

GaussDB(DWS) 3.0

Developer Guide

Issue 02
Date 2024-07-19



Copyright © Huawei Cloud Computing Technologies Co., Ltd. 2024. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of Huawei Cloud Computing Technologies Co., Ltd.

Trademarks and Permissions



HUAWEI and other Huawei trademarks are the property of Huawei Technologies Co., Ltd.

All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

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

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

Contents

1 Reading Guide.....	1
2 Introduction to GaussDB(DWS) 3.0.....	2
3 Support and Constraints.....	6
4 SQL Syntax Reference.....	12
4.1 CREATE TABLE.....	12
4.2 CREATE EXTERNAL SCHEMA.....	29
4.3 ALTER EXTERNAL SCHEMA.....	31
4.4 ALTER TABLE.....	32
5 Function.....	48
6 System Catalogs.....	63
6.1 PG_CLASS.....	63
6.2 PG_CONSTRAINT.....	68
6.3 PG_EXTERNAL_NAMESPACE.....	70
6.4 PG_NAMESPACE.....	71
6.5 PG_PARTITION.....	71
6.6 PG_REWRITE.....	73
6.7 PG_TRIGGER.....	74
6.8 PGXC_GROUP.....	75
6.9 PGXC_NODE.....	77
7 System Views.....	79
7.1 PGXC_DISK_CACHE_STATS.....	79
7.2 PGXC_DISK_CACHE_PATH_INFO.....	80
7.3 PGXC_DISK_CACHE_ALL_STATS.....	80
7.4 PGXC_OBS_IO_SCHEDULER_STATS.....	82
7.5 PGXC_OBS_IO_SCHEDULER_PERIODIC_STATS.....	83
8 GUC Parameters.....	86
9 Development Practices.....	95
9.1 Data Reading/Writing Across Logical Clusters.....	95
9.2 Data Lakehouse.....	97

9.2.1 Accessing HiveMetaStore Across Clusters..... 97

1 Reading Guide

This document describes how to develop and use the cloud-native data warehouse **GaussDB(DWS) 3.0** (DWS 3.0 for short). Sections in this document such as Syntax Reference, System Tables, and GUC Parameters in this document describe only the data warehouse **GaussDB(DWS) 3.0**.

Other common syntaxes, system catalogs, views, functions, and GUC parameters are not described in this document. For details, see *Development Guide* and *SQL Syntax Reference* for **GaussDB(DWS) 2.0**.

2 Introduction to GaussDB(DWS) 3.0

The newly released GaussDB(DWS) 3.0 version provides resource pooling, massive storage, and the MPP architecture with decoupled computing and storage. This enables high elasticity, real-time data import and sharing, and lake warehouse integration.

Description

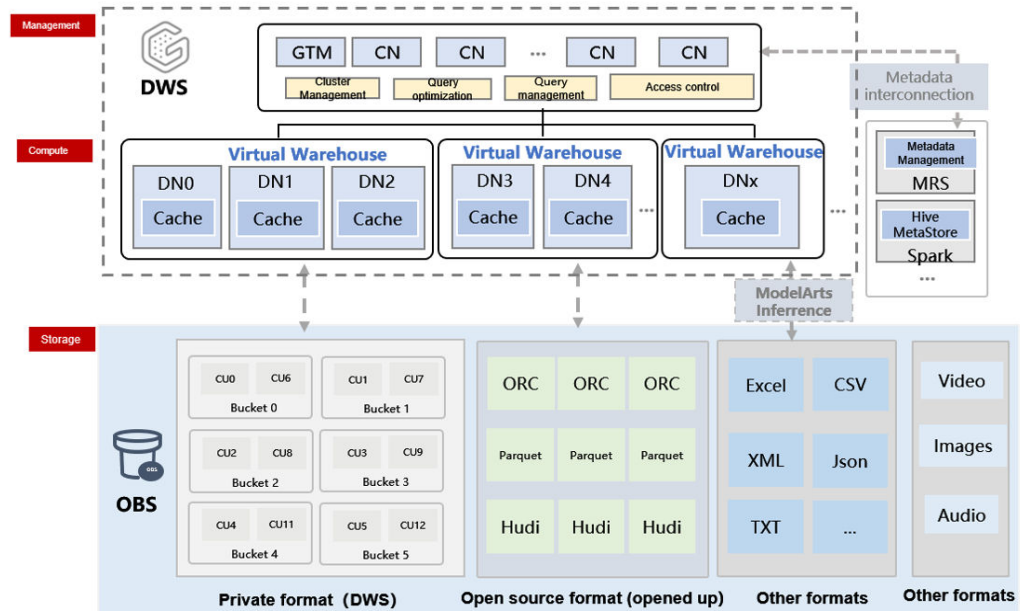
GaussDB(DWS) 3.0 uses decoupled computing and storage, which enables independent scaling of compute and storage resources. This feature enables users to quickly and independently scale computing capabilities during peak and off-peak hours. Storage can be expanded without limitation and paid on-demand to quickly and agilely responds to service changes with higher cost-effectiveness.

GaussDB(DWS) 3.0 has the following advantages:

- **Lakehouse:** GaussDB(DWS) 3.0 provides an integrated lakehouse that is easier to maintain and operate. It seamlessly interconnects with DLI, supports automatic metadata import, external table query acceleration, joined query of internal and external tables, data lake format read and write, and simpler data import.
- **High elasticity:** Computing resources can be quickly scaled, storage space can be used on demand, greatly reducing the cost. Historical data does not need to be migrated to other storage media, enabling one-stop data analysis for industries such as finance and Internet.
- **Data sharing:** Multiple loads share one copy of data in real time, while the computing resources are isolated. Multiple writes and reads are supported.

Architecture

Figure 2-1 GaussDB(DWS) 3.0 architecture



- Serverless and cloud native
 - Decoupled storage, computing, and management layers; independent, flexible, and fast scaling of computing and storage resources
 - Cost-effective, meeting diverse workload requirements and strict load isolation requirements
- Highly scalable
 - Logical clusters (virtual warehouses) can be scaled in or out in many ways.
 - Data is shared among multiple logical clusters in real time. Multiple loads share one copy of data.
 - Logical clusters are used to linearly improve throughput and concurrency, and provide good read/write isolation and load isolation capabilities.
- Data lakehouse
 - Seamless hybrid query across data lakes and data warehouses
 - In data lake analysis, you can enjoy the ultimate performance and precise control of data warehouses.

Version Differences

Table 2-1 Differences between GaussDB(DWS) 3.0 and GaussDB(DWS) 2.0

Version	DWS 2.0	DWS 3.0
Application scenarios	Converged data analysis using OLAP. It is used in sectors such as finance, government and enterprise, e-commerce, and energy.	Converged analysis, and offline integrated OLAP analysis. Optimized for Internet scenarios.
Advantages	High cost-effectiveness Hot and cold data analysis and elastic scaling of storage and computing resources.	Low cost and high concurrency. Decoupled storage and compute, on-demand storage usage, rapid computing scaling, unlimited computing power, and unlimited capacity. Data sharing and lake warehouse integration.
Features	Excellent performance in interactive analysis and offline processing of massive data, as well as complex data mining.	Real-time data import, real-time analysis, offline processing, interactive query, and high performance for large-scale data and complex data mining.
SQL syntax	Compatible with the SQL syntax of the cloud data warehouse.	Compatible with the SQL syntax of the cloud data warehouse.
GUC parameter	You can configure a wide variety of GUC parameters to tailor your data warehouse environment.	You can configure a wide variety of GUC parameters to tailor your data warehouse environment.

Application Scenarios

- Data lakehouse

Seamless access to the data lake

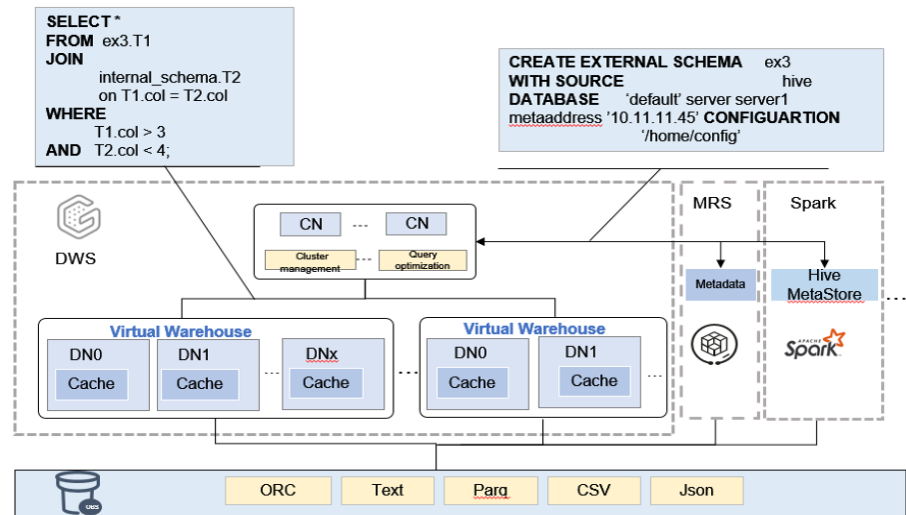
- With the interconnection with Hive Metastore metadata management, you can directly access the data table definitions in the data lake. You do not need to create a foreign table. You only need to create an external schema.
- The following data formats are supported: ORC and Parquet.

Convergent query

- Hybrid query of any data in the data lake and warehouse
- The query result is directly sent to the warehouse or data lake. No data needs to be transferred or copied.

Excellent query performance

- High-quality query plans and efficient execution engines
- Precise load management methods



- **Highly scalable**
Computing resources can be quickly scaled, storage space can be used on demand, greatly reducing cost. It is applicable to stable services and sensitive services.
 - Two scaling modes are provided. You can scale in or out the current cluster or add a logical cluster.
 - The scaling is performed very quickly without data redistribution or copy.
 - A logical cluster can improve concurrency and throughput. It can also be used to bind different services to different VWs to implement read/write isolation. It is applicable to scenarios where service loads change periodically, for example, batch service increase from 00:00 to 07:00.
- **Data sharing**
One copy of data carries various loads. Data can be shared in real time, and data of different services can be quickly shared.
 - Any logical cluster can carry read and write loads.
 - Data is visible shared among multiple logical clusters and does not need to be copied.

3 Support and Constraints

The differences between operations supported by GaussDB(DWS) 3.0 lie in the management console and database operations.

For details about the operation differences supported on the management console, see [Table 3-1](#). For details about the supported database capabilities, see [Table 3-2](#).

Table 3-1 Console Operation

Module	Function	DWS 2.0	DWS 3.0
Navigation menu	Dashboard	Yes	Yes
	Cluster Management	Yes	Yes
	DR management	Yes	No
	Snapshot management	Yes	Yes
	Parameters	Yes	Yes
	Incident management	Yes	Yes
	Alarm management	Yes	Yes
	Client connections	Yes	Yes
Dashboard	Resources	Yes	Yes
	Alarms	Yes	Yes
	Recent events	Yes	Yes
	Cluster monitoring metrics (DMS)	Yes	Yes

Module	Function	DWS 2.0	DWS 3.0
Cluster management	Monitoring panel (DMS)	Yes	Yes
	Monitoring metrics (Cloud Eye)	Yes	Yes
	Restart	Yes	Yes
	Scaling	Yes	Yes
	Redistributing data	Yes	Yes
	Viewing redistribution details	Yes	Yes
	Resetting passwords	Yes	Yes
	Creating snapshots	Yes	Yes
	Canceling read-only status	Yes	Yes
	Deletion	Yes	Yes
	Managing CNs	Yes	Yes
	Storage space scaling	Yes	Yes
Basic Information	Basic information	Yes	Yes
	ELB	Yes	Yes
	Resource pool	Yes	Yes
	Logical cluster	Yes	Yes
	Snapshot	Yes	Yes
	Parameter modifications	Yes	Yes
	Security settings	Yes	Yes
	MRS data sources	Yes	Yes
	Monitoring Panel	Yes	Yes
	Tags	Yes	Yes
	Node management	Yes	Yes

Module	Function	DWS 2.0	DWS 3.0
DR management	DR management	Yes	No
Snapshot management	Restoration	Yes	Yes
	Deletion	Yes	Yes
	Copy	Yes	Yes
Incident management	Event management (general)	Yes	Yes
Alarm management	Alarm management	Yes	Yes
Client connections	Client connections	Yes	Yes
Others	Inspection	Yes	Yes
	Intelligent O&M	Yes	Yes
	Node restoration	Yes	Yes
	Warm backup on the tenant side	Yes	Yes
	OpenApi	Yes	No

Table 3-2 Database operations

Type	Syntax	Supported
Basic functions	CREATE TABLE	Yes
	CREATE TABLE LIKE	Yes
	DROP TABLE	Yes
	INSERT	Yes
	COPY	Yes
	SELECT	Yes
	TRUNCATE	Yes
	EXPLAIN	Yes
	ANALYZE	Yes
	VACUUM	Yes
	ALTER TABLE DROP PARTITION	Yes
	ALTER TABLE ADD PARTITION	Yes

Type	Syntax	Supported
	ALTER TABLE SET WITH OPTION	Yes
	ALTER TABLE DROP COLUMN	Yes
	ALTER TABLE ADD COLUMN	Yes
	ALTER TABLE ADD NODELIST	Yes
	ALTER TABLE CHANGE OWNER	Yes
	ALTER TABLE RENAME COLUMN	Yes
	ALTER TABLE TRUNCATE PARTITION	Yes
	Other ALTER TABLE syntax	Yes
	CREATE INDEX	Yes
	DROP INDEX	Yes
	DELETE	Yes
	ALTER INDEX	Yes
	MERGE	Yes
	SELECT INTO	Yes
	UPDATE	Yes
	CREATE TABLE AS	Yes
	Webhook	No
	PRIMARY KEY	Yes
	UNIQUE CONSTRAINT	Yes
	UNLOG tables	Yes
	Custom types	No
Explicit cursors	Yes	
Transaction capability	Sub-transactions	Yes
	Transaction Isolation Levels	Yes

Type	Syntax	Supported
Advanced functions	Materialized views	No
	Stored procedures	No
	AUTO VACUUM	Yes
	AUTO ANALYZE	Yes
	GIS	No

Table 3-3 Data types supported by column-store tables

Category	Data Type	Description	Length
Numeric Type	smallint	Small integer, also called INT2	2
	integer	Typical choice for integer, also called INT4	4
	bigint	Big integer, also called INT8	8
	decimal	Arbitrary precision type	Variable length
	numeric	Arbitrary precision type	Variable length
	real	Single-precision floating point	4
	double precision	Double-precision floating point	8
	smallserial	Two-byte auto-incrementing integer	2
	serial	Four-byte auto-incrementing integer	4
	bigserial	Eight-byte auto-incrementing integer	8
Monetary Type	money	Currency amount	8
Character Type	character varying(n), varchar(n)	Variable-length string	Variable length
	character(n), char(n)	Fixed-length string	n

Category	Data Type	Description	Length
	character, char	Single-byte internal type	1
	text	Variable-length string	Variable length
	nvarchar2	Variable-length string	Variable length
	clob	A big text object	Variable length
Date/Time Type	timestamp with time zone	Date and time (with time zone)	8
	timestamp without time zone	Date and time	8
	date	Date and time (Oracle compatibility mode); date (other compatibility modes)	When using Oracle compatibility mode, the storage space is 8 bytes, whereas in other compatibility modes, it is 4 bytes.
	time without time zone	Time within one day.	8
	time with time zone	Time within one day (with time zone)	12
	interval	Time interval	16

4 SQL Syntax Reference

4.1 CREATE TABLE

Function

Creates a new empty table in the current database.

This table is owned by the user who executes the command. However, if the system administrator creates a table in the schema with the same name as a common user, the owner of the table is the user (not the system administrator).

Precautions

- For details about the data types supported by column-store tables, see [Table 3-3](#).
- It is recommended that the number of column-store and HDFS partitioned tables do not exceed 1000.
- The primary key constraint and unique constraint in the table must contain a distribution column.
- A system column cannot be set as a primary key in a row-store REPLICATION distributed table.
- If an error occurs during table creation, after it is fixed, the system may fail to delete the empty disk files created before the last automatic clearance. This problem seldom occurs.
- Column-store tables support the **PARTIAL CLUSTER KEY** and table-level primary key and unique constraints, but do not support table-level foreign key constraints.
- Only the NULL, NOT NULL, and DEFAULT constant values can be used as column-store table column constraints.
- Whether column-store tables support a delta table is specified by the **enable_delta** parameter. The threshold for storing data into a delta table is specified by the **deltarow_threshold** parameter.
- Multi-temperature tables support only partitioned column-store tables and depend on available OBS tablespaces.

- Multi-temperature tables support only the default tablespace **default_obs_tbs**. If you need to add an OBS tablespace, contact technical support.
- The cloud native 3.0 version is compatible with all column-store versions. When creating a table, you need to explicitly specify the value of **colversion** (**1.0**, **2.0**, or **3.0**). If **colversion** is set to **3.0**, a table in decoupled storage and computing mode is created. If **colversion** is not explicitly specified, a column-store table of version 3.0 is created by default. When creating a table in decoupled storage and computing mode, set **colversion** to **3.0** and set **orientation** to **column**.
- The tables in decoupled storage and computing mode in the cloud native 3.0 version does not support delta tables. Even if the table-level parameter **enable_delta** is enabled, data is still inserted into the primary table. Therefore, no action is performed when **vacuum deltamerge** is executed on the table.
- Tables in decoupled storage and computing mode in the cloud native 3.0 version do not support Hstore tables, hot and cold tables, or time series tables.
- Tables in decoupled storage and computing mode in the cloud native 3.0 version support only column-store tables and depend on OBS. The default OBS tablespace is **cu_obs_tbs**.
- To create a table in decoupled storage and computing mode in the cloud native 3.0 version, you must have the USAGE permission on the default schema (named **CSTORE**).
- Temporary tables cannot be created for tables in decoupled storage and computing mode in the cloud native 3.0 version. The created temporary tables are automatically converted to column-store tables whose colversion is 2.0.

Syntax

```
CREATE [ [ GLOBAL | LOCAL | VOLATILE ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE [ IF NOT EXISTS ]
table_name
  { ( { column_name data_type [ compress_mode ] [ COLLATE collation ] [ column_constraint [ ... ] ]
    | table_constraint
    | LIKE source_table [ like_option [...] ] }
    [ ... ] )
  | LIKE source_table [ like_option [...] ] }
[ WITH ( {storage_parameter = value} [, ... ] ) ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ COMPRESS | NOCOMPRESS ]
[ DISTRIBUTE BY { REPLICATION | ROUNDROBIN | { HASH ( column_name [,...] ) } } ]
[ TO { GROUP groupname | NODE ( nodename [, ... ] ) } ]
[ COMMENT [=] 'text' ];
```

- **column_constraint** is as follows:

```
[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  CHECK ( expression ) |
  DEFAULT default_expr |
  ON UPDATE on_update_expr |
  COMMENT 'text' |
  UNIQUE [ NULLS [NOT] DISTINCT | NULLS IGNORE ] index_parameters |
  PRIMARY KEY index_parameters }
[ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

- **compress_mode** of a column is as follows:

```
{ DELTA | PREFIX | DICTIONARY | NUMSTR | NOCOMPRESS }
```

- **table_constraint** is as follows:
[CONSTRAINT constraint_name]
{ CHECK (expression) |
UNIQUE [NULLS [NOT] DISTINCT | NULLS IGNORE] (column_name [, ...]) index_parameters |
PRIMARY KEY (column_name [, ...]) index_parameters |
PARTIAL CLUSTER KEY (column_name [, ...]) }
[DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE]
- **like_option** is as follows:
{ INCLUDING | EXCLUDING } { DEFAULTS | CONSTRAINTS | INDEXES | STORAGE | COMMENTS |
PARTITION | REOPTIONS | DISTRIBUTION | DROPCOLUMNS | ALL }
- **index_parameters** is as follows:
[WITH ({storage_parameter = value} [, ...])]

Parameters

- **UNLOGGED**

If this key word is specified, the created table is not a log table. Data written to unlogged tables is not written to the write-ahead log, which makes them considerably faster than ordinary tables. However, an unlogged table is automatically truncated after a crash or unclean shutdown, incurring data loss risks. The contents of an unlogged table are also not replicated to standby servers. Any indexes created on an unlogged table are not automatically logged as well.

Usage scenario: Unlogged tables do not ensure safe data. Users can back up data before using unlogged tables; for example, users should back up the data before a system upgrade.

Troubleshooting: If data is missing in the indexes of unlogged tables due to some unexpected operations such as an unclean shutdown, users should re-create the indexes with errors.

NOTICE

The unlogged table uses no primary/standby mechanism. In the case of system faults or abnormal breakpoints, data loss may occur. Therefore, the UNLOGGED table cannot be used to store basic data.

- **GLOBAL | LOCAL | VOLATILE**

When creating a temporary table, you can specify the **GLOBAL**, **LOCAL**, or **VOLATILE** before **TEMP** or **TEMPORARY**. Currently, the keywords **GLOBAL** and **LOCAL** are introduced for standard SQL compatibility. No matter whether **GLOBAL** or **LOCAL** is specified, the GaussDB(DWS) creates a **LOCAL** temporary table. If **VOLATILE** is specified, a **VOLATILE** temporary table is created.

- **TEMPORARY | TEMP**

If **TEMP** or **TEMPORARY** is specified, the created table is a temporary table. Temporary tables are automatically dropped at the end of a session, or optionally at the end of the current transaction. Therefore, apart from CN and other CN errors connected by the current session, you can still create and use temporary table in the current session. Temporary tables are created only in the current session. If a DDL statement involves operations on temporary tables, a DDL error will be generated. Therefore, you are not advised to perform operations on temporary tables in DDL statements. **TEMP** is equivalent to **TEMPORARY**.

NOTICE

- Local or volatile temporary tables are visible to the current session through schema of the **pg_temp** start. Users should not delete schema started with **pg_temp**, **pg_toast_temp**.
- If **TEMPORARY** or **TEMP** is not specified when you create a table and the schema of the specified table starts with **pg_temp_**, the table is created as a temporary table.
- Similar to common tables, all metadata of local temporary tables is stored in system catalogs. Volatile temporary tables store table structure metadata in memory, except the schema metadata. Compared with local temporary tables, volatile temporary tables have the following constraints:
 - After a CN or DN is restarted, data in its memory will be lost, and accordingly, volatile temporary tables on it will become invalid.
 - Currently, volatile temporary tables do not support table structure modification, such as **ALTER** and **GRANT**.
 - Volatile temporary tables and local temporary tables share temporary schemas. Therefore, in the same session, the **VOLATILE** temporary table and local temporary table cannot have the same name.
 - Volatile temporary table information is not stored in system catalogs. Therefore, Volatile metadata cannot be queried by running DML statements in system catalogs.
 - Volatile temporary tables support only common row-store and column-store tables. Delta tables, time series tables, and cold and hot tables are not supported.
 - Views cannot be created based on volatile temporary tables.
 - A tablespace cannot be specified when a temporary table is created. The default tablespace of a volatile temporary table is **pg_volatile**.
 - The following constraints cannot be specified when a volatile temporary table is created: **CHECK**, **UNIQUE**, **PRIMARY KEY**, **TRIGGER**, **EXCLUDE**, and **PARTIAL CLUSTER**.

IF NOT EXISTS

If **IF NOT EXISTS** is specified, a table will be created if there is no table using the specified name. If there is already a table using the specified name, no error will be reported. A message will be displayed indicating that the table already exists, and the database will skip table creation.

table_name

Specifies the name of the table to be created.

The table name can contain a maximum of 63 characters, including letters, digits, underscores (**_**), dollar signs (**\$**), and number signs (**#**). It must start with a letter or underscore (**_**).

column_name

Specifies the name of a column to be created in the new table.

The column name can contain a maximum of 63 characters, including letters, digits, underscores (**_**), dollar signs (**\$**), and number signs (**#**). It must start with a letter or underscore (**_**).

- **data_type**

Specifies the data type of the column.

 **NOTE**

In a database compatible with Teradata or MySQL syntax, if the data type of a column is set to DATE, the DATE type is returned. Otherwise, the TIMESTAMP type is returned.

- **compress_mode**

Specifies the compress option of the table, only available for row-store table. The option specifies the algorithm preferentially used by table columns.

Value range: DELTA, PREFIX, DICTIONARY, NUMSTR, NOCOMPRESS

- **COLLATE collation**

Assigns a collation to the column (which must be of a collatable data type). If no collation is specified, the default collation is used.

- **LIKE source_table [like_option ...]**

Specifies a table from which the new table automatically copies all column names, their data types, and their not-null constraints.

The new table and the source table are decoupled after creation is complete. Changes to the source table will not be applied to the new table, and it is not possible to include data of the new table in scans of the source table.

Columns and constraints copied by **LIKE** are not merged with the same name. If the same name is specified explicitly or in another **LIKE** clause, an error is reported.

- The default expressions or the **ON UPDATE** expressions are copied from the source table to the new table only if **INCLUDING DEFAULTS** is specified. The default behavior is to exclude default expressions, resulting in the copied columns in the new table having default values **NULL**.
- The **CHECK** constraints are copied from the source table to the new table only when **INCLUDING CONSTRAINTS** is specified. Other types of constraints are never copied to the new table. **NOT NULL** constraints are always copied to the new table. These rules also apply to column constraints and table constraints.
- Any indexes on the source table will not be created on the new table, unless the **INCLUDING INDEXES** clause is specified.
- **STORAGE** settings for the copied column definitions are copied only if **INCLUDING STORAGE** is specified. The default behavior is to exclude **STORAGE** settings.
- If **INCLUDING COMMENTS** is specified, comments for the copied columns, constraints, and indexes are copied. The default behavior is to exclude comments.
- If **INCLUDING PARTITION** is specified, the partition definitions of the source table are copied to the new table, and the new table no longer uses the **PARTITION BY** clause. The default behavior is to exclude partition definition of the source table.
- If **INCLUDING REOPTIONS** is specified, the storage parameter (**WITH** clause of the source table) of the source table is copied to the new table. The default behavior is to exclude partition definition of the storage parameter of the source table.

- If **INCLUDING DISTRIBUTION** is specified, the distribution information of the source table is copied to the new table, including distribution type and column, and the new table no longer use the **DISTRIBUTE BY** clause. The default behavior is to exclude distribution information of the source table.
- If **INCLUDING DROPCOLUMNS** is specified, the deleted column information in the source table is copied to the new table. By default, the deleted column information of the source table is not copied.
- **INCLUDING ALL** contains the meaning of **INCLUDING DEFAULTS**, **INCLUDING CONSTRAINTS**, **INCLUDING INDEXES**, **INCLUDING STORAGE**, **INCLUDING COMMENTS**, **INCLUDING PARTITION**, **INCLUDING RELOPTIONS**, **INCLUDING DISTRIBUTION**, and **INCLUDING DROPCOLUMNS**.
- If **EXCLUDING** is specified, the specified parameters are not included.
- For an OBS multi-temperature table, all partitions of the new table are local hot partitions after **INCLUDING PARTITION** is specified.

NOTICE

- If the source table contains a sequence with the SERIAL, BIGSERIAL, or SMALLSERIAL data type, or a column in the source table is a sequence by default and the sequence is created for this table by using **CREATE SEQUENCE... OWNED BY**, these sequences will not be copied to the new table, and another sequence specific to the new table will be created. This is different from earlier versions. To share a sequence between the source table and new table, create a shared sequence (do not use **OWNED BY**) and set a column in the source table to this sequence.
 - You are not advised to set a column in the source table to the sequence specific to another table especially when the table is distributed in specific Node Groups, because doing so may result in **CREATE TABLE ... LIKE** execution failures. In addition, doing so may cause the sequence to become invalid in the source sequence because the sequence will also be deleted from the source table when it is deleted from the table that the sequence is specific to. To share a sequence among multiple tables, you are advised to create a shared sequence for them.
-
- **WITH ({ storage_parameter = value } [, ...])**
Specifies an optional storage parameter for a table or an index.

NOTE

Using Numeric of any precision to define column, specifies precision p and scale s. When precision and scale are not specified, the input will be displayed.

The description of parameters is as follows:

- **FILLFACTOR**

The fillfactor of a table is a percentage between 10 and 100. 100 (complete packing) is the default value. When a smaller fillfactor is specified, **INSERT** operations pack table pages only to the indicated percentage. The remaining space on each page is reserved for updating rows on that page. This gives **UPDATE** a chance to place the updated

copy of a row on the same page, which is more efficient than placing it on a different page. For a table whose records are never updated, setting the fillfactor to 100 (complete packing) is the appropriate choice, but in heavily updated tables smaller fillfactors are appropriate. The parameter has no meaning for column-store tables.

Value range: 10 to 100

– **ORIENTATION**

Specifies the storage mode (row-store, column-store) for table data. This parameter cannot be modified once it is set.

Valid value:

- **ROW** indicates that table data is stored in rows.
ROW applies to OLTP service, which has many interactive transactions. An interaction involves many columns in the table. Using ROW can improve the efficiency.
- **COLUMN** indicates that the data is stored in columns.
COLUMN applies to the data warehouse service, which has a large amount of aggregation computing, and involves a few column operations.

Default value: **ROW** (row-store)

 **NOTE**

In cluster 8.1.3 and later versions, the GUC parameter **default_orientation** (default value: **row**) is added. If the storage mode is not specified when a table is created, by default, the table is created based on the value of the parameter (row, column, column enabledelta).

– **COMPRESSION**

Specifies the compression level of the table data. It determines the compression ratio and time. Generally, the higher the level of compression, the higher the ratio, the longer the time, and the lower the level of compression, the lower the ratio, the shorter the time. The actual compression ratio depends on the distribution characteristics of loading table data.

Valid value:

The valid values for column-store tables are **YES/NO** and **LOW/MIDDLE/HIGH**, and the default is **LOW**. When this parameter is set to **YES**, the compression level is **LOW** by default.

 **NOTE**

- Currently, row-store table compression is not supported.
- To determine the size of a new GaussDB(DWS) cluster, consider the size of ORC data compressed and migrated to column-store tables in GaussDB(DWS). If the compression level is low, the size of a copy is about 1.5 to 2 times that of ORC. If the compression level is high, the size of a copy is basically the same as that of ORC.
- The middle compression of column-stores uses dictionary compression. For data not suitable for dictionary compression, the file size after middle compression may be greater than that of after low compression.

GaussDB(DWS) provides the following compression algorithms:

Table 4-1 Compression algorithms for column-based storage

COMPRESSION	NUMERIC	STRING	INT
LOW	Delta compression + RLE compression	LZ4 compression	Delta compression (RLE is optional.)
MIDDLE	Delta compression + RLE compression + LZ4 compression	dict compression or LZ4 compression	Delta compression or LZ4 compression (RLE is optional)
HIGH	Delta compression + RLE compression + zlib compression	dict compression or zlib compression	Delta compression or zlib compression (RLE is optional)

- **COMPRESSLEVEL**
Specifies the compression level of the table data. It determines the compression ratio and time. This divides a compression level into sublevels, providing you with more choices for compression rate and duration. As the value becomes greater, the compression rate becomes higher and duration longer at the same compression level. The parameter is only valid for column-store tables.
Value range: 0–3.
Default value: **0**
- **MAX_BATCHROW**
Specifies the maximum of a storage unit during data loading process. The parameter is only valid for column-store tables.
Value range: 10000 to 60000
Default value: **60000**
- **PARTIAL_CLUSTER_ROWS**
Specifies the number of records to be partial cluster stored during data loading process. The parameter is only valid for column-store tables.
Value range: 600000 to 2147483647
- **enable_delta**
Specifies whether to enable delta tables in column-store tables. The parameter is only valid for column-store tables.
Default value: **off**
- **enable_hstore**
Specifies whether an H-Store table will be created (based on column-store tables). The parameter is only valid for column-store tables. This parameter is supported by version 8.2.0.100 or later clusters. Currently, cloud native 3.0 does not support this parameter.
Default value: **off**

 NOTE

If this parameter is enabled, the following GUC parameters must be set to ensure that H-Store tables are cleared.

```
autovacuum=on, autovacuum_max_workers=6,  
autovacuum_max_workers_hstore=3.
```

– **enable_disaster_cstore**

Specifies whether fine-grained DR will be enabled for column-store tables. This parameter only takes effect on column-store tables whose COLVERSION is 2.0 and cannot be set to **true** if **enable_hstore** is **true**. This parameter is supported by version 8.2.0.100 or later clusters. Currently, it is not supported by cloud native 3.0.

– **fine_disaster_table_role**

Specifies whether the fine-grained DR table will be set as a primary or secondary table. This parameter can be **true** only when the **enable_disaster_cstore** parameter has been set to **true**. Currently, cloud native 3.0 does not support this parameter.

This parameter is supported by version 8.2.0.100 or later clusters.

Valid value:

- **primary**: Specifies the primary fine-grained DR table.
- **standby**: Specifies the standby fine-grained DR table.

– **DELTAROW_THRESHOLD**

Specifies the upper limit of to-be-imported rows for triggering the data import to a delta table when data is to be imported to a column-store table. This parameter takes effect only if the **enable_delta** table parameter is set to **on**. The parameter is only valid for column-store tables.

Value range: 0 to 60000

Default value: 6000

– **COLVERSION**

Specifies the version of the column-store format. You can switch between different storage formats.

Valid value:

1.0: Each column in a column-store table is stored in a separate file. The file name is **relfilenode.C1.0**, **relfilenode.C2.0**, **relfilenode.C3.0**, or similar.

2.0: All columns of a column-store table are combined and stored in a file. The file is named **relfilenode.C1.0**.

3.0: Each column of a column-store table is stored in a file. The file is stored in the OBS file system and named **C1_fileid.0**.

Default value: **2.0** for the data warehouse 2.0 version and **3.0** for the cloud native 3.0 version

 NOTE

- OBS cold and hot tables support only the COLVERSION 2.0 format.
 - For clusters of version 8.1.0, the default value of this parameter is **1.0**. For clusters of version 8.1.1 or later, the default value of this parameter is **2.0**. If the cluster version is upgraded from 8.1.0 to 8.1.1 or later, the default value of this parameter changes from **1.0** to **2.0**.
 - When creating a column-store table, set **COLVERSION** to **2.0**. Compared with the **1.0** storage format, the performance is significantly improved:
 - The time required for creating a column-store wide table is significantly reduced.
 - In the Roach data backup scenario, the backup time is significantly reduced.
 - The build and catch up time is greatly reduced.
 - The occupied disk space decreases significantly.
 - The cloud native 3.0 version is compatible with all column-store versions. When creating a table, you need to explicitly specify the value of **colversion** (**1.0**, **2.0**, or **3.0**). If **colversion** is set to **3.0**, a table in decoupled storage and computing mode is created. If **colversion** is not explicitly specified, a column-store table of version 3.0 is created by default. When creating a table in decoupled storage and computing mode, set **colversion** to **3.0** and set **orientation** to **column**.
 - In cloud native 3.0 mode, the **colversion** of a table in decoupled storage and computing mode cannot be changed (for example, from 2.0 to 3.0) using **ALTER TABLE**.
- **analyze_mode**
Specifies the mode of table-level auto-analyze.
Valid value:
- **frozen**: disables all **ANALYZE** operations (dynamic sampling can still be triggered when no statistics are collected).
 - **backend**: allows only **ANALYZE** triggered by **AUTOVACUUM** polling.
 - **runtime**: allows only runtime **ANALYZE** triggered by the optimizer.
 - **all**: Both backend and runtime **AUTO-ANALYZE** can be triggered.
- Default value: **all**
- **SKIP_FPI_HINT**
Indicates whether to skip the hint bits operation when the full-page writes (FPW) log needs to be written during sequential scanning.
Default value: **false**

 NOTE

- If **SKIP_FPI_HINT** is set to **true** and the checkpoint operation is performed on a table, no Xlog will be generated when the table is sequentially scanned. This applies to intermediate tables that are queried less frequently, reducing the size of Xlogs and improving query performance.
- **cache_policy** (supported only in cloud native 3.0)
Specifies the cache mode of tables or partitioned tables (disks). If one of the following values is specified in the cache policy, hot cache is used. Otherwise, cold cache is used. Hot cache occupies more space than cold cache and uses more complex replacement policies.

Value range:

- **ALL**: Hot cache is used for the entire table.
- **NONE**: Cold cache is used for the entire table.
- **HPN : N** : The first N partitions in a partitioned table use hot cache. The rest of the partitions use cold cache.
- **HPL : $P1, P2, \dots$** : In a partitioned table, the specified partitions use hot cache. The rest of the partitions use cold cache.

Default value: **ALL**

 **NOTE**

- For foreign tables and non-partitioned tables, only the **ALL** and **NONE** cache policies are supported.
- Only range-partitioned and list-partitioned internal tables support HPN and HPL cache policies.
- **ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP }**

ON COMMIT determines what to do when you commit a temporary table creation operation. The three options are as follows. Currently, only **PRESERVE ROWS** and **DELETE ROWS** can be used.

 - **PRESERVE ROWS** (Default): No special action is taken at the ends of transactions. The temporary table and its table data are unchanged.
 - **DELETE ROWS**: All rows in the temporary table will be deleted at the end of each transaction block.
 - **DROP**: The temporary table will be dropped at the end of the current transaction block.
- **COMPRESS | NOCOMPRESS**

If you specify **COMPRESS** in the **CREATE TABLE** statement, the compression feature is triggered in the case of a bulk **INSERT** operation. If this feature is enabled, a scan is performed for all tuple data within the page to generate a dictionary and then the tuple data is compressed and stored. If **NOCOMPRESS** is specified, the table is not compressed.

Default value: **NOCOMPRESS**, tuple data is not compressed before storage.
- **DISTRIBUTE BY**

Specifies how the table is distributed or replicated between DNs.

Valid value:

 - **REPLICATION**: Each row in the table exists on all DNs, that is, each DN has complete table data.
 - **ROUNDROBIN**: Each row in the table is sent to each DN in turn. Therefore, data is evenly distributed on each DN. This value is supported only in 8.1.2 or later.
 - **HASH (column_name)**: Each row of the table will be placed into all the DNs based on the hash value of the specified column.

 NOTE

- When **DISTRIBUTE BY HASH (column_name)** is specified, the primary key and its unique index must contain the **column_name** column.
- When **DISTRIBUTE BY HASH (column_name)** in a referenced table is specified, the foreign key of the reference table must contain the **column_name** column.
- If **TO GROUP** is set to a replication table node group (supported in 8.1.2 or later), **DISTRIBUTE BY** must be set to **REPLICATION**. If **DISTRIBUTE BY** is not specified, the created table is automatically set as a replication table.
- The hybrid data warehouse (standalone) has only one DN. Therefore, the distribution rule is ignored and cannot be modified.

Default value: determined by the GUC parameter **default_distribution_mode**

- When **default_distribution_mode** is set to **roundrobin**, the default value of **DISTRIBUTE BY** is selected according to the following rules:
 - i. If the primary key or unique constraint is included during table creation, hash distribution is selected. The distribution column is the column corresponding to the primary key or unique constraint.
 - ii. If the primary key or unique constraint is not included during table creation, round-robin distribution is selected.
- When **default_distribution_mode** is set to **hash**, the default value of **DISTRIBUTE BY** is selected according to the following rules:
 - i. If the primary key or unique constraint is included during table creation, hash distribution is selected. The distribution column is the column corresponding to the primary key or unique constraint.
 - ii. If the primary key or unique constraint is not included during table creation but there are columns whose data types can be used as distribution columns, hash distribution is selected. The distribution column is the first column whose data type can be used as a distribution column.
 - iii. If the primary key or unique constraint is not included during table creation and no column whose data type can be used as a distribution column exists, round-robin distribution is selected.

The following data types can be used as distribution columns:

- Integer types: **TINYINT**, **SMALLINT**, **INT**, **BIGINT**, and **NUMERIC/DECIMAL**
- Character types: **CHAR**, **BPCHAR**, **VARCHAR**, **VARCHAR2**, **NVARCHAR2**, and **TEXT**
- Date/time types: **DATE**, **TIME**, **TIMETZ**, **TIMESTAMP**, **TIMESTAMPTZ**, **INTERVAL**, and **SMALLDATETIME**

 **NOTE**

When you create a table, the choices of distribution keys and partition keys have major impact on SQL query performance. Therefore, choosing proper distribution column and partition key with strategies.

- **Selecting an Appropriate Distribution Column**

In the data distributed table using Hash, an appropriate distributed array should be used to distribute and store data on multiple DNs evenly, preventing data skew (uneven data distribution across several DNs). Determine the proper distribution column based on the following principles:

1. Determine whether data is skewed.

Connect to the database and run the following statements to check the number of tuples on each DN: Replace *tablename* with the actual name of the table to be analyzed.

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

If tuple numbers vary greatly (several times or tenfold) in each DN, a data skew occurs. Change the data distribution key based on the following principles:

2. Run the ALTER TABLE statement to adjust the distribution column. The rules for selecting a distribution column are as follows:

The column value 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.

With the above principles met, you can select join conditions as distribution keys so that join tasks can be pushed down to DNs, reducing the amount of data transferred between the DNs.

3. If a proper distribution column cannot be found to make data evenly distributed on each DN, you can use the **REPLICATION** or **ROUNDROBIN** data distribution mode. The **REPLICATION** data distribution mode stores complete data on each DN. Therefore, if a table is large and no proper distribution column can be found, the **ROUNDROBIN** data distribution mode is recommended. The **ROUNDROBIN** data distribution mode is supported in 8.1.2 or later.

- **Selecting appropriate partition keys**

In range partitioning, the table is partitioned into ranges defined by a key column or set of columns, with no overlap between the ranges of values assigned to different partitions. Each range has a dedicated partition for data storage.

Modify partition keys to make the query result stored in the same or least partitions (partition pruning). Obtaining consecutive I/O to improve the query performance.

In actual services, time is used to filter query objects. Therefore, you can use time as a partition key, and change the key value based on the total data volume and single data query volume.

- **TO { GROUP groupname | NODE (nodename [, ...]) }**

TO GROUP specifies the Node Group in which the table is created. Currently, it cannot be used for HDFS tables. **TO NODE** is used for internal scale-out tools.

In logical cluster mode, if a user creates a table without specifying **TO GROUP**, the table will be created in the node group bound to the user by default. If the user (for example, an administrator or a common user) is not bound to any logical clusters, the table will be created in the logical cluster

specified by the GUC parameter **default_storage_nodegroup** by default. If **default_storage_nodegroup** is set to **installation**, tables will be created in the first logical cluster (the logical cluster with the smallest OID in **pgxc_group**).

If the node group specified by **TO GROUP** is a replication table node group, the table is created on all CNs and DN, but the replication table data is distributed only on the DN in the replication table node group.

Cloud native 3.0 supports read-only logical clusters. If a user is not bound to any read-only logical clusters but sets **TO GROUP** to a logical cluster in a table creation statement, an error will be reported during table creation. If a user bound to a read-only logical cluster creates a table, the table will be created in the logical cluster specified by the GUC parameter **default_storage_nodegroup**. If **default_storage_nodegroup** is set to **installation**, tables will be created in the first logical cluster.

- **COMMENT [=] 'text'**

The **COMMENT** clause can specify table comments during table creation.

- **CONSTRAINT constraint_name**

Specifies a name for a column or table constraint. The optional constraint clauses specify constraints that new or updated rows must satisfy for an insert or update operation to succeed.

There are two ways to define constraints:

- A column constraint is defined as part of a column definition, and it is bound to a particular column.
- A table constraint is not bound to any particular columns but can apply to more than one column.

- **NOT NULL**

Indicates that the column is not allowed to contain **NULL** values.

- **NULL**

The column is allowed to contain **NULL** values. This is the default setting.

This clause is only provided for compatibility with non-standard SQL databases. You are advised not to use this clause.

- **CHECK (expression)**

Specifies an expression producing a Boolean result which new or updated rows must satisfy for an insert or update operation to succeed. Expressions evaluating to **TRUE** or **UNKNOWN** succeed. If any row of an insert or update operation produces a **FALSE** result, an error exception is raised and the insert or update does not alter the database.

A check constraint specified as a column constraint should reference only the column's values, while an expression appearing in a table constraint can reference multiple columns.

 **NOTE**

<>**NULL** and **!=NULL** are invalid in an expression. Change them to **IS NOT NULL**.

- **DEFAULT default_expr**

Assigns a default data value for a column. The value can be any variable-free expressions (Subqueries and cross-references to other columns in the current table are not allowed). The data type of the default expression must match the data type of the column.

The default expression will be used in any insert operation that does not specify a value for the column. If there is no default value for a column, then the default value is **NULL**.

- **ON UPDATE on_update_expr**

The **ON UPDATE** clause specifies a timestamp function for a column. Ensure that the data type of the column for which the **ON UPDATE** clause specifies a timestamp function is timestamp or timestampz.

When an SQL statement containing the **UPDATE** operation is executed, this column is automatically updated to the time specified by the timestamp function.

 **NOTE**

The **on_update_expr** function supports only CURRENT_TIMESTAMP, CURRENT_TIME, CURRENT_DATE, LOCALTIME, LOCALTIMESTAMP.

- **COMMENT 'text'**

The **COMMENT** clause can specify a comment for a column.

- **UNIQUE [NULLS [NOT] DISTINCT | NULLS IGNORE] index_parameters**

UNIQUE [NULLS [NOT] DISTINCT | NULLS IGNORE] (column_name [, ...]) index_parameters

Specifies that a group of one or more columns of a table can contain only unique values.

The [**NULLS [NOT] DISTINCT | NULLS IGNORE**] field is used to specify how to process null values in the index column of the Unique index.

Default value: This parameter is left empty by default. NULL values can be inserted repeatedly.

When the inserted data is compared with the original data in the table, the NULL value can be processed in any of the following ways:

- **NULLS DISTINCT:** NULL values are unequal and can be inserted repeatedly.
- **NULLS NOT DISTINCT:** NULL values are equal. If all index columns are NULL, NULL values cannot be inserted repeatedly. If some index columns are NULL, data can be inserted only when non-null values are different.
- **NULLS IGNORE:** NULL values are skipped during the equivalent comparison. If all index columns are NULL, NULL values can be inserted repeatedly. If some index columns are NULL, data can be inserted only when non-null values are different.

The following table lists the behaviors of the three processing modes.

Table 4-2 Processing of NULL values in index columns in unique indexes

Constraint	All Index Columns Are NULL	Some Index Columns Are NULL.
NULLS DISTINCT	Can be inserted repeatedly.	Can be inserted repeatedly.

Constraint	All Index Columns Are NULL	Some Index Columns Are NULL.
NULLS NOT DISTINCT	Cannot be inserted repeatedly.	Cannot be inserted if the non-null values are equal. Can be inserted if the non-null values are not equal.
NULLS IGNORE	Can be inserted repeatedly.	Cannot be inserted if the non-null values are equal. Can be inserted if the non-null values are not equal.

 **NOTE**

If **DISTRIBUTE BY REPLICATION** is not specified, the column table that contains only unique values must contain distribution columns.

- **PRIMARY KEY index_parameters**

PRIMARY KEY (column_name [, ...]) index_parameters

Specifies the primary key constraint specifies that a column or columns of a table can contain only unique (non-duplicate) and non-null values.

Only one primary key can be specified for a table.

 **NOTE**

If **DISTRIBUTE BY REPLICATION** is not specified, the column set with a primary key constraint must contain distributed columns.

- **DEFERRABLE | NOT DEFERRABLE**

Controls whether the constraint can be deferred. A constraint that is not deferrable will be checked immediately after every command. Checking of constraints that are deferrable can be postponed until the end of the transaction using the **SET CONSTRAINTS** command. **NOT DEFERRABLE** is the default value. Currently, only **UNIQUE** and **PRIMARY KEY** constraints of row-store tables accept this clause. All the other constraints are not deferrable.

- **PARTIAL CLUSTER KEY**

Specifies a partial cluster key for storage. When importing data to a column-store table, you can perform local data sorting by specified columns (single or multiple).

- **INITIALLY IMMEDIATE | INITIALLY DEFERRED**

If a constraint is deferrable, this clause specifies the default time to check the constraint.

- If the constraint is **INITIALLY IMMEDIATE** (default value), it is checked after each statement.
- If the constraint is **INITIALLY DEFERRED**, it is checked only at the end of the transaction.

The constraint check time can be altered using the **SET CONSTRAINTS** command.

Examples

Specify the cache policy when creating a table (supported only in cloud native 3.0).

```
CREATE TABLE Sports
(
  N_NATIONKEY INT NOT NULL
, N_NAME CHAR(25) NOT NULL
, N_REGIONKEY INT NOT NULL
, N_COMMENT VARCHAR(152)
) WITH (orientation = column, colversion = 3.0, cache_policy = 'HPL: Balls, Basketball')
tablespace cu_obs_tbs
DISTRIBUTE BY ROUNDROBIN
partition by list(N_NAME)
(
  partition Balls values ('Basketball', 'football', 'badminton'),
  partition Athletics values ('High jump', 'long jump', 'javelin'),
  partition Water_Sports values ('Surfing', 'diving', 'swimming'),
  partition Shooting values ('air guns', 'Rifles', 'archery'),
  partition rest values (DEFAULT)
);
```

Define a unique column constraint for the table.

```
CREATE TABLE CUSTOMER
(
  C_CUSTKEY BIGINT NOT NULL CONSTRAINT C_CUSTKEY_pk PRIMARY KEY ,
  C_NAME VARCHAR(25) ,
  C_ADDRESS VARCHAR(40) ,
  C_NATIONKEY INT ,
  C_PHONE CHAR(15) ,
  C_ACCTBAL DECIMAL(15,2)
)
DISTRIBUTE BY HASH(C_CUSTKEY);
```

Define a primary key table constraint for the table. You can define a primary key table constraint on one or more columns of a table.

```
CREATE TABLE CUSTOMER
(
  C_CUSTKEY BIGINT ,
  C_NAME VARCHAR(25) ,
  C_ADDRESS VARCHAR(40) ,
  C_NATIONKEY INT ,
  C_PHONE CHAR(15) ,
  C_ACCTBAL DECIMAL(15,2) ,
  CONSTRAINT C_CUSTKEY_KEY PRIMARY KEY(C_CUSTKEY,C_NAME)
)
DISTRIBUTE BY HASH(C_CUSTKEY,C_NAME);
```

Define the **CHECK** column constraint.

```
CREATE TABLE CUSTOMER
(
  C_CUSTKEY BIGINT NOT NULL CONSTRAINT C_CUSTKEY_pk PRIMARY KEY ,
  C_NAME VARCHAR(25) ,
  C_ADDRESS VARCHAR(40) ,
  C_NATIONKEY INT NOT NULL CHECK (C_NATIONKEY > 0)
)
DISTRIBUTE BY HASH(C_CUSTKEY);
```

Define the **CHECK** table constraint.

```
CREATE TABLE CUSTOMER
(
  C_CUSTKEY BIGINT NOT NULL CONSTRAINT C_CUSTKEY_pk PRIMARY KEY ,
  C_NAME VARCHAR(25) ,
  C_ADDRESS VARCHAR(40) ,
```



```
C_NATIONKEY INT
CONSTRAINT C_CUSTKEY_KEY2 CHECK(C_CUSTKEY > 0 AND C_NAME <> '')
)
DISTRIBUTE BY HASH(C_CUSTKEY);
```

Create a column-store table and specify the storage format and compression mode.

```
CREATE TABLE customer_address
(
  ca_address_sk INTEGER NOT NULL ,
  ca_address_id CHARACTER(16) NOT NULL ,
  ca_street_number CHARACTER(10) ,
  ca_street_name CHARACTER varying(60) ,
  ca_street_type CHARACTER(15) ,
  ca_suite_number CHARACTER(10)
)
WITH (ORIENTATION = COLUMN, COMPRESSION=HIGH,COLVERSION=2.0)
DISTRIBUTE BY HASH (ca_address_sk);
```

Use **DEFAULT** to declare a default value for column **W_STATE**.

```
CREATE TABLE warehouse_t
(
  W_WAREHOUSE_SK INTEGER NOT NULL,
  W_WAREHOUSE_ID CHAR(16) NOT NULL,
  W_WAREHOUSE_NAME VARCHAR(20) UNIQUE DEFERRABLE,
  W_WAREHOUSE_SQ_FT INTEGER ,
  W_COUNTY VARCHAR(30) ,
  W_STATE CHAR(2) DEFAULT 'GA',
  W_ZIP CHAR(10)
);
```

Create the **CUSTOMER_bk** table in LIKE mode.

```
CREATE TABLE CUSTOMER_bk (LIKE CUSTOMER INCLUDING ALL);
```

4.2 CREATE EXTERNAL SCHEMA

Description

Creates an EXTERNAL schema.

This syntax is used to create EXTERNAL SCHEMA to access the table created in Hive. You can use an external schema name as the prefix for access. If there is no schema name prefix, you can access the named objects in the current schema.

NOTE

Only DWS 3.0 supports the CREATE EXTERNAL SCHEMA syntax.

Important Notes

- A user who has the CREATE permission on the current database can create a foreign schema.
- When creating a named object, do not use EXTERNAL SCHEMA as the prefix. Objects cannot be created in EXTERNAL SCHEMA. Currently, only EXTERNAL SCHEMA can be used to perform SELECT, INSERT, and INSERT OVERWRITE operations on tables created in Hive.
- CREATE EXTERNAL SCHEMA does not support subcommands for creating objects in the new schema.

Syntax

- Create an external schema with a specified name.

```
CREATE EXTERNAL SCHEMA schema_name  
  WITH SOURCE source_type  
  DATABASE 'db_name'  
  SERVER srv_name  
  METAADDRESS 'address'  
  CONFIGURATION 'confpath';
```

Parameters

- **schema_name**
Name of an external schema.
Value range: a string. It must comply with the naming convention.

NOTICE

- The name must be unique,
- and cannot start with **pg_**.

- **SOURCE**

Type of the external metadata storage engine. Currently, **source_type** can only be **Hive**.

- **DATABASE**

Hive database corresponding to the external schema.

There is a many-to-one mapping between external schemas and Hive databases.

- **SERVER**

Value range: an existing foreign server.

You can associate an external schema with a foreign server to access external data.

- **METAADDRESS**

Hivemetastore communication interface.

- **CONFIGURATION**

Path for storing hivemetastore configuration files.

NOTE

If objects in the schema on the current search path are with the same name, specify the schemas different objects are in. You can run the **SHOW SEARCH_PATH** command to check the schemas on the current search path.

Examples

Create an EXTERNAL SCHEMA ex1:

```
CREATE EXTERNAL SCHEMA ex1  
  WITH SOURCE hive  
  DATABASE 'demo'  
  SERVER hdfs_server  
  METAADDRESS '*** ** ** ** **'  
  CONFIGURATION '/MRS/config'
```

Helpful Links

[ALTER EXTERNAL SCHEMA](#)

4.3 ALTER EXTERNAL SCHEMA

Function

Modifies EXTERNAL SCHEMA.

NOTE

Only DWS 3.0 supports the ALTER EXTERNAL SCHEMA syntax.

Syntax

- Modifies an external schema based on the specified name.

```
ALTER EXTERNAL SCHEMA schema_name
  WITH SOURCE source_type
  DATABASE 'db_name'
  SERVER srv_name
  METAADDRESS 'address'
  CONFIGURATION 'confpath';
```

Parameters

- **schema_name**
Name of an external schema.
Value range: a string. It must comply with the naming convention.

NOTICE

- The name must be unique,
- and cannot start with **pg_**.

-
- **SOURCE**

Type of the external metadata storage engine. Currently, **source_type** can only be **Hive**.

- **DATABASE**

Hive database corresponding to the external schema.

There is a many-to-one mapping between external schemas and Hive databases.

- **SERVER**

Value range: an existing foreign server.

You can associate an external schema with a foreign server to access external data.

- **METAADDRESS**

Hivemetastore communication interface.

- **CONFIGURATION**

Path for storing hivemetastore configuration files.

NOTE

If objects in the schema on the current search path are with the same name, specify the schemas different objects are in. You can run the **SHOW SEARCH_PATH** command to check the schemas on the current search path.

Example

Modify the database and FOREIGN SERVER corresponding to ex1.

```
ALTER EXTERNAL SCHEMA ex1
WITH DATABASE 'hms'
SERVER obs_server;
```

4.4 ALTER TABLE

Function

ALTER TABLE is used to modify tables, including modifying table definitions, renaming tables, renaming specified columns in tables, renaming table constraints, setting table schemas, enabling or disabling row-level access control, and adding or updating multiple columns.

Precautions

- Only the owner of a table, a user granted with the ALTER permission for the table, or a system administrator has the permission to run the **ALTER TABLE** statement. To change the owner or schema of a table, you must be the owner of the table or a system administrator.
- The storage parameter **ORIENTATION** cannot be modified.
- Currently, **SET SCHEMA** can only set schemas to user schemas. It cannot set a schema to a system internal schema.
- Column-store tables support **PARTIAL CLUSTER KEY** but do not support table-level foreign key constraints. In 8.1.1 or later, column-store tables support the **PRIMARY KEY** constraint and table-level **UNIQUE** constraint.
- In a column-store table, you can perform **ADD COLUMN, ALTER TYPE, SET STATISTICS, DROP COLUMN** operations. The data types of the new and modified columns should be supported by column storage. The **USING** option of **ALTER TYPE** only supports constant expression and expression involved in the column.
- The column constraints supported by column-store tables include **NULL, NOT NULL, and DEFAULT** constant values. Only the **DEFAULT** value can be modified (**SET DEFAULT** and **DROP DEFAULT**), and only the **NOT NULL** constraint can be deleted.
- The **NOT NULL** constraint and **PRIMARY KEY** constraint can be added to column-store tables. This constraint is supported by version 8.2.0 or later clusters.
- When you modify the **COLVERSION** or **enable_delta** parameter of a column-store table, other ALTER operations cannot be performed.

- Auto-increment columns cannot be added, or a column in which the **DEFAULT** value contains the `nextval()` expression cannot be added either.
- Row-level access control cannot be enabled for HDFS tables, foreign tables, and temporary tables.
- If you delete the PRIMARY KEY constraint by specifying the constraint name, the NOT NULL constraint is not deleted. You can manually delete the NOT NULL constraint as needed.
- The **cold_tablespace** and **storage_policy** parameters of **ALTER RESET** cannot be used in OBS multi-temperature tables, and **COLVERSION** cannot be changed to **1.0** for such tables.
- You can change a column-store table whose **COLVERSION** parameter is **2.0** to an OBS multi-temperature table. The **COLD_TABLESPACE** and **STORAGE_POLICY** parameters must be added.
- You can use **ALTER TABLE** to change the values of **STORAGE_POLICY** for **REOPTIONS**. After the cold/hot switchover policy is changed, the cold/hot attribute of the existing cold data will not change. The new policy takes effect for the next cold/hot switchover.
- When an **ALTER TABLE** operation is performed on a table, it triggers table rebuilding. During this rebuilding process, data is dumped into a new data file. Once the rebuilding is complete, the original file is deleted. However, it is important to note that if the table is large, the rebuilding process can consume a significant amount of disk space. When the disk space is insufficient, exercise caution when performing the **ALTER TABLE** operation on large tables to prevent the cluster from being read-only.
 - Change the data type of a column.
 - Add columns (including the oid column) to a row-store table.
 - Modify **COLVERSION** for a column-store table.
 - Specify the **DEFAULT** constant values for a column added to a column-store table, while the **DEFAULT** values contain volatile functions or the **DEFAULT** values are not **NULL** and do not belong to a specific data type.

Syntax

- **ALTER TABLE** modifies the definition of a table.
`ALTER TABLE [IF EXISTS] { table_name [*] | ONLY table_name | ONLY (table_name) }
action [, ...];`

There are several clauses of **action**:

```
column_clause  
| ADD table_constraint [ NOT VALID ]  
| ADD table_constraint_using_index  
| VALIDATE CONSTRAINT constraint_name  
| DROP CONSTRAINT [ IF EXISTS ] constraint_name [ RESTRICT | CASCADE ]  
| CLUSTER ON index_name  
| SET WITHOUT CLUSTER  
| SET ( {storage_parameter = value} [, ... ] )  
| RESET ( storage_parameter [, ... ] )  
| OWNER TO new_owner  
| SET TABLESPACE new_tablespace  
| SET {COMPRESS|NOCOMPRESS}  
| DISTRIBUTE BY { REPLICATION | ROUNDROBIN | { HASH ( column_name [,...] ) } }  
| TO { GROUP groupname | NODE ( nodename [, ... ] ) }  
| ADD NODE ( nodename [, ... ] )  
| DELETE NODE ( nodename [, ... ] )  
| DISABLE TRIGGER [ trigger_name | ALL | USER ]  
| ENABLE TRIGGER [ trigger_name | ALL | USER ]
```

```
| ENABLE REPLICA TRIGGER trigger_name  
| ENABLE ALWAYS TRIGGER trigger_name  
| DISABLE ROW LEVEL SECURITY  
| ENABLE ROW LEVEL SECURITY  
| FORCE ROW LEVEL SECURITY  
| NO FORCE ROW LEVEL SECURITY  
| REFRESH STORAGE
```

 NOTE

- **ADD table_constraint [NOT VALID]**
Adds a new table constraint.
- **ADD table_constraint_using_index**
Adds primary key constraint or unique constraint based on the unique index.
- **VALIDATE CONSTRAINT constraint_name**
Validates a foreign key or check constraint that was previously created as **NOT VALID**, by scanning the table to ensure there are no rows for which the constraint is not satisfied. Nothing happens if the constraint is already marked valid.
- **DROP CONSTRAINT [IF EXISTS] constraint_name [RESTRICT | CASCADE]**
Drops a table constraint.
- **CLUSTER ON index_name**
Selects the default index for future **CLUSTER** operations. It does not actually re-cluster the table.
- **SET WITHOUT CLUSTER**
Removes the most recently used **CLUSTER** index specification from the table. This operation affects future cluster operations that do not specify an index.
- **SET ({storage_parameter = value} [, ...])**
Changes one or more storage parameters for the table.
- **RESET (storage_parameter [, ...])**
Resets one or more storage parameters to their defaults. As with **SET**, a table rewrite might be needed to update the table entirely.
- **OWNER TO new_owner**
Changes the owner of the table, sequence, or view to the specified user.
- **SET {COMPRESS|NOCOMPRESS}**
Sets the compression feature of a table. The table compression feature affects only the storage mode of data inserted in a batch subsequently and does not affect storage of existing data. Setting the table compression feature will result in the fact that there are both compressed and uncompressed data in the table.
- **DISTRIBUTE BY { REPLICATION | ROUNDROBIN | { HASH (column_name [,...]) } }**
Changing a table's distribution mode will physically redistribute the table data based on the new distribution mode. After the distribution mode is changed, you are advised to manually run the **ANALYZE** statement to collect new statistics about the table.

 NOTE

- This operation is a major change operation, involving table distribution information modification and physical data redistribution. During the modification, services are blocked. After the modification, the original execution plan of services will change. Perform this operation according to the standard change process.
- This operation is a resource-intensive operation. If you need to modify the distribution mode of large tables, perform the operation when the computing and storage resources are sufficient. Ensure that the remaining space of the entire cluster and the tablespace where the original table is located is sufficient to store a table that has the same size as the original table and is distributed in the new distribution mode.
- **TO { GROUP groupname | NODE (nodename [, ...]) }**
The syntax is only available in extended mode (when GUC parameter **support_extended_features** is **on**). Exercise caution when enabling the mode. It is

used for tools like internal dilatation tools. Common users should not use the mode.

- **ADD NODE (nodename [, ...])**

It is only available for tools like internal dilatation. General users should not use the mode.

- **DELETE NODE (nodename [, ...])**

It is only available for internal scale-in tools. Common users should not use the syntax.

- **DISABLE TRIGGER [trigger_name | ALL | USER]**

Disables a single trigger specified by **trigger_name**, disables all triggers, or disables only user triggers (excluding internally generated constraint triggers, for example, deferrable unique constraint triggers and exclusion constraints triggers).

 **NOTE**

Exercise caution when using this function because data integrity cannot be ensured as expected if the triggers are not executed.

- **ENABLE TRIGGER [trigger_name | ALL | USER]**

Enables a single trigger specified by **trigger_name**, enables all triggers, or enables only user triggers.

- **ENABLE REPLICA TRIGGER trigger_name**

Determines that the trigger firing mechanism is affected by the configuration variable **session_replication_role**. When the replication role is **origin** (default value) or **local**, a simple trigger is fired.

When **ENABLE REPLICA** is configured for a trigger, it is fired only when the session is in **replica** mode.

- **ENABLE ALWAYS TRIGGER trigger_name**

Determines that all triggers are fired regardless of the current replication mode.

- **DISABLE/ENABLE ROW LEVEL SECURITY**

Enables or disables row-level access control for a table.

If row-level access control is enabled for a data table but no row-level access control policy is defined, the row-level access to the data table is not affected. If row-level access control for a table is disabled, the row-level access to the table is not affected even if a row-level access control policy has been defined. For details, see section **CREATE ROW LEVEL SECURITY POLICY**.

- **NO FORCE/FORCE ROW LEVEL SECURITY**

Forcibly enables or disables row-level access control for a table.

By default, the table owner is not affected by the row-level access control feature. However, if row-level access control is forcibly enabled, the table owner (excluding system administrators) will be affected. System administrators are not affected by any row-level access control policies.

- **REFRESH STORAGE**

Changes the local hot partitions that meet the criteria specified in the **storage_policy** parameter of an OBS multi-temperature table to the cold partitions stored in the OBS.

For example, if **storage_policy** is set to **'LMT:10'** for an OBS multi-temperature table when it is created, the partitions that are not updated within the last 10 days are switched to cold partitions in the OBS.

- There are several clauses of **column_clause**:

```
ADD [ COLUMN ] column_name data_type [ compress_mode ] [ COLLATE collation ]
[ column_constraint [ ... ] ]
| MODIFY [ COLUMN ] column_name data_type
| MODIFY [ COLUMN ] column_name [ CONSTRAINT constraint_name ] NOT NULL
[ ENABLE ]
```



```
| MODIFY [ COLUMN ] column_name [ CONSTRAINT constraint_name ] NULL
| MODIFY [ COLUMN ] column_name DEFAULT default_expr
| MODIFY [ COLUMN ] column_name ON UPDATE on_update_expr
| MODIFY [ COLUMN ] column_name COMMENT comment_text
| DROP [ COLUMN ] [ IF EXISTS ] column_name [ RESTRICT | CASCADE ]
| ALTER [ COLUMN ] column_name [ SET DATA ] TYPE data_type [ COLLATE collation ]
[ USING expression ]
| ALTER [ COLUMN ] column_name { SET DEFAULT expression | DROP DEFAULT }
| ALTER [ COLUMN ] column_name { SET | DROP } NOT NULL
| ALTER [ COLUMN ] column_name SET STATISTICS [PERCENT] integer
| ADD STATISTICS (( column_1_name, column_2_name [, ...] ))
| ADD { INDEX | UNIQUE [ INDEX ] } [ index_name ] ( { { column_name | ( expression ) }
[ COLLATE collation ] [ opclass ] [ ASC | DESC ] [ NULLS LAST ] } [, ...] ) [ USING method ]
[ NULLS [ NOT ] DISTINCT | NULLS IGNORE ] [ COMMENT 'text' ] LOCAL [ ( { PARTITION
index_partition_name } [, ...] ) ] [ WITH ( { storage_parameter = value } [, ...] ) ]
| ADD { INDEX | UNIQUE [ INDEX ] } [ index_name ] ( { { column_name | ( expression ) }
[ COLLATE collation ] [ opclass ] [ ASC | DESC ] [ NULLS { FIRST | LAST } ] } [, ...] ) [ USING
method ] [ NULLS [ NOT ] DISTINCT | NULLS IGNORE ] [ COMMENT 'text' ] [ WITH
( { storage_parameter = value } [, ...] ) ] [ WHERE predicate ]
| DROP { INDEX | KEY } index_name
| CHANGE [ COLUMN ] old_column_name new_column_name data_type [ [ CONSTRAINT
constraint_name ] NOT NULL [ ENABLE ] ]
[ CONSTRAINT constraint_name ] NULL | DEFAULT default_expr | COMMENT 'text' ]
| DELETE STATISTICS (( column_1_name, column_2_name [, ...] ))
| ALTER [ COLUMN ] column_name SET ( {attribute_option = value} [, ...] )
| ALTER [ COLUMN ] column_name RESET ( attribute_option [, ...] )
| ALTER [ COLUMN ] column_name SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN }
```

 NOTE

- **ADD [COLUMN] column_name data_type [compress_mode] [COLLATE collation] [column_constraint [...]]**
Adds a column to a table. If a column is added with **ADD COLUMN**, all existing rows in the table are initialized with the column's default value (**NULL** if no **DEFAULT** clause is specified).
- **ADD ({ column_name data_type [compress_mode] } [, ...])**
Adds columns in the table.
- **MODIFY [COLUMN] column_name data_type**
Modifies the data type of an existing field in a table.
- **MODIFY [COLUMN] column_name [CONSTRAINT constraint_name] NOT NULL [ENABLE]**
Adds a NOT NULL constraint to a column of a table. Currently, this clause is unavailable to column-store tables.
- **MODIFY [COLUMN] column_name [CONSTRAINT constraint_name] NULL**
Deletes the NOT NULL constraint to a certain column in the table.
- **MODIFY [COLUMN] column_name DEFAULT default_expr**
Changes the default value of the table.
- **MODIFY [COLUMN] column_name ON UPDATE on_update_expr**
Modifies the ON UPDATE expression of a specified column in a table. The column must be of the timestamp or timestampz type. If **on_update_expr** is NULL, the **ON UPDATE** clause is deleted.
- **MODIFY [COLUMN] column_name COMMENT comment_text**
Modifies the comment of the table.
- **DROP [COLUMN] [IF EXISTS] column_name [RESTRICT | CASCADE]**
Drops a column from a table. Index and constraint related to the column are automatically dropped. If an object not belonging to the table depends on the column, **CASCADE** must be specified, such as foreign key reference and view.
The **DROP COLUMN** form does not physically remove the column, but simply makes it invisible to SQL operations. Subsequent insert and update operations in the table will store a **NULL** value for the column. Therefore, column deletion takes a short period of time but does not immediately release the table space on the disks, because the space occupied by the deleted column is not reclaimed. The space will be reclaimed when **VACUUM** is executed.
- **ALTER [COLUMN] column_name [SET DATA] TYPE data_type [COLLATE collation] [USING expression]**
Change the data type of a field in the table. Only the type conversion of the same category (between values, character strings, and time) is allowed. Indexes and simple table constraints on the column will automatically use the new data type by reparsing the originally supplied expression.
ALTER TYPE requires an entire table be rewritten. This is an advantage sometimes, because it frees up unnecessary space from a table. For example, to reclaim the space occupied by a deleted column, the fastest method is to use the command.

```
ALTER TABLE table ALTER COLUMN anycol TYPE anytype;
```


In this command, **anycol** indicates any column existing in the table and **anytype** indicates the type of the prototype of the column. **ALTER TYPE** does not change the table except that the table is forcibly rewritten. In this way, the data that is no longer used is deleted.
- **ALTER [COLUMN] column_name { SET DEFAULT expression | DROP DEFAULT }**

Sets or removes the default value for a column. The default values only apply to subsequent **INSERT** commands; they do not cause rows already in the table to change. Defaults can also be created for views, in which case they are inserted into **INSERT** statements on the view before the view's **ON INSERT** rule is applied.

- **ALTER [COLUMN] column_name { SET | DROP } NOT NULL**
Changes whether a column is marked to allow **NULL** values or to reject **NULL** values. You can only use **SET NOT NULL** when the column contains no **NULL** values.
- **ALTER [COLUMN] column_name SET STATISTICS [PERCENT] integer**
Specifies the per-column statistics-gathering target for subsequent **ANALYZE** operations. The value ranges from **0** to **10000**. Set it to **-1** to revert to using the default system statistics target.
- **ADD { INDEX | UNIQUE [INDEX] } [index_name] ({ { column_name | (expression) } [COLLATE collation] [opclass] [ASC | DESC] [NULLS LAST] } [, ...]) [USING method] [NULLS [NOT] DISTINCT | NULLS IGNORE] [COMMENT 'text'] LOCAL [({ PARTITION index_partition_name } [, ...])] [WITH ({ storage_parameter = value } [, ...])]**
Create an index for the partitioned table. For details about the parameters, see **CREATE INDEX**.
- **ADD { INDEX | UNIQUE [INDEX] } [index_name] ({ { column_name | (expression) } [COLLATE collation] [opclass] [ASC | DESC] [NULLS { FIRST | LAST }] } [, ...]) [USING method] [NULLS [NOT] DISTINCT | NULLS IGNORE] [COMMENT 'text'] [WITH ({storage_parameter = value} [, ...])] [WHERE predicate]**
Create an index on the table. For details about the parameters, see **CREATE INDEX**.
- **DROP { INDEX | KEY } index_name**
Deletes an index from a table.
- **CHANGE [COLUMN] old_column_name new_column_name data_type [[CONSTRAINT constraint_name] NOT NULL [ENABLE]] [CONSTRAINT constraint_name] NULL | DEFAULT default_expr | COMMENT 'text']**
Modifies the column information in the table, such as column names and column field information.
- **{ADD | DELETE} STATISTICS ((column_1_name, column_2_name [, ...]))**
Adds or deletes the declaration of collecting multi-column statistics to collect multi-column statistics as needed when **ANALYZE** is performed for a table or a database. The statistics about a maximum of 32 columns can be collected at a time. You are not allowed to add or delete the declaration for system tables or foreign tables
- **ALTER [COLUMN] column_name SET ({attribute_option = value} [, ...])**
ALTER [COLUMN] column_name RESET (attribute_option [, ...])
Sets or resets per-attribute options.
The attribute option parameters are **n_distinct**, **n_distinct_inherited**, and **cstore_cu_sample_ratio**. **n_distinct** specifies and fixes the statistics of a table's distinct values. **n_distinct_inherited** specifies and inherits the distinct value statistics. **cstore_cu_sample_ratio** specifies the CU ratio for **ANALYZE** on a column-store table. Currently, the **n_distinct_inherited** parameter cannot be **SET** or **RESET**.
 - **n_distinct**
Sets the distinct value statistics of the column.
Value range: -1.0 to INT_MAX

Default value: **0**, indicating that this parameter is not set.

- `n_distinct_inherited`

Sets the distinct value statistics of the column in an inherited table.

Value range: -1.0 to INT_MAX

Default value: **0**, indicating that this parameter is not set.

- `cstore_cu_sample_ratio`

Specifies the expansion multiple in the calculation of CUs to be sampled during ANALYZE on a column-store table.

Value range: 1.0-10000.0

Default value: **1.0**

- **ALTER [COLUMN] column_name SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN }**

Sets the storage mode for a column. This clause specifies whether this column is held inline or in a secondary TOAST table, and whether the data should be compressed. This statement can only be used for row-based tables. SET STORAGE only sets the strategy to be used for future table operations.

- **column_constraint** is as follows:

```
[ CONSTRAINT constraint_name ]
{ NOT NULL |
  NULL |
  CHECK ( expression ) |
  DEFAULT default_expr |
  UNIQUE [ NULLS [ NOT ] DISTINCT | NULLS IGNORE ] index_parameters |
  PRIMARY KEY index_parameters }
[ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

- **compress_mode** of a column is as follows:

```
[ DELTA | PREFIX | DICTIONARY | NUMSTR | NOCOMPRESS ]
```

- **table_constraint_using_index** used to add the primary key constraint or unique constraint based on the unique index is as follows:

```
[ CONSTRAINT constraint_name ]
{ UNIQUE | PRIMARY KEY } USING INDEX index_name
[ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

- **table_constraint** is as follows:

```
[ CONSTRAINT constraint_name ]
{ CHECK ( expression ) |
  UNIQUE [ NULLS [ NOT ] DISTINCT | NULLS IGNORE ] ( column_name [, ... ] )
index_parameters |
  PRIMARY KEY ( column_name [, ... ] ) index_parameters }
[ DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

index_parameters is as follows:

```
[ WITH ( {storage_parameter = value} [, ... ] ) ]
[ USING INDEX TABLESPACE tablespace_name ]
```

- Changes the data type of an existing column in the table. Only the type conversion of the same category (between values, strings, and time) is allowed.

```
ALTER TABLE [ IF EXISTS ] table_name
  MODIFY ( { column_name data_type [ CONSTRAINT constraint_name ] NOT NULL [ ENABLE ] |
  [ CONSTRAINT constraint_name ] NULL | DEFAULT default_expr | COMMENT 'text' } [, ... ] );
```

- Rename the table. Changing the table name does not affect the stored data. The new table name can be prefixed with the schema name of the original table. The schema name cannot be changed at the same time.

```
ALTER TABLE [ IF EXISTS ] table_name
  RENAME TO new_table_name;
ALTER TABLE [ IF EXISTS ] table_name
  RENAME TO schema.new_table_name;
```

- Rename the specified column in the table.

```
ALTER TABLE [ IF EXISTS ] { table_name [*] | ONLY table_name | ONLY ( table_name ) }  
  RENAME [ COLUMN ] column_name TO new_column_name;
```
- Rename the constraint of the table.

```
ALTER TABLE { table_name [*] | ONLY table_name | ONLY ( table_name ) }  
  RENAME CONSTRAINT constraint_name TO new_constraint_name;
```
- Set the schema of the table.

```
ALTER TABLE [ IF EXISTS ] table_name  
  SET SCHEMA new_schema;
```

NOTE



- The schema setting moves the table into another schema. Associated indexes and constraints owned by table columns are migrated as well. Currently, the schema for sequences cannot be changed. If the table has sequences, delete the sequences, and create them again or delete the ownership between the table and sequences. In this way, the table schema can be changed.
- To change the schema of a table, you must also have CREATE privilege on the new schema. To add the table as a new child of a parent table, you must own the parent table as well. To alter the owner, you must also be a direct or indirect member of the new owning role, and that role must have CREATE permission on the table's schema. These restrictions enforce that altering the owner does not do anything you could not do by dropping and recreating the table. However, a system administrator can alter ownership of any table anyway.
- All the actions except for **RENAME** and **SET SCHEMA** can be combined into a list of multiple alterations to apply in parallel. For example, it is possible to add several columns or alter the type of several columns in a single command. This is useful with large tables, since only one pass over the table need be made.
- Adding a **CHECK** or **NOT NULL** constraint requires scanning the table to verify that existing rows meet the constraint.
- Adding a column with a non-null default or changing the type of an existing column will require the entire table to be rewritten. Table rebuilding may take a significant amount of time for a large table; and will temporarily require as much as double the disk space.
- Add columns.

```
ALTER TABLE [ IF EXISTS ] table_name  
  ADD ( { column_name data_type [ compress_mode ] [ COLLATE collation ] [ column_constraint  
[ ... ] } } [, ...] );
```
- Update columns.

```
ALTER TABLE [ IF EXISTS ] table_name  
  MODIFY ( { column_name data_type | column_name [ CONSTRAINT constraint_name ] NOT NULL  
[ ENABLE ] | column_name [ CONSTRAINT constraint_name ] NULL } [, ...] );
```

Parameter Description

- **IF EXISTS**
Sends a notification instead of an error if no tables have identical names. The notification prompts that the table you are querying does not exist.
- **table_name [*] | ONLY table_name | ONLY (table_name)**
table_name is the name of table that you need to modify.
If **ONLY** is specified, only the table is modified. If **ONLY** is not specified, the table and all subtables will be modified. You can add the asterisk (*) option following the table name to specify that all subtables are scanned, which is the default operation.
- **constraint_name**
Specifies the name of an existing constraint to drop.

- **index_name**
Specifies the name of this index.
- **storage_parameter**
Specifies the name of a storage parameter.
The following options are added for partition management:
 - **PERIOD** (interval type)
Sets the period for automatically creating partitions in partition management.
For details about the value range of **PERIOD** and the restrictions on enabling this function, see "CREATE TABLE PARTITION".
 **NOTE**
 - If this parameter is not configured when you create a table, you can run the **set** statements to configure this parameter and enable automatic partition creation. If this parameter has been configured before, you can run the **set** statements to modify this parameter.
 - You can run the **reset** command to disable the automatic partition creation. However, if the automatic partition deletion is enabled, the automatic partition creation cannot be disabled.
 - **TTL** (interval type)
Set the partition expiration time for automatically deleting partitions in partition management.
For details about the TTL range and restrictions on enabling this function, see "CREATE TABLE PARTITION".
 **NOTE**
 - If this parameter is not configured when you create a table, you can run the **set** statements to configure this parameter and enable automatic partition deletion. If this parameter has been configured before, you can run the **set** statements to modify this parameter.
 - You can run the **reset** command to disable the automatic partition deletion.
- **new_owner**
Specifies the name of the new table owner.
- **new_tablespace**
Specifies the new name of the tablespace to which the table belongs.
- **column_name, column_1_name, column_2_name**
Specifies the name of a new or an existing column.
- **data_type**
Specifies the type of a new column or a new type of an existing column.
- **compress_mode**
Specifies the compress options of the table, only available for row-based tables. The clause specifies the algorithm preferentially used by the column.
- **collation**
Specifies the collation rule name of a column. The optional **COLLATE** clause specifies a collation for the new column; if omitted, the collation is the default for the new column.

- **USING expression**

A **USING** clause specifies how to compute the new column value from the old; if omitted, the default conversion is an assignment cast from old data type to new. A **USING** clause must be provided if there is no implicit or assignment cast from the old to new type.

 **NOTE**

USING in **ALTER TYPE** can specify any expression involving the old values of the row; that is, it can refer to any columns other than the one being converted. This allows very general conversions to be done with the **ALTER TYPE** syntax. Because of this flexibility, the **USING** expression is not applied to the column's default value (if any); the result might not be a constant expression as required for a default. This means that when there is no implicit or assignment cast from old to new type, **ALTER TYPE** might fail to convert the default even though a **USING** clause is supplied. In such cases, drop the default with **DROP DEFAULT**, perform the **ALTER TYPE**, and then use **SET DEFAULT** to add a suitable new default. Similar considerations apply to indexes and constraints involving the column.

- **NOT NULL | NULL**

Sets whether the column allows null values.

- **integer**

Specifies the constant value of an integer with a sign. If **PERCENT** is used, the range of **integer** is from 0 to 100.

- **attribute_option**

Specifies an attribute option.

- **PLAIN | EXTERNAL | EXTENDED | MAIN**

Specifies a column storage mode.

- **PLAIN** must be used for fixed-length values (such as integers). It must be inline and uncompressed.
- **MAIN** is for inline, compressible data.
- **EXTERNAL** is for external, uncompressed data. Use of **EXTERNAL** will make substring operations on **text** and **bytea** values run faster, at the penalty of increased storage space.
- **EXTENDED** is for external, compressed data. **EXTENDED** is the default for most data types that support non-**PLAIN** storage.

- **CHECK (expression)**

New or updated rows must satisfy for an insert or update operation to succeed. Expressions evaluating to TRUE succeed. If any row of an insert or update operation produces a FALSE result, an error exception is raised and the insert or update does not alter the database.

A check constraint specified as a column constraint should reference only the column's values, while an expression appearing in a table constraint can reference multiple columns.

Currently, **CHECK** expression does not include subqueries and cannot use variables apart from the current column.

- **DEFAULT default_expr**

Assigns a default data value for a column.

The data type of the default expression must match the data type of the column.

The default expression will be used in any insert operation that does not specify a value for the column. If there is no default value for a column, then the default value is **NULL**.

If a suffix operator, such as (!), is used in **default_expr**, enclose the operator in parentheses.

- **UNIQUE [NULLS [NOT] DISTINCT | NULLS IGNORE] index_parameters**
UNIQUE (column_name [, ...]) [NULLS [NOT] DISTINCT | NULLS IGNORE] index_parameters

The **UNIQUE** constraint specifies that a group of one or more columns of a table can contain only unique values.

The [**NULLS [NOT] DISTINCT | NULLS IGNORE**] field is used to specify how to process null values in the index column of the Unique index.

Default value: This parameter is left empty by default. NULL values can be inserted repeatedly.

When the inserted data is compared with the original data in the table, the NULL value can be processed in any of the following ways:

- **NULLS DISTINCT:** NULL values are unequal and can be inserted repeatedly.
- **NULLS NOT DISTINCT:** NULL values are equal. If all index columns are NULL, NULL values cannot be inserted repeatedly. If some index columns are NULL, data can be inserted only when non-null values are different.
- **NULLS IGNORE:** NULL values are skipped during the equivalent comparison. If all index columns are NULL, NULL values can be inserted repeatedly. If some index columns are NULL, data can be inserted only when non-null values are different.

The following table lists the behaviors of the three processing modes.

Table 4-3 Processing of NULL values in index columns in unique indexes

Constraint	All Index Columns Are NULL	Some Index Columns Are NULL.
NULLS DISTINCT	Can be inserted repeatedly.	Can be inserted repeatedly.
NULLS NOT DISTINCT	Cannot be inserted repeatedly.	Cannot be inserted if the non-null values are equal. Can be inserted if the non-null values are not equal.
NULLS IGNORE	Can be inserted repeatedly.	Cannot be inserted if the non-null values are equal. Can be inserted if the non-null values are not equal.

- **PRIMARY KEY index_parameters**
PRIMARY KEY (column_name [, ...]) index_parameters

The primary key constraint specifies that a column or columns of a table can contain only unique (non-duplicate) and non-null values.

- **DEFERRABLE | NOT DEFERRABLE | INITIALLY DEFERRED | INITIALLY IMMEDIATE**

Sets whether the constraint is deferrable. This option is unavailable to column-store tables.

 - **DEFERRABLE**: deferrable can be postponed until the end of the transaction using the **SET CONSTRAINTS** command.
 - **NOT DEFERRABLE**: checks immediately after the execution of each command.
 - **INITIALLY IMMEDIATE**: checks immediately after the execution of each statement.
 - **INITIALLY DEFERRED**: checks when the transaction ends.
- **WITH ({storage_parameter = value} [, ...])**

Specifies an optional storage parameter for a table or an index.
- **COMPRESS|NOCOMPRESS**
 - **NOCOMPRESS**: If the **NOCOMPRESS** keyword is specified, the existing compression feature of the table is not changed.
 - **COMPRESS**: If the **COMPRESS** keyword is specified, the table compression feature is triggered if tuples are inserted in a batch.
- **new_table_name**

Specifies the new table name.
- **new_column_name**

Specifies the new name of a specific column in a table.
- **new_constraint_name**

Specifies the new name of a table constraint.
- **new_schema**

Specifies the new schema name.
- **CASCADE**

Automatically drops objects that depend on the dropped column or constraint (for example, views referencing the column).
- **RESTRICT**

Refuses to drop the column or constraint if there are any dependent objects. This is the default behavior.
- **schema_name**

Specifies the schema name of a table.
- **cache_policy**

Table cache policy. It is supported only in cloud native 3.0. For details, see [cache_policy](#).

Table Operation Examples

Rename a table:

```
ALTER TABLE CUSTOMER RENAME TO CUSTOMER_t;
```

Add a new table constraint:

```
ALTER TABLE customer_address ADD PRIMARY KEY(ca_address_sk);
```

Adds primary key constraint or unique constraint based on the unique index.

Create an index **CUSTOMER_constraint1** for the table **CUSTOMER**. Then add primary key constraints, and rename the created index.

```
CREATE UNIQUE INDEX CUSTOMER_constraint1 ON CUSTOMER(C_CUSTKEY);  
ALTER TABLE CUSTOMER ADD CONSTRAINT CUSTOMER_constraint2 PRIMARY KEY USING INDEX  
CUSTOMER_constraint1;
```

Rename a table constraint:

```
ALTER TABLE CUSTOMER RENAME CONSTRAINT CUSTOMER_constraint2 TO CUSTOMER_constraint;
```

Delete a table constraint:

```
ALTER TABLE CUSTOMER DROP CONSTRAINT CUSTOMER_constraint;
```

Add a table index:

```
ALTER TABLE CUSTOMER ADD INDEX CUSTOMER_index(C_CUSTKEY);
```

Delete a table index:

```
ALTER TABLE CUSTOMER DROP INDEX CUSTOMER_index;  
ALTER TABLE CUSTOMER DROP KEY CUSTOMER_index;
```

Add a partial cluster key to a column-store table:

```
ALTER TABLE customer_address ADD CONSTRAINT customer_address_cluster PARTIAL CLUSTER  
KEY(ca_address_sk);
```

Delete a partial cluster key from a column-store table:

```
ALTER TABLE customer_address DROP CONSTRAINT customer_address_cluster;
```

Change the storage format of a column-store table:

```
ALTER TABLE customer_address SET (COLVERSION = 1.0);
```

Change the distribution mode of a table:

```
ALTER TABLE customer_address DISTRIBUTE BY REPLICATION;
```

Change the schema of a table:

```
ALTER TABLE customer_address SET SCHEMA tpcds;
```

Change the data temperature for a single table:

```
ALTER TABLE cold_hot_table REFRESH STORAGE;
```

Change a column-store partitioned table to a hot and cold table.

```
CREATE table test_1(id int,d_time date)  
WITH(ORIENTATION=COLUMN)  
DISTRIBUTE BY HASH (id)  
PARTITION BY RANGE (d_time)  
(PARTITION p1 START('2022-01-01') END('2022-01-31') EVERY(interval '1 day'))  
ALTER TABLE test_1 SET (storage_policy = 'LMT:100');
```

Modify the table cache policy (supported only in cloud native 3.0):

```
ALTER TABLE orders SET (cache_policy = 'NONE');
```

Column Operation Examples

Add a column to a table:

```
ALTER TABLE warehouse_t ADD W_GOODS_CATEGORY int;
```

Modify the column name and column information in the table:

```
ALTER TABLE warehouse_t CHANGE W_GOODS_CATEGORY W_GOODS_CATEGORY2 DECIMAL NOT NULL  
COMMENT 'W_GOODS_CATEGORY';
```

Add a primary key to a table:

```
ALTER TABLE warehouse_t ADD PRIMARY KEY(w_warehouse_name);
```

Rename a column:

```
ALTER TABLE CUSTOMER RENAME C_PHONE TO new_C_PHONE;
```

Add columns to a table:

```
ALTER TABLE CUSTOMER ADD (C_COMMENT VARCHAR(117) NOT NULL, C_COUNT int);
```

Change the data type of a column in the table and set the column constraint to **NOT NULL**:

```
ALTER TABLE CUSTOMER MODIFY C_MKTSEGMENT varchar(20) NOT NULL;
```

Add the NOT NULL constraint to a certain column in the table:

```
ALTER TABLE CUSTOMER ALTER COLUMN C_PHONE SET NOT NULL;
```

Delete a column from a table:

```
ALTER TABLE CUSTOMER DROP COLUMN C_COUNT;
```

Add an index to a column in the table:

```
ALTER TABLE customer_address MODIFY ca_address_id varchar(20) CONSTRAINT ca_address_index CHECK  
(ca_address_id > 0);
```

Add a timestamp column with the **ON UPDATE** expression to the **customer_address** table:

```
ALTER TABLE customer_address ADD COLUMN C_TIME timestamp on update current_timestamp;
```

Delete the timestamp column with the **ON UPDATE** expression from the **customer_address**:

```
ALTER TABLE customer_address MODIFY COLUMN C_TIME timestamp on update NULL;
```

5 Function

get_meta_version(Oid)

Description: Obtains the metadata version information cached in a session on a DN. The input parameter is the OID of the primary table. The output parameter is the version information of all auxiliary tables related to the primary table, including the index, partition, and primary table.

Return type: record

The following table describes return columns.

Column	Type	Description
obj_oid	Oid	Oid of the metadata object
obj_type	char	Metadata type. The options are p (partition), i (index), and r (relation).
obj_parent_oid	Oid	OID of the primary table attached to the metadata object
meta_version	Xid	Version information of the metadata object

Example:

```
SELECT * FROM get_meta_version(16972);
obj_oid | obj_type | obj_parent_oid | meta_version
-----+-----+-----+-----
16972 | r | 16972 | 267910
16952 | p | 16972 | 267910
16958 | p | 16972 | 267910
(3 row)
```

get_meta_version()

Description: Obtains all metadata version information cached in a session on a DN. The output is the version information of all primary tables and related auxiliary tables, including indexes, partitions, and primary tables.

Return type: record

The following table describes return columns.

Column	Type	Description
obj_oid	Oid	Oid of the metadata object
obj_type	char	Metadata type. The options are p (partition), i (index), and r (relation)
obj_parent_oid	Oid	OID of the primary table attached to the metadata object
meta_version	Xid	Version information of the metadata object

Example:

```
SELECT * FROM get_meta_version();
obj_oid | obj_type | obj_parent_oid | meta_version
-----+-----+-----+-----
 16972 |    r    |    16972    |    267910
 16952 |    p    |    16972    |    267910
 16958 |    p    |    16972    |    267910
(3 row)
```

pgxc_get_meta_version(schemaname, relname)

Description: Obtains the version information of specified metadata cached in sessions on all DN. The input parameters are the schema name and table name of the primary table. The output parameters are the version information of all auxiliary tables related to the primary table, including the index, partition, and primary table.

Return type: record

The following table describes return columns.

Column	Type	Description
node_name	text	DN name
obj_oid	Oid	Oid of the metadata object
obj_type	char	Metadata type. The options are p (partition), i (index), and r (relation).
obj_parent_oid	Oid	OID of the primary table attached to the metadata object
meta_version	Xid	Version information of the metadata object

Example:

```
SELECT * FROM pgxc_get_meta_version('mtc', 't1');
node_name | obj_oid | obj_type | obj_parent_oid | meta_version
-----+-----+-----+-----+-----
datanode1 | 16972 |    r    |    16972    |    267910
```

```
datanode1 | 16952 | p | 16972 | 267910
datanode1 | 16958 | p | 16972 | 267910
datanode2 | 16972 | r | 16972 | 267910
datanode2 | 16952 | p | 16972 | 267910
datanode2 | 16958 | p | 16972 | 267910
datanode3 | 16972 | r | 16972 | 267910
datanode3 | 16952 | p | 16972 | 267910
datanode3 | 16958 | p | 16972 | 267910
(9 row)
```

pgxc_get_meta_version()

Description: Obtains all metadata version information cached in sessions on all DN. The output is the version information of all auxiliary tables related to the primary table, including the index, partition, and primary table itself.

Return type: record

The following table describes return columns.

Column	Type	Description
node_name	text	DN Name
obj_oid	Oid	Oid of the metadata object
obj_type	char	Metadata type. The options are p (partition), i (index), and r (relation).
obj_parent_oid	Oid	OID of the primary table attached to the metadata object
meta_version	Xid	Version information of the metadata object

Example:

```
SELECT * FROM pgxc_get_meta_version();
node_name | obj_oid | obj_type | obj_parent_oid | meta_version
-----+-----+-----+-----+-----
datanode1 | 16972 | r | 16972 | 267910
datanode1 | 16952 | p | 16972 | 267910
datanode1 | 16958 | p | 16972 | 267910
datanode2 | 16972 | r | 16972 | 267910
datanode2 | 16952 | p | 16972 | 267910
datanode2 | 16958 | p | 16972 | 267910
datanode3 | 16972 | r | 16972 | 267910
datanode3 | 16952 | p | 16972 | 267910
datanode3 | 16958 | p | 16972 | 267910
(9 row)
```

clean_dn_metadata(int)

Description: Clears all metadata cached in DN sessions. If this parameter is set to 1, the metadata cached in all sessions is cleared. If this parameter is set to other values, the metadata cached in the current session is cleared.

Return type: int

The following table describes return columns.

Column	Type	Description
cleaned_num	int	Number of deleted metadata caches

Example:

```
ELECT * FROM clean_dn_metadata(1);
cleaned_num
-----
          2
(1 row)
```

pgxc_clean_dn_metadata(int)

Description: Clears all metadata cached in all DN sessions. If this parameter is set to 1, the metadata cached in all sessions is cleared. If this parameter is set to other values, the metadata cached in the current session is cleared.

Return type: record

The following table describes return columns.

Column	Type	Description
node_name	text	DN name
cleaned_num	int	Number of deleted metadata caches

Example:

```
SELECT * FROM pgxc_clean_dn_metadata(1);
node_name | cleaned_num
-----|-----
datanode1 |          2
datanode2 |          2
datanode3 |          2
(3 row)
```

get_global_meta_cache(int)

Description: Obtains the global cache metadata on a DN. The input parameter is the bucket number, which ranges from 0 to 511.

Return type: record

The following table describes return columns.

Column	Type	Description
bucket_idx	int	ID of the bucket where the metadata object resides
meta_seq	int	Location of the metadata object in the bucket

Column	Type	Description
db_oid	Oid	OID of the database where the metadata object is located
meta_oid	Oid	Oid of the metadata object
meta_part_num	int	Number of partitions contained in a metadata object
meta_idx_num	int	Number of indexes contained in a metadata object
meta_version	text	Version information of the metadata object

Example:

```
EXECUTE DIRECT ON (datanode5) 'SELECT * FROM get_global_meta_cache(1);
bucket_idx | meta_seq | db_oid | meta_oid | meta_part_num | meta_idx_num | meta_version
-----+-----+-----+-----+-----+-----+-----
1 | 1 | 16852 | 18552 | 0 | 0 | 18552 2394152 ,
(1 row)
```

get_global_meta_cache()

Description: Obtains the global cache metadata on a DN.

Return type: record

The following table describes return columns.

Column	Type	Description
bucket_idx	int	ID of the bucket where the metadata object resides
meta_seq	int	Location of the metadata object in the bucket
db_oid	Oid	OID of the database where the metadata object is located
meta_oid	Oid	Oid of the metadata object
meta_part_num	int	Number of partitions contained in a metadata object
meta_idx_num	int	Number of indexes contained in a metadata object
meta_version	text	Version information of the metadata object

Example:

```
EXECUTE DIRECT ON (datanode5) 'SELECT * FROM get_global_meta_cache()';
bucket_idx | meta_seq | db_oid | meta_oid | meta_part_num | meta_idx_num | meta_version
-----+-----+-----+-----+-----+-----+-----
```



```
1 | 1 | 16852 | 18552 | 0 | 0 | 18552 2394152 ,
(1 row)
```

pgxc_get_global_meta_cache()

Description: Obtains the global cache metadata on all DNs.

Return type: record

The following table describes return columns.

Column	Type	Description
node_name	text	DN name
bucket_idx	int	ID of the bucket where the metadata object resides
meta_seq	int	Location of the metadata object in the bucket
db_oid	Oid	OID of the database where the metadata object is located
meta_oid	Oid	Oid of the metadata object
meta_part_num	int	Number of partitions contained in a metadata object
meta_idx_num	int	Number of indexes contained in a metadata object
meta_version	text	Version information of the metadata object

Example:

```
SELECT * FROM pgxc_get_global_meta_cache();
node_name | bucket_idx | meta_seq | db_oid | meta_oid | meta_part_num | meta_idx_num | meta_version
-----+-----+-----+-----+-----+-----+-----+-----
datanode1 | 1 | 1 | 16852 | 18552 | 0 | 0 | 18552 2394152 ,
datanode2 | 1 | 1 | 16852 | 18552 | 0 | 0 | 18552 2394152 ,
datanode3 | 1 | 1 | 16852 | 18552 | 0 | 0 | 18552 2394152 ,
(3 row)
```

global_meta_cache_reset()

Description: Clears global cached metadata on DNs.

Return type: record

The following table describes return columns.

Column	Type	Description
meta_num	int	Number of metadata records to be deleted

Example:

```
ELECT * FROM global_meta_cache_reset();
meta_num
-----
      1
(1 row)
```

pgxc_global_meta_cache_reset()

Description: Clears the global cache metadata on all DNs.

Return type: record

The following table describes return columns.

Column	Type	Description
node_name	text	DN name
meta_num	int	Number of metadata records to be deleted

Example:

```
SELECT * FROM pgxc_global_meta_cache_reset();
node_name | meta_num
-----+-----
datanode1 |      1
datanode2 |      1
datanode3 |      1
(3 row)
```

pg_obs_file_size(scheme_name.tablename)

Description: Obtains the CU file name and size of a table or partition on OBS. This function is valid only for tables whose **colversion** is **3**.

Return type: record

The function parameter fields are as follows:

Column	Type	Description
schema_name e.tablename	regclass	schema.tablename/tablename/oid of the primary table or OID of the partitioned table. If the OID of the primary table is the same as that of the partition, you are advised to use the table name as the input parameter.

Example:

```
-- The input parameter is tablename.
select pg_obs_file_size('t2_col_part_obs');
pg_obs_file_size
-----
(C1_16777266462721.0,1024)
(C1_16777266429953.0,1024)
(C1_16777249734657.0,1024)
(C1_16777249701889.0,1024)
```

```
(4 rows)
-- The input parameter is schema.tablename.
select pg_obs_file_size('public.t2_col_part_obs');
       pg_obs_file_size
-----
(C1_16777266462721.0,1024)
(C1_16777266429953.0,1024)
(C1_16777249734657.0,1024)
(C1_16777249701889.0,1024)
(4 rows)
-- The input parameter is oid.
select pg_obs_file_size(16593);
       pg_obs_file_size
-----
(C1_16777266462721.0,1024)
(C1_16777266429953.0,1024)
(C1_16777249734657.0,1024)
(C1_16777249701889.0,1024)
(4 rows)
```

pg_obs_file_size(scheme_name.tablename,partition_name)

Description: Obtains the column-store CU file name and file size of a partitioned table on OBS. This function is valid only for column-store tables whose colversion is 3.

Return type: record

The function parameter fields are as follows:

Column	Type	Description
scheme_name.tablename	regclass	schema.tablename/tablename/oid of the primary table
partition_name	cstring	Partition table name

Example:

```
select pg_obs_file_size('public.t2_col_part_obs','p1');
       pg_obs_file_size
-----
(C1_16777266462721.0,1024)
(C1_16777266429953.0,1024)
(C1_16777249734657.0,1024)
(C1_16777249701889.0,1024)
(4 rows)
```

pg_scan_residualfiles()

Description: Scans all residual file records in the database where the current node resides. When it is executed on a CN, it scans the database of the CN and OBS for residual files. When it is executed on a DN, it scans the database of the DN for residual files. This function is a database-level function and applies only to the current database. This function cannot be executed on the standby node.

Return type: record

The following table describes return columns.

Column	Type	Description
pgscrfl	text	Local path of the metadata file that records residual file information

Example:

```
select * from pg_scan_residualfiles();
      pgscrfl
-----
pg_residualfiles/pgscrfl_meta_15842_20230912182912146379
(1 row)
```

pgxc_scan_residualfiles()

Description: Scans all nodes for the residual files of the current database. This function is a cluster-level function and can be executed only on a CN. It is related to the database where the CN is located. This function cannot be executed on the standby node.

Parameter type: none

Return type: record

The following table describes return columns.

Column	Type	Description
node_name	text	Unified name shared by the active and standby nodes
instance_id	text	Name of the node where the residual file is.
pgscrfl	text	Local path of the metadata file that records residual file information

Example:

```
select * from pgxc_scan_residualfiles();
 node_name | instance_id | pgscrfl
-----+-----+-----
 datanode1 | datanode1   | pg_residualfiles/pgscrfl_meta_15854_20231106095437555205
 coordinator1 | coordinator1 | pg_residualfiles/pgscrfl_meta_15854_20231106095438240991
(1 row)
```

pg_get_scan_residualfiles()

Description: Obtains all residual file records of the current node. This function is an instance-level function and is irrelevant to the current database. It can run on any instance. This function cannot be executed on the standby node.

Return type: record

The following table describes return columns.

Column	Type	Description
handled	bool	Whether the residual file has been handled
dbname	text	Database name
residualfile	text	Path of the residual file
size	int	Size of the residual file. The value of this parameter is 0 for residual files in the OBS path.
inode	int	Index node ID of the residual file in the file system. The index node ID of the residual file on OBS is 0.
atime	time	Last access time of the residual file. This parameter is left blank for residual files in an OBS path.
mtime	time	Last modified time of the residual file. This parameter is left blank for residual files in an OBS path.
ctime	time	Last status change time of the residual file. This parameter is left blank for residual files in an OBS path.
filepath	text	Local path of the metadata file that records residual file information
notes	text	Notes

Example:

```
select * from pg_get_scan_residualfiles();
handled | dbname |
residualfile | size | inode | atime | mtime | ctime |
filepath |
notes
-----+-----
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
-----
f | postgres | /test/obsview/cudesc_check/user1/obs.xxx.com/cu_obs_tbs/tablespace_secondary/
15854/19865 | 0 | 0 | | | | pgscrf_meta_15854_20231106095438240991 |
(1 row)
```

pgxc_get_scan_residualfiles()

Description: Obtains residual file records on all nodes. This function is a cluster-level function and can be executed only on CNs. It is irrelevant to the current database. This function cannot be executed on the standby node.

Return type: record

The following table describes return columns.

Column	Type	Description
node_name	text	Unified name shared by the active and standby nodes
instance_id	text	Name of the node where the residual file is.
handled	bool	Whether the residual file has been handled
dbname	text	Database name
residualfile	text	Path of the residual file
size	int	Size of the residual file. The value of this parameter is 0 for residual files on OBS.
inode	int	Index node ID of the residual file in the file system. The value of this parameter is 0 for residual file ins an OBS path.
atime	time	Last access time of the residual file. This parameter is left blank for residual files in an OBS path.
mtime	time	Last modified time of the residual file. This parameter is left blank for residual files in an OBS path.
ctime	time	Last status change time of the residual file. This parameter is left blank for residual files in an OBS path.
filepath	text	Local path of the metadata file that records residual file information
notes	text	Notes

Example:

```
select * from pgxc_get_scan_residualfiles();
 node_name | instance_id | handled | dbname |
 residualfile | size | inode | atime |
 mtime | ctime | filepath | notes
-----+-----+-----+-----
+-----+-----+-----+-----+
+-----+-----+-----+-----+
 datanode1 | datanode1 | f | postgres | base/
 15854/19863 | | | | | | 0 | 2939427 |
 2023-11-06 09:54:15+08 | 2023-11-06 09:54:15+08 | 2023-11-06 09:54:15+08 |
 pgscrif_meta_15854_20231106095437555205 |
 coordinator1 | coordinator1 | f | postgres | /test/obsview/cudesc_check/user1/obs.xxx.com/cu_obs_tbs/
 tablespace_secondary/15854/19865 | 0 | 0 | | | |
 pgscrif_meta_15854_20231106095438240991 |
 (2 rows)
```

pg_archive_scan_residualfiles()

Description: Archives all residual file records of the current node. This function is an instance-level function and is irrelevant to the current database. It can run on any instance. This function cannot be executed on the standby node.

Return type: record

The following table describes return columns.

Column	Type	Description
archive	text	Archived folder path Residual files in the OBS path are archived in the corresponding OBS database directory.
count	int	Number of files in the archived folder
size	int	Size of the file in the archived folder

Example:

```
select * from pg_archive_scan_residualfiles();
      archive      | count | size
-----+-----+-----
pg_residualfiles/archive/pgscr_archive_15842_20230912182934335330|    1 |    0
(1 row)
```

pgxc_archive_scan_residualfiles()

Description: Archives residual file records on all nodes. This function is a cluster-level function and can be executed only on CNs. It is irrelevant to the current database. This function cannot be executed on the standby node.

Return type: record

The following table describes return columns.

Column	Type	Description
node_name	text	Unified name shared by the active and standby nodes
instance_id	text	Name of the node where the residual file is.
archive	text	Archived folder path Residual files in the OBS path are archived in the corresponding OBS database directory.
count	int	Number of files in the archived folder
size	int	Size of the file in the archived folder

Example:

```
select * from pgxc_archive_scan_residualfiles();
 node_name | instance_id | archive      | count | size
```

```
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
datanode1 | datanode1 | pg_residualfiles/archive/pgscrf_archive_20231106103246489550 | 1 | 0
coordinator1 | coordinator1 | pg_residualfiles/archive/pgscrf_archive_20231106103246592449 | 1 | 0
(2 rows)
```

pg_rm_scan_residualfiles_archive()

Description: Deletes files from the archived file list on the current instance. This function is an instance-level function and is irrelevant to the current database. It can run on any instance. This function cannot be executed on the standby node.

Return type: record

The following table describes return columns.

Column	Type	Description
count	int	Number of deleted residual files. For residual files in the local path, the number of deleted files is counted. For residual files in the OBS path, the number of deleted table directories is counted.
size	int	Total size of local files in the deleted residual files. The value of this parameter is 0 for all residual files in the OBS path.

Example:

```
select * from pg_rm_scan_residualfiles_archive();
 count | size
-----+-----
      1 |    0
(1 row)
```

pgxc_rm_scan_residualfiles_archive()

Description: Deletes files in the archive directory from all nodes. This function is a cluster-level function and can be executed only on CNs. It is irrelevant to the current database. This function cannot be executed on the standby node.

Return type: record

The following table describes return columns.

Column	Type	Description
node_name	text	Unified name shared by the active and standby nodes
instance_id	text	Name of the node where the residual file is.

Column	Type	Description
count	int	Number of deleted residual files. For residual files in the local path, the number of deleted files is counted. For residual files in the OBS path, the number of deleted table directories is counted.
size	int	Total size of local files in the deleted residual files. The value of this parameter is 0 for all residual files in the OBS path.

Example:

```
select * from pgxc_rm_scan_residualfiles_archive();
 node_name | instance_id | count | size
-----+-----+-----+-----
 datanode1 | datanode1  | 1    | 0
 coordinator1 | coordinator1 | 1    | 0
(2 rows)
```

analyze_table(scheme_name, rel_name, sample_rate, random_rate default null, parallel_degree default null)

Description: Samples data to a temporary table in parallel, performs Full Analyze on the temporary table, and updates statistics.

Return type: record

The function parameter columns are as follows:

Column	Type	Description
scheme_name	name	Scheme name of the primary table.
rel_name	name	Primary table name
sample_rate	float8	Sampling rate percentage. The value ranges from 0 to 100. Calculation: $\min((10w / \text{total_rows of the table}) \times 100\%)$
random_seed	float8	Random seed. If it is not set, the default value 0 is used.
parallel_degree	int	Concurrency. If it is not set, the default value is 10.

Example:

```
call analyze_table('public','t1_col_obs',10,0,20);
```

pgxc_clear_disk_cache()

Description: Deletes all disk cache files.

Return type: void

Example:

```
postgres=# select pgxc_clear_disk_cache();  
pgxc_clear_disk_cache
```

(1 row)

6 System Catalogs

6.1 PG_CLASS

PG_CLASS records database objects and their relations.

Table 6-1 PG_CLASS columns

Name	Type	Description
oid	oid	Row identifier (hidden attribute; must be explicitly selected)
relname	name	Name of an object, such as a table, index, or view
relnamespace	oid	OID of the namespace that contains the relationship
reltype	oid	Data type that corresponds to this table's row type (the index is 0 because the index does not have pg_type record)
reloftype	oid	OID is of composite type. 0 indicates other types.
relowner	oid	Owner of the relationship
relam	oid	Specifies the access method used, such as B-tree and hash, if this is an index
relfilenode	oid	Name of the on-disk file of this relationship. If such file does not exist, the value is 0 .
reltablespace	oid	Tablespace in which this relationship is stored. If its value is 0 , the default tablespace in this database is used. This column is meaningless if the relationship has no on-disk file.
relpages	double precision	Size of the on-disk representation of this table in pages (of size BLCKSZ). This is only an estimate used by the optimizer.

Name	Type	Description
reltuples	double precision	Number of rows in the table. This is only an estimate used by the optimizer.
relallvisible	integer	Number of pages marked as all visible in the table. This column is used by the optimizer for optimizing SQL execution. It is updated by VACUUM , ANALYZE , and a few DDL statements such as CREATE INDEX .
reltoastrelid	oid	OID of the TOAST table associated with this table. The OID is 0 if no TOAST table exists. The TOAST table stores large columns "offline" in a secondary table.
reltoastidxid	oid	OID of the index for a TOAST table. The OID is 0 for a table other than a TOAST table.
reldeltarelid	oid	OID of a Delta table Delta tables belong to column-store tables. They store long tail data generated during data import.
reldeltaidx	oid	OID of the index for a Delta table
relcudescrelid	oid	OID of a CU description table CU description tables (Desc tables) belong to column-store tables. They control whether storage data in the HDFS table directory is visible.
relcudescidx	oid	OID of the index for a CU description table
relhasindex	boolean	Its value is true if this column is a table and has (or recently had) at least one index. It is set by CREATE INDEX but is not immediately cleared by DROP INDEX . If the VACUUM process detects that a table has no index, it clears the relhasindex column and sets the value to false .
relisshared	boolean	Its value is true if the table is shared across all databases in the cluster. Only certain system catalogs (such as pg_database) are shared.
relpersistence	"char"	<ul style="list-style-type: none">• p indicates a permanent table.• u indicates a non-log table.• t indicates a temporary table.

Name	Type	Description
relkind	"char"	<ul style="list-style-type: none">• r indicates an ordinary table.• i indicates an index.• S indicates a sequence.• v indicates a view.• c indicates the composite type.• t indicates a TOAST table.• f indicates a foreign table.
relnatts	smallint	Number of user columns in the relationship (excluding system columns) pg_attribute has the same number of rows corresponding to the user columns.
relchecks	smallint	Number of constraints on a table. For details, see PG_CONSTRAINT .
relhasoids	boolean	Its value is true if an OID is generated for each row of the relationship.
relhaspkey	boolean	Its value is true if the table has (or once had) a primary key.
relhasrules	boolean	Its value is true if the table has rules. See table PG_REWRITE to check whether it has rules.
relhastriggers	boolean	Its value is true if the table has (or once had) triggers. See PG_TRIGGER .
relhas subclass	boolean	Its value is true if the table has (or once had) any inheritance child table.
relcmprs	tinyint	Whether the compression feature is enabled for the table. Note that only batch insertion triggers compression so ordinary CRUD does not trigger compression. <ul style="list-style-type: none">• 0 indicates other tables that do not support compression (primarily system tables, on which the compression attribute cannot be modified).• 1 indicates that the compression feature of the table data is NOCOMPRESS or has no specified keyword.• 2 indicates that the compression feature of the table data is COMPRESS.
relhasclusterkey	boolean	Whether the local cluster storage is used

Name	Type	Description
relrowmovement	boolean	Whether the row migration is allowed when the partitioned table is updated <ul style="list-style-type: none"> • true indicates that the row migration is allowed. • false indicates that the row migration is not allowed.
parttype	"char"	Whether the table or index has the property of a partitioned table <ul style="list-style-type: none"> • p indicates that the table or index has the property of a partitioned table. • n indicates that the table or index does not have the property of a partitioned table. • v indicates that the table is the value partitioned table in the HDFS.
relfrozenxid	xid32	All transaction IDs before this one have been replaced with a permanent ("frozen") transaction ID in this table. This column is used to track whether the table needs to be vacuumed in order to prevent transaction ID wraparound (or to allow pg_clog to be shrunk). The value is 0 (InvalidTransactionId) if the relationship is not a table. To ensure forward compatibility, this column is reserved. The relfrozenxid64 column is added to record the information.
relacl	aclitem[]	Access permissions The command output of the query is as follows: rolename=xxxx/yyyy --Assigning privileges to a role =xxxx/yyyy --Assigning the permission to public xxxx indicates the assigned privileges, and yyyy indicates the roles that are assigned to the privileges. For details about permission descriptions, see Table 6-2 .
reloptions	text[]	Access-method-specific options, as "keyword=value" strings
relfrozenxid64	xid	All transaction IDs before this one have been replaced with a permanent ("frozen") transaction ID in this table. This column is used to track whether the table needs to be vacuumed in order to prevent transaction ID wraparound (or to allow pg_clog to be shrunk). The value is 0 (InvalidTransactionId) if the relationship is not a table.

Table 6-2 Description of privileges

Parameter	Description
r	SELECT (read)
w	UPDATE (write)
a	INSERT (insert)
d	DELETE
D	TRUNCATE
x	REFERENCES
t	TRIGGER
X	EXECUTE
U	USAGE
C	CREATE
c	CONNECT
T	TEMPORARY
A	ANALYZE ANALYSE
L	ALTER
P	DROP
v	VACUUM
arwdDxtA, vLP	ALL PRIVILEGES (used for tables)
*	Authorization options for preceding permissions

Examples

View the OID and relfilenode of a table.

```
SELECT oid,relname,relfilenode FROM pg_class WHERE relname = 'table_name';
```

Count row-store tables.

```
SELECT 'row count:'||count(1) as point FROM pg_class WHERE relkind = 'r' and oid > 16384 and  
reloptions::text not like '%column%' and reloptions::text not like '%internal_mask%';
```

Count column-store tables.

```
SELECT 'column count:'||count(1) as point FROM pg_class WHERE relkind = 'r' and oid > 16384 and  
reloptions::text like '%column%';
```

Query the comments of all tables in the database.

```
SELECT relname as tablename,obj_description(relfilenode,'pg_class') as comment FROM pg_class;
```

6.2 PG_CONSTRAINT

PG_CONSTRAINT records check, primary key, unique, and foreign key constraints on the tables.

Table 6-3 PG_CONSTRAINT columns

Name	Type	Description
conname	name	Constraint name (not necessarily unique)
connamespace	oid	OID of the namespace that contains the constraint
contype	"char"	<ul style="list-style-type: none">• c indicates check constraints.• f indicates foreign key constraints.• p indicates primary key constraints.• u indicates unique constraints.• t indicates trigger constraints.
condeferrable	boolean	Whether the constraint can be deferrable
condeferred	boolean	Whether the constraint can be deferrable by default
convalidated	boolean	Whether the constraint is valid. Currently, only foreign key and check constraints can be set to false.
conrelid	oid	Table containing this constraint. The value is 0 if it is not a table constraint.
contypid	oid	Domain containing this constraint. The value is 0 if it is not a domain constraint.
conindid	oid	ID of the index associated with the constraint
confrelid	oid	Referenced table if this constraint is a foreign key; otherwise, the value is 0 .
confupdtype	"char"	Foreign key update action code <ul style="list-style-type: none">• a indicates no action.• r indicates restriction.• c indicates cascading.• n indicates that the parameter is set to null.• d indicates that the default value is used.

Name	Type	Description
confdeltype	"char"	Foreign key deletion action code <ul style="list-style-type: none">• a indicates no action.• r indicates restriction.• c indicates cascading.• n indicates that the parameter is set to null.• d indicates that the default value is used.
confmatchtype	"char"	Foreign key match type <ul style="list-style-type: none">• f indicates full match.• p indicates partial match.• u indicates simple match (not specified).
conislocal	boolean	Whether the local constraint is defined for the relationship
coninhcount	integer	Number of direct inheritance parent tables this constraint has. When the number is not 0, the constraint cannot be deleted or renamed.
connoinherit	boolean	Whether the constraint can be inherited
consoft	boolean	Whether the column indicates an informational constraint.
conopt	boolean	Whether you can use Informational Constraint to optimize the execution plan.
conkey	smallint[]	Column list of the constrained control if this column is a table constraint
confkey	smallint[]	List of referenced columns if this column is a foreign key
confpeqop	oid[]	ID list of the equality operators for PK = FK comparisons if this column is a foreign key
conppeqop	oid[]	ID list of the equality operators for PK = PK comparisons if this column is a foreign key
conffeqop	oid[]	ID list of the equality operators for FK = FK comparisons if this column is a foreign key
conexclp	oid[]	ID list of the per-column exclusion operators if this column is an exclusion constraint
conbin	pg_node_tree	Internal representation of the expression if this column is a check constraint

Name	Type	Description
consrc	text	Human-readable representation of the expression if this column is a check constraint

NOTICE

- **consrc** is not updated when referenced objects change; for example, it will not track renaming of columns. Rather than relying on this field, it's best to use **pg_get_constraintdef()** to extract the definition of a check constraint.
- **pg_class.relchecks** must be consistent with the number of check-constraint entries in this table for each relationship.

6.3 PG_EXTERNAL_NAMESPACE

Stores EXTERNAL SCHEMA information. This system catalog is supported only by DWS 3.0.

Table 6-4 PG_EXTERNAL_NAMESPACE columns

Column	Type	Description
nspid	Oid	EXTERNAL Schema Oid
srvname	text	<i>Name of the foreign server</i>
source	text	Metadata service type
address	text	Metadata service address
database	text	Metadata server database
confpath	text	Path of the configuration file of the metadata server
ensoptions	text[]	Reserved field, which is left empty currently.
catalog	text	Metadata server catalog

Examples

Query the created **EXTERNAL SCHEMA ex1**:

```
SELECT * FROM pg_external_namespace WHERE nspid = (SELECT oid FROM pg_namespace WHERE nspname = 'ex1');
```

6.4 PG_NAMESPACE

PG_NAMESPACE records the namespaces, that is, schema-related information. The **nsptype** column is added to the cloud native data warehouse 3.0 to distinguish external schemas from common schemas. The value of external schema is **e**, and the value of common schema is **i**.

Table 6-5 PG_NAMESPACE columns

Column	Type	Description
nspname	name	Name of the namespace
nspowner	oid	Owner of the namespace
nsptimeline	bigint	Timeline when the namespace is created on the DN This column is for internal use and valid only on the DN.
nspace	aclitem[]	Access permissions. For details, see GRANT and REVOKE.
permspace	bigint	Quota of a schema's permanent tablespace
usedspace	bigint	Used size of a schema's permanent tablespace
nsptype	char	Distinguishes external schemas from common schemas.

6.5 PG_PARTITION

PG_PARTITION records all partitioned tables, table partitions, toast tables on table partitions, and index partitions in the database. Partitioned index information is not stored in the **PG_PARTITION** system catalog.

Table 6-6 PG_PARTITION columns

Name	Type	Description
relname	name	Names of the partitioned tables, table partitions, TOAST tables on table partitions, and index partitions
parttype	"char"	Object type <ul style="list-style-type: none">● r indicates a partitioned table.● p indicates a table partition.● x indicates an index partition.● t indicates a TOAST table.

Name	Type	Description
parentid	oid	OID of the partitioned table in PG_CLASS when the object is a partitioned table or table partition OID of the partitioned index when the object is an index partition
rangenum	integer	Reserved field.
intervalnum	integer	Reserved field.
partstrategy	"char"	Partition policy of the partitioned table. The following policies are supported: r indicates the range partition. v indicates the numeric partition. l : indicates the list partition.
relfilenode	oid	Physical storage locations of the table partition, index partition, and TOAST table on the table partition.
reltablespace	oid	OID of the tablespace containing the table partition, index partition, TOAST table on the table partition
relpages	double precision	Statistics: numbers of data pages of the table partition and index partition
reltuples	double precision	Statistics: numbers of tuples of the table partition and index partition
relallvisible	integer	Statistics: number of visible data pages of the table partition and index partition
reltoastrelid	oid	OID of the TOAST table corresponding to the table partition
reltoastidxid	oid	OID of the TOAST table index corresponding to the table partition
indextblid	oid	OID of the table partition corresponding to the index partition
indisusable	boolean	Whether the index partition is available
reldeltarelid	oid	OID of a Delta table
reldeltaidx	oid	OID of the index for a Delta table
relcudescrelid	oid	OID of a CU description table
relcudescidx	oid	OID of the index for a CU description table

Name	Type	Description
relfrozenxid	xid32	Frozen transaction ID To ensure forward compatibility, this column is reserved. The relfrozenxid64 column is added to record the information.
intspnum	integer	Number of tablespaces that the interval partition belongs to
partkey	int2vector	Column number of the partition key
intervaltablespace	oidvector	Tablespace that the interval partition belongs to. Interval partitions fall in the tablespaces in the round-robin manner.
interval	text[]	Interval value of the interval partition
boundaries	text[]	Upper boundary of the range partition and interval partition
transit	text[]	Transit of the interval partition
reloptions	text[]	Storage property of a partition used for collecting online scale-out information. Same as pg_class.reloptions , it is a keyword=value string.
relfrozenxid64	xid	Frozen transaