data from remote servers to GaussDB(DWS).

In this tutorial, you will:

- Generate the source data files in CSV format to be used in this tutorial.
- Upload the source data files to a data server.
- Create foreign tables used for importing data from a data server to GaussDB(DWS) through GDS.
- Start GaussDB(DWS), create a table, and import data to the table.
- Analyze import errors based on the information in the error table and correct these errors.

7.2 Step 1: Preparing an ECS as the GDS Server

The following parts are included in this section:

- **Preparing an ECS as the GDS Server**

  Before using GDS to import and export data, prepare one or more Linux ECSs in the same VPC as the data warehouse cluster to install the GDS tool package. These ECSs serve as the GDS servers.

  - Importing data

    Before data import, upload source data files to GDS servers. Then, the GDS servers can also be called data servers.

    If source data files are uploaded to multiple servers for storage due to large data volume, you need to install, configure, and start GDS on each data server.

  - Exporting data

    Data is exported from GaussDB(DWS) as data files to GDS servers.

- **Downloading the GDS Tool Package and SSL Certificate**

  Before installing GDS, log in to the GaussDB(DWS) management console and download the GDS tool package and SSL certificate that match the cluster version.

Preparing an ECS as the GDS Server

For details about how to purchase a Linux ECS, see **Purchasing an ECS with Customized Configurations** in the *Elastic Cloud Server Getting Started*. For details about how to log in to the ECS, see **Logging In to a Linux ECS**.

The ECS that functions as the GDS server must meet the following requirements:

- The GDS server must use an operating system supported by the GDS tool package. For details, see **Downloading the GDS Tool Package and SSL Certificate**.
- The GDS server can communicate with the data warehouse cluster.
  - The data warehouse cluster connects to the GDS server using the private network address, and they must belong to the same region, VPC, and subnet.

    The data import rate is affected by the network bandwidth. Therefore, the private network address is recommended.
  - The GDS server can properly receive network access requests from the data warehouse cluster.

    Port: Plan a listening port for GDS. This port is used by the data warehouse cluster to connect to the GDS server. You need to specify the listening port when enabling GDS. If it is not specified, the default port 8098 is used.

    Firewall: If a firewall is enabled on the GDS server, add an inbound rule for the GDS listening port to allow the data warehouse cluster to connect to the GDS server. Otherwise, the connection cannot be set up.

    NOTE

    Note that the listening port must be specified to the port that has been opened on the firewall when enabling GDS.

    The following example describes how to create an ECS as the GDS server.

    First of all, open the GDS listening port in the inbound rule of the security group where the ECS resides.

Table 7-1 Inbound rule example

Parameter	Example Value
Protocol	TCP
Port Range	5000 **NOTE** Enter the listening port of the GDS server.

Parameter	Example Value
Source	Select **IP Address** and enter the IP address of the data warehouse cluster. Example value: **192.168.0.10/32**.

If the firewall is enabled on the ECS, ensure that the listening port of the GDS server is opened on the firewall.

```
iptables  -I INPUT -p tcp -m tcp --dport <gds_port> -j ACCEPT
```

Downloading the GDS Tool Package and SSL Certificate

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

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

Step 3 Select the GaussDB(DWS) client of the corresponding version from the drop-down list of **gsql CLI Client**.

Choose a corresponding client version according to the cluster version and operating system to which the client is to be installed.

- The **Redhat x86_64** client can be used on the following operating systems:
  - RHEL 6.4 to RHEL 7.6.
  - CentOS 6.4 to CentOS 7.4
  - EulerOS 2.3
- The **SUSE x86_64** client can be used on the following operating systems:
  - SLES 11.1 to SLES 11.4
  - SLES 12.0 to SLES 12.3
- The **Euler Kunpeng_64** client can be used on the following operating systems:
  - EulerOS 2.8
- The **Stream Euler X86_64** client can be used on the following operating system:
  - EulerOS 2.2
- The **Stream Euler Kunpeng_64** client can be used on the following operating system:
  - EulerOS 2.8

Figure 7-1 Downloading a gsql client

NOTE

The CPU architecture of the client must be the same as that of the cluster. If the cluster uses x86 servers, select a x86 client.

Step 4 Click **Download**.

Step 5 (Optional) To start the GDS service in SSL encryption mode, click **here** to download the SSL certificate in the **Download Client and Driver** area.

If GDS is started in SSL encryption mode, data transmitted between the GDS server and the data warehouse cluster is encrypted to ensure data security.

NOTE

The SSL mode is more secure than other modes. You are advised to start GDS in SSL mode.

----End

7.3 Step 2: Preparing Source Data

You can import data in TEXT, CSV, or FIXED format from a remote server to GaussDB(DWS) clusters. This tutorial uses data in CSV format as an example. The method is the same for TEXT and FIXED data except that the parameter settings of foreign tables are different. For details, see **Using GDS to Import Data from a Remote Server**.

Preparing Source Data Files

To demonstrate how to import multiple files, this tutorial uses the following three CSV data files as an example. Generally, the source data files are exported from a database. In this tutorial, the CSV source data files are manually created.

The sample files here are the same as those used in "Tutorial: Importing Data from OBS to a Cluster." If you have retained those sample files, directly **upload them to the data server**.

- Data file **product_info0.csv**

  The file contains the following data:

  ```
  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.
  ```

- Data file **product_info1.csv**

  The file contains the following data:

  ```
  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.
  ```

- Data file **product_info2.csv**

  The file contains the following data:

  ```
  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."
```

CSV is short for Comma Separated Values. A CSV file is similar to a TXT or DOC file. It can also be considered a text file containing records, which are separated into columns by commas (,) or tabs. The column sequence in each record is the same. In Windows, CSV files can be opened in different applications, such as Notepad, Excel, and Notepad++.

The following describes how to generate a CSV file in Windows:

Step 1 Create a text file and open it in Notepad++. Copy the sample data into it. Then, check the total number of rows and check whether the data of rows is correctly separated.

Step 2 Choose **Format** > **Encode in UTF-8 without BOM**.

Step 3 Choose **File** > **Save as**.

Step 4 In the displayed dialog box, enter the file name and click **Save**.

To identify the file type, use the file name extension .csv when entering the file name.

----End

Uploading Source Data Files to a Data Server

Step 1 Log in as user **root** to the server 192.168.0.90 storing source data files (also known as the data server or GDS server).

Step 2 Create the directory **/input_data**.

```
mkdir -p /input_data
```

Step 3 Use MobaXterm to upload source data files to the created directory.

----End

7.4 Step 3: Installing, Configuring, and Starting GDS on a Data Server

This topic describes how to install, configure, and start GDS on the servers where source data files are stored. Then you can connect GDS to GaussDB(DWS) to import data.

Step 1 Before using GDS to import or export data, perform the "Preparing an ECS as the GDS Data Server" and "Downloading the GDS Package and SSL Certificate" operations in **Step 1: Preparing an ECS as the GDS Server**.

Step 2 Log in as user **root** to the data server (192.168.0.90) where GDS is to be installed and run the following command to create the **/opt/bin/dws** directory for storing the GDS package:

```
mkdir -p /opt/bin/dws
```

Step 3 Upload the GDS package to the created directory.

Use the SUSE Linux package as an example. Upload the GDS package **dws_client_redhat_x64.tar.gz** to the directory created in the previous step.

Step 4 Go to the new directory and decompress the package.

```
cd /opt/bin/dws/gds
tar -zxvf dws_client_redhat_x64.tar.gz
```

Step 5 (Optional) If SSL is used, upload the SSL certificate to the directory created in **Step 2**.

Step 6 Create the user **gds_user** and the user group **gdsgrp** it belongs to. This user is used to start GDS and must have the permission to read the source data file directory.

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

Step 7 Change the owner of the GDS package and source data file directory to **gds_user** and the user group to **gdsgrp**.

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

Step 8 Switch to user **gds_user**.

```
su - gds_user
```

Step 9 Start the GDS.

- If SSL encryption is not used, run the following command to start GDS:

  ```
  /opt/bin/dws/gds/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
  ```

- If SSL is enabled, run the following command to start GDS after performing **Step 5**:

  ```
  /opt/bin/dws/gds/gds -d /input_data/ -p 192.168.0.90:5000 -H 10.10.0.1/24 -l /opt/bin/dws/gds/gds_log.txt -D --enable-ssl --ssl-dir /opt/bin/dws/gds
  ```

Replace the italic parts as required.

- **-d dir**: directory storing data files that contain data to be imported. It is **/input_data/** in this tutorial.
- **-p ip:port**: listening IP address and port for GDS. The default value is **127.0.0.1**. Replace it with the IP address of a 10GE network that can communicate with GaussDB(DWS). The listening port can be any one ranging from **1024** to **65535**. The default port is **8098**. This parameter is set to **192.168.0.90:5000** in this tutorial.
- **-H address_string**: network segment for hosts that can connect to and use GDS. The value must be in CIDR format. Set this parameter to enable GaussDB(DWS) to access GDS for data import. Ensure that the network segment covers all hosts in GaussDB(DWS).
- **-l log_file**: GDS log directory and log file name. It is **/opt/bin/dws/gds/gds_log.txt** in this tutorial.
- **-D**: GDS in daemon mode. This parameter is used only in Linux.
- **--enable-ssl**: enables SSL for data transmission.
- **--ssl-dir**: directory storing the SSL certificate. Set it to the certificate directory mentioned in **Step 5**.

----End

7.5 Step 4: Creating a Foreign Table in the GaussDB(DWS) Database

Step 1 Use the SQL client tool to connect to the GaussDB(DWS) database.

Step 2 Create a foreign table based on **Table 7-2**.

```
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';
```

If the following information is displayed, the foreign table has been created:

```
CREATE FOREIGN TABLE
```

Table 7-2 Configurations in the foreign table

Configuration Item	Value	Description
SERVER	gsmpp_server	This parameter is always set to **gsmpp_server**. You do not need to change it.
LOCATION	gsfs://192.168.0.90:5000/*	This parameter specifies the location of source data files. If SSL transmission is used, use the gsfss protocol. In this tutorial, the location is **gsfss://192.168.0.90:5000/***.
FORMAT	CSV	This parameter specifies the format of source data files.

Configuration Item	Value	Description
ENCODING	UTF-8	This parameter specifies the data encoding.
DELIMITER	It is set to a comma (,).	This parameter specifies the column delimiter.
HEADER	**false** (default value)	This parameter specifies whether a data file contains a header. This parameter is valid only for data files in CSV format. Data files in **Preparing Source Data Files** do not contain a header. Therefore, this parameter is set to **false**.
FILL_MISSING_FIELDS	true	This parameter specifies how to handle the problem that the last column of a row in a source data file is lost during data import. The default value is **false** or **off**. **true** is used in this tutorial. • **true**/**on**: The last column is set to **NULL**. No error is reported. • **false**/**off**: Error "missing data for column "tt"" is reported. For example, the last column **product_comment_content** of the second row in the **product_info2.csv** source data file is lost. If **FILL_MISSING_FIELDS** is set to **false** or **off**, information similar to the following will be displayed in the error table during data import: **missing data for column** "product_comment_content"

Configuration Item	Value	Description
IGNORE_EXTRA_DATA	true	This parameter specifies whether to ignore excessive columns when the number of columns in a source data file exceeds that defined in the foreign table. The default value is **false** or **off**. **true** is used in this tutorial. • **true**/**on**: The excessive columns of a row are ignored. No error is reported. • **false**/**off**: Error "extra data after last expected column" is reported. For example, the number of columns in the third record in the **product_info2.csv** source data file is greater than that defined for the foreign table. If **IGNORE_EXTRA_DATA** is set to **false** or **off**, information similar to the following will be displayed in the error table during data import: **extra data after last expected column**
PER NODE REJECT LIMIT 'value'	unlimited	This parameter specifies the allowed number of data format errors on each DN during data import. If the number of errors exceeds the specified value on any DN, data import fails, an error is reported, and the system exits data import. It is set to **unlimited** in this tutorial, indicating that all data format errors during import can be recorded.
READ ONLY	-	Syntax defined in a foreign table can be used for both importing data to the GaussDB(DWS) cluster and for exporting data from the cluster. To import data to the cluster, use **READ ONLY** for the foreign table. To export data, use **WRITE ONLY**.
WITH error_table_name	Error table name: **product_info_err**	Data format errors during import are recorded in the table specified by **product_info_err**. You can query this table after the import to obtain error details.

----End

7.6 Step 5: Importing Data to GaussDB(DWS)

Step 1 Run the following statements to create the **product_info** table in GaussDB(DWS) to store imported data:

```
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);
```

Step 2 (Optional) This step is not required in this example because no index is created in **Step 1**. If the target table has indexes, the index information will be incrementally updated during import, affecting data import performance. You are advised to delete the indexes from the target table before the import. You can create index again after the import is complete.

1. Assume that the ordinary index **product_idx** exists in the **product_id** column of the target table **product_info**. Delete the index from the table before importing data.
   ```
   DROP INDEX product_idx;
   ```
2. After importing the data, create the index again.
   ```
   CREATE INDEX product_idx ON product_info(product_id);
   ```

Step 3 Import data from source data files to the **product_info** table through the foreign table **product_info_ext**.

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

If information similar to the following is displayed, the data has been imported:

```
INSERT 0 20
```

Step 4 Run **SELECT** to view the data imported to GaussDB(DWS).

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

If the following information is displayed, the import is successful:

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

----End

7.7 Step 6: Analyzing and Handling Import Errors

This section describes how to handle data format errors that occurred during data import. If no error information is reported, skip this step.

Step 1 Query error information in the error table.

```
SELECT * FROM product_info_err ;
```

Step 2 Handle errors if any.

In this tutorial, there should be no error information in the error table.

Alternatively, you can change **FILL_MISSING_FIELDS** and **IGNORE_EXTRA_DATA** in the foreign table created in **Step 4: Creating a Foreign Table in the GaussDB(DWS) Database** to **false**, and import the data again. Then, query the error table. In this case, you will find the following data format error information:

- The last column **product_comment_content** of the second row in the **product_info2.csv** source data file is lost.
- The number of columns in the third record in the **product_info2.csv** source data file is greater than that defined for the foreign table.

```
postgres=# select * from product_info_err;
 nodeid |          begintime          |          filename           | rownum |                                rawrecord                                 |                  detail
--------+-----------------------------+-----------------------------+--------+--------------------------------------------------------------------------+-------------------------------------
      0 | 2018-08-31 15:57:32.221265+08 | /input_data/product_info2.csv |      2 | 98,"FKQB-I",2017-09-15,"B","2017 new shoes men","red","M",4345,2017-09-18,5473              | missing data for column "product_comment_content"
      0 | 2018-08-31 15:57:32.221265+08 | /input_data/product_info2.csv |      3 | 50,"DMQY-K",2017-09-21,"A","2017 pants men","red","37",28,2017-09-25,58,"good","good","good" | extra data after last expected column
(2 rows)
```

For details about error tables and troubleshooting, see **Handling Import Errors**.

----End

7.8 Step 7: Optimizing the Query Efficiency After Data Import

After data is imported, run the **ANALYZE** statement to generate table statistics. The execution plan generator uses the statistics to generate the most efficient query execution plan.

If a large number of rows were updated or deleted during import, run **VACUUM FULL** before **ANALYZE**. A large number of update and delete operations generate a lot of disk page fragments, reducing the query efficiency. **VACUUM FULL** can restore disk page fragments and return them to the OS.

Step 1 Run **VACUUM FULL** on the **product_info** table.

```
VACUUM FULL product_info
VACUUM
```

Step 2 Update statistics of the **product_info** table.

```
ANALYZE product_info;
ANALYZE
```

----End

7.9 Step 8: Stopping GDS

Stop GDS after data is imported successfully.

Procedure

Step 1 Log in as user **gds_user** to the data server where GDS is installed.

Step 2 Perform the following operations to stop GDS:

1. Query the GDS process ID: The GDS process ID is 128954.

```
ps -ef|grep gds
gds_user 128954      1  0 15:03 ?        00:00:00 gds -d /input_data/ -p 192.168.0.90:5000 -
l /opt/bin/dws/gds/gds_log.txt  -D
gds_user 129003 118723  0 15:04 pts/0    00:00:00 grep gds
```

2. Run the **kill** command to stop GDS. **128954** in the command is the GDS process ID.

```
kill -9 128954
```

----End

7.10 Step 9: Cleaning Up Resources

After completing operations in this tutorial, if you no longer need to use the resources created during the operations, you can delete them to avoid resource waste or quota occupation.

Deleting the Foreign Table and Target Table

Step 1 Run the following command to delete the target table **product_info**:

```
DROP TABLE product_info;
```

If the following output is displayed, the index has been deleted:

```
DROP TABLE
```

Step 2 Run the following command to delete the foreign table **product_info_ext**:

```
DROP FOREIGN TABLE product_info_ext;
```

If the following output is displayed, the foreign table has been deleted:

```
DROP FOREIGN TABLE
```

----End

8 Analysis of Passed Vehicles at Traffic Checkpoints

This practice demonstrates the analysis of passed vehicles at traffic checkpoints. In this practice, 890 million pieces of data from traffic checkpoints are loaded to a single database table on GaussDB(DWS) for performing accurate query and fuzzy query. It is an example of high-performance query of historical data on GaussDB(DWS).

NOTE

The sample data has been uploaded to the **traffic-data** folder in an OBS bucket, and all HUAWEI CLOUD accounts have been granted the read-only permission to access the OBS bucket.

This practice takes about 40 minutes. The basic process is as follows:

1. **Making Preparations**
2. **Step 1: Creating a Cluster**
3. **Step 2: Using Data Studio to Connect to a Cluster**
4. **Step 3: Importing Sample Data**
5. **Step 4: Performing Vehicle Analysis**

Making Preparations

- Register a HUAWEI CLOUD account and check the account status before using GaussDB (DWS). The account cannot be in arrears or frozen.
- Obtain the AK/SK of the account. For details, see **Creating Access Keys (AK and SK)**.

Step 1: Creating a Cluster

Step 1 Log in to the HUAWEI CLOUD management console.

Step 2 Choose **Service List** > **EI Enterprise Intelligence** > **Data Warehouse Service**.

Step 3 In the navigation pane on the left, choose **Cluster Management**. On the displayed page, click **Buy DWS Cluster** in the upper right corner.

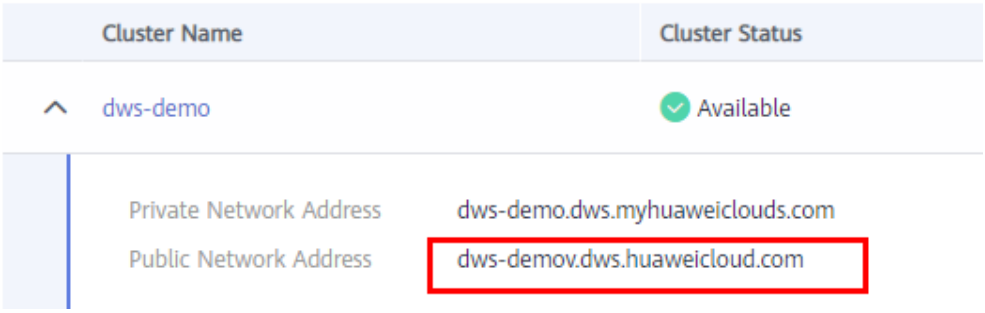
Step 4 Configure the parameters according to **Table 8-1**.

Table 8-1 Software configuration

Parameter	Configuration Method
Region	Select the **CN North-Beijing4**. **NOTE** This section uses **CN North-Beijing4** as an example. If you want to perform operations in other regions, ensure that all operations are performed in the same region.
AZ	AZ2
Cluster Type	Standard
CPU Architecture	x86
Node Flavor	dws2.m6.4xlarge.8 (16 vCPU | 128 GB | 2000 GB SSD)
Nodes	3
Cluster Name	dws-demo
Administrator Account	dbadmin
Administrator Password	Demo@123456
Confirm Password	Demo@123456
Database Port	8000
VPC	vpc-default
Subnet	subnet-default(192.168.0.0/24)
Security Group	Automatic creation
EIP	Buy now
Bandwidth	1 Mbit/s
Advanced Settings	Default

Step 5 Confirm the information, click **Buy Now**, and then click **Submit**.

Step 6 Wait about 6 minutes. After the cluster is created, click next to the cluster name. On the displayed cluster information page, record the **Public Network Address**, for example, **dws-demov.dws.huaweicloud.com**.



----End

Step 2: Using Data Studio to Connect to a Cluster

Step 1 Ensure that JDK 1.8.0 or later has been installed on the client host. Choose **PC > Properties > Advanced System Settings > Environment Variables** and set **JAVA_HOME** (for example, **C:\Program Files\Java\jdk1.8.0_191**). Add **; %JAVA_HOME%\bin** to the variable **path**.

Step 2 On the **Connection Management** page of the GaussDB(DWS) console, download the Data Studio GUI client.

Step 3 Decompress the downloaded Data Studio software package, go to the decompressed directory, and double-click **Data Studio.exe** to start the client.

Step 4 On the Data Studio main menu, choose **File > New Connection**. In the dialog box that is displayed, configure the connection based on **Table 8-2**.

Table 8-2 Data Studio software configuration

Parameter	Configuration Method
Database Type	GaussDB(DWS)
Connection Name	dws-demo
Host	dws-demov.dws.huaweicloud.com The value of this parameter must be the same as the value of **Public Network Address** queried in **Step 1: Creating a Cluster**.
Host Port	8000
Database Name	postgres

Parameter	Configuration Method
User Name	dbadmin
Password	Demo@123456
Enable SSL	Disable

Step 5 Click **OK**.

----End

Step 3: Importing Sample Data

After connecting to the cluster using the SQL client tool, perform the following operations in the SQL client tool to import the sample data from traffic checkpoints and perform data queries.

Step 1 Execute the following statement to create the **traffic** database:

```
create database traffic encoding 'utf8' template template0;
```

Step 2 Perform the following steps to switch to the new database:

1. In the **Object Browser** window of the Data Studio client, right-click the database connection and choose **Refresh** from the shortcut menu. Then, the new database is displayed.
2. Right-click the name of the new database **traffic** and choose **Connect to DB** from the shortcut menu.
3. Right-click the name of the new database **traffic** and choose **Open Terminal** from the shortcut menu. The SQL command window for connecting to the specified database is displayed. Perform the following steps in the window.

Step 3 Execute the following statements to create a database table for storing vehicle information from traffic checkpoints:

```
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);
```

Step 4 Create a foreign table, which is used to identify and associate the source data on OBS. Replace *<Access_Key_Id>* and *<Secret_Access_Key>* with the actual values obtained in **Making Preparations**.

```
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://dws-demo-cn-north-4/traffic-data/gcxx',
        format 'text',
        delimiter ',',
        access_key '<Access_Key_Id>',
        secret_access_key '<Secret_Access_Key>',
        chunksize '64',
        IGNORE_EXTRA_DATA 'on'
);
```

Step 5 Execute the following statement to import data from the foreign table to the database table:

```
insert into traffic_data.GCJL select * from tpchobs.GCJL_OBS;
```

It takes some time to import data.

----End

Step 4: Performing Vehicle Analysis

1. **Executing Analyze**

   This statement collects statistics related to common table content in databases. The statistics are saved under the system directory PG_STATISTIC. The statistics are useful when you run the planner, which provides you with an efficient query execution plan.

   Execute the following statement to generate the statistics related to the tables:

   ```
   Analyze;
   ```

2. **Querying the data volume of the data table**

   Execute the following statement to query the number of loaded data records:

   ```
   set current_schema= traffic_data;
   Select count(*) from traffic_data.gcjl;
   ```

3. **Accurate vehicle query**

   Execute the following statement to query driving route by license plate number and time segment: GaussDB(DWS) responds to the request in seconds.

   ```
   set current_schema= traffic_data;
   select hphm, kkbh, gcsj
   from traffic_data.gcjl
   where hphm = 'YueD38641'
   and gcsj between '2016-01-06' and '2016-01-07'
   order by gcsj desc;
   ```

4. **Fuzzy vehicle query**

   Execute the following statement to query vehicle tracks by license plate number and time segment. GaussDB(DWS) responds to the request in seconds.

   ```
   set current_schema= traffic_data;
   select hphm, kkbh, gcsj
   from traffic_data.gcjl
   where hphm like '%A23F%'
   and kkbh in('508', '1125', '2120')
   and gcsj between '2016-01-01' and '2016-01-07'
   order by hphm,gcsj desc;
   ```


9 Supply Chain Requirement Analysis of a Company

This practice describes how to load the sample data set from OBS to a data warehouse cluster and perform data queries. This example comprises multi-table analysis and theme analysis in the data analysis scenario.

NOTE

In this example, a standard TPC-H-1x data set of 1 GB size has been generated on GaussDB(DWS), and has been uploaded to the **tpch** folder of an OBS bucket. All HUAWEI CLOUD accounts have been granted the read-only permission to access the OBS bucket. Users can easily import the data set using their accounts.

This practice takes about 60 minutes. The basic process is as follows:

1. **Making Preparations**
2. **Step 1: Importing Sample Data**
3. **Step 2: Performing Multi-Table Analysis and Theme Analysis**

Scenario Description

Purpose: Understand the basic functions and data import of GaussDB (DWS) and analyze the order data of a company and its suppliers. The analysis dimensions are as follows:

1. Analyze the revenue brought by suppliers in a region to the company. The statistics can be used to determine whether a local allocation center needs to be established in a given region.
2. By analyzing the relationship between parts and suppliers, you can obtain the number of suppliers that can supply parts based on the specified contribution conditions. This information can be used to determine whether there are sufficient suppliers when the order quantity is large and the task is urgent.
3. Analyze the revenue loss of small orders. You can query the average annual revenue loss if there are no small orders. Filter out small orders that are lower than the 20% of the average supply volume, and calculate the total amount of those small orders to figure out the average annual revenue loss.

Making Preparations

- Register a HUAWEI CLOUD account and check the account status before using GaussDB (DWS). The account cannot be in arrears or frozen.
- Obtain the AK/SK of the account. For details, see **Creating Access Keys (AK and SK)**.
- A cluster has been created and connected using Data Studio. For details, see **Analysis of Passed Vehicles at Traffic Checkpoints**.

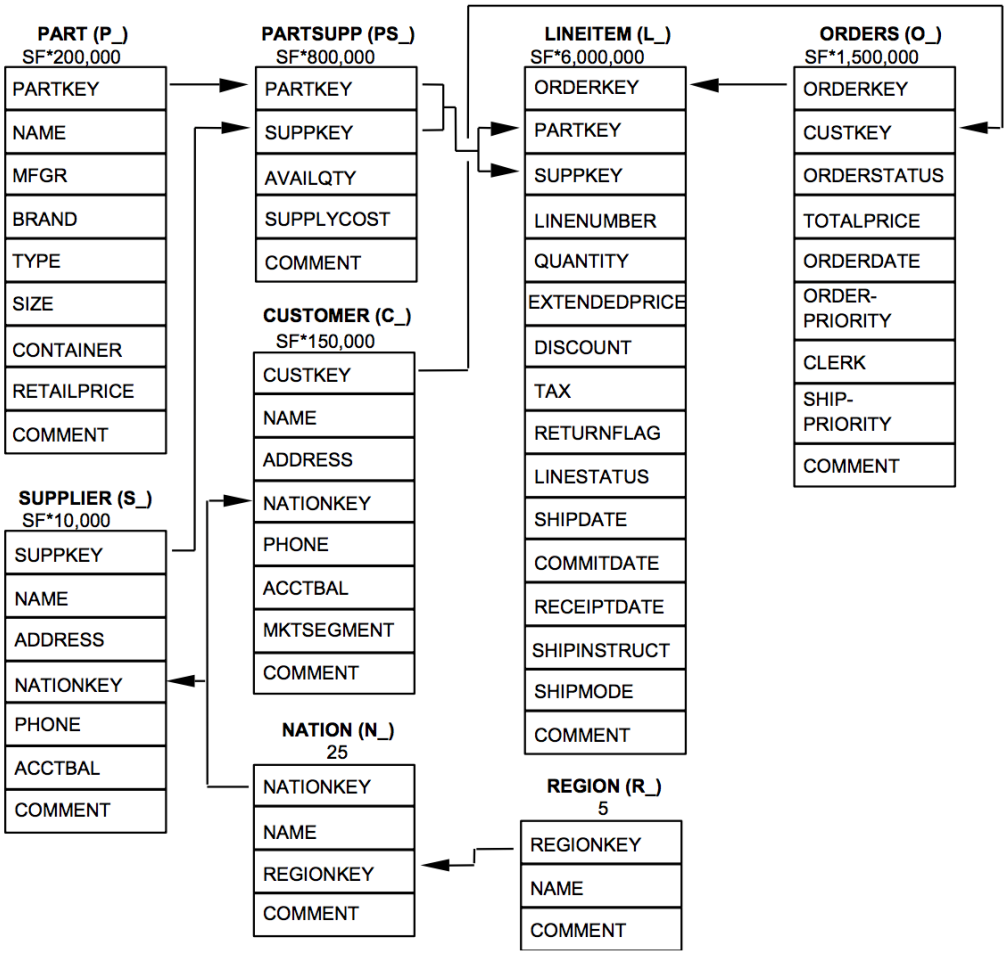
Step 1: Importing Sample Data

After connecting to the cluster using the SQL client tool, perform the following operations in the SQL client tool to import the TPC-H sample data and perform data queries.

Step 1 Create a database table.

The TPC-H sample data consists of eight database tables whose associations are shown in **Figure 9-1**.

Figure 9-1 TPC-H data tables



Copy and execute the following table creation statements to create corresponding data tables in the **postgres** database.

```
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);
```

Step 2 Create a foreign table, which is used to identify and associate the source data on OBS.

<obs_bucket_name > indicates the OBS bucket name. In this practice, **CN North-Beijing4** is used as an example. You can enter **dws-demo-cn-north-4**. Replace *<Access_Key_Id >* and *<Secret_Access_Key >* with the actual values obtained in **Making Preparations**.

```
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'
);
```

Step 3 Copy and execute the following statements to import the foreign table data to the corresponding database table.

Run the **insert** command to import the data in the OBS foreign table to the GaussDB(DWS) database table. The database kernel concurrently imports the OBS data at a high speed to 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;
```

It takes 10 minutes to import data.

----End

Step 2: Performing Multi-Table Analysis and Theme Analysis

The following uses standard TPC-H query as an example to demonstrate how to perform basic data query on GaussDB(DWS).

Before querying data, run the **Analyze** command to generate statistics related to the database table. The statistics data is stored in system table PG_STATISTIC and is useful when you run the planner, which provides you with an efficient query execution plan.

The following are querying examples:

- **Querying revenue of a supplier in a region (TPCH-Q5)**

  By executing the TPCH-Q5 query statement, you can query the revenue statistics of a spare parts supplier in a region. The revenue is calculated based on **sum(l_extendedprice * (1 - l_discount))**. The statistics can be used to determine whether a local allocation center needs to be established in a given region.

  Copy and execute the following TPCH-Q5 statement for query. This statement features multi-table connection query operations with group by, sort by, aggregate, and subquery.

  ```
  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;
  ```

- **Querying relationships between spare parts and suppliers (TPCH-Q16)**

  By executing the TPCH-Q16 query statement, you can obtain the number of suppliers that can supply spare parts with the specified contribution conditions. This information can be used to determine whether there are sufficient suppliers when the order quantity is large and the task is urgent.

Copy and execute the following TPCH-Q16 statement for query. The statement features multi-table connection operations with group by, sort by, aggregate, deduplicate, and NOT IN subquery.

```
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;
```

- **Querying revenue loss of small orders (TPCH-Q17)**

  You can query the average annual revenue loss if there are no small orders. Filter out small orders that are lower than the 20% of the average supply volume, and calculate the total amount of those small orders to figure out the average annual revenue loss.

  Copy and execute the following TPCH-Q17 statement for query. The statement features multi-table connection operations with aggregate and aggregate subquery.

```
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
);
```


10 Operations Status Analysis of a Retail Department Store

Background

In this practice, the daily business data of each retail store is loaded from OBS to the corresponding table in the data warehouse cluster for summarizing and querying KPIs. This data includes store turnover, customer flow, monthly sales ranking, monthly customer flow conversion rate, monthly price-rent ratio, and sales per unit area. This example demonstrates the multidimensional query and analysis of GaussDB(DWS) in the retail scenario.

NOTE

The sample data has been uploaded to the **retail-data** folder in an OBS bucket, and all HUAWEI CLOUD accounts have been granted the read-only permission to access the OBS bucket.

This practice takes about 60 minutes. The basic process is as follows:

1. **Making Preparations**
2. **Step 1: Importing Sample Data from the Retail Department Store**
3. **Step 2: Performing Operations Status Analysis**

Making Preparations

- Register a HUAWEI CLOUD account and check the account status before using GaussDB (DWS). The account cannot be in arrears or frozen.
- Obtain the AK/SK of the account. For details, see **Creating Access Keys (AK and SK)**.
- A cluster has been created and connected using Data Studio. For details, see **Analysis of Passed Vehicles at Traffic Checkpoints**.

Step 1: Importing Sample Data from the Retail Department Store

After connecting to the cluster using the SQL client tool, perform the following operations in the SQL client tool to import the sample data from retail department stores and perform queries.

Step 1 Execute the following statement to create the **retail** database:

```
create database retail encoding 'utf8' template template0;
```

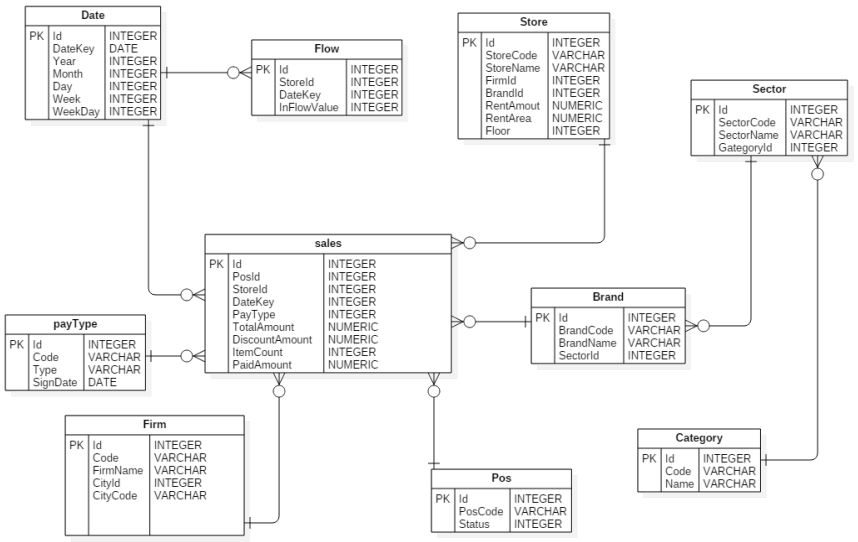
Step 2 Perform the following steps to switch to the new database:

1. In the **Object Browser** window of the Data Studio client, right-click the database connection and choose **Refresh** from the shortcut menu. Then, the new database is displayed.
2. Right-click the name of the new database **retail** and choose **Connect to DB** from the shortcut menu.
3. Right-click the name of the new database **retail** and choose **Open Terminal** from the shortcut menu. The SQL command window for connecting to the specified database is displayed. Perform the following steps in the window.

Step 3 Create a database table.

The sample data consists of 10 database tables whose associations are shown in **Figure 10-1**.

Figure 10-1 Sample data tables of retail department stores



Copy and execute the following statements to switch to create a database table of retail department store information.

```
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);
```

Step 4 Create a foreign table, which is used to identify and associate the source data on OBS.

<obs_bucket_name > indicates the OBS bucket name.In this practice, **CN North-Beijing4** is used as an example. You can enter **dws-demo-cn-north-4**. Replace *<Access_Key_Id >* and *<Secret_Access_Key >* with the actual values obtained in **Making Preparations**.

```
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'
);
```

Step 5 Copy and execute the following statements to import the foreign table data to the cluster:

```
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;
```

It takes some time to import data.

Step 6 Copy and execute the following statement to create the **v_sales_flow_details** view:

```
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;
```

----End

Step 2: Performing Operations Status Analysis

The following uses standard query of retail information from department stores as an example to demonstrate how to perform basic data query on GaussDB(DWS).

Before querying data, run the **Analyze** command to generate statistics related to the database table. The statistics data is stored in system table PG_STATISTIC and is useful when you run the planner, which provides you with an efficient query execution plan.

The following are querying examples:

- **Querying the total sales revenue of each store**

  Copy and execute the following statements to query the total turnover of each store:

  ```
  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;
```

- **Querying the sales revenue and price-rent ratio of each store**

  Copy and execute the following statement to query the sales revenue and price-rent ratio of each store:

```
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;
```

- **Analyzing the sales revenue of each province**

  Copy and execute the following statement to analyze and query the sales revenue of all provinces:

```
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;
```

- **Analyzing and comparing the price-rent ratio and customer flow conversion rate of each store**

```
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;
```

- **Analyzing brands in the retail industry**

```
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;
```

- **Querying daily sales information of each brand**

```
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;
```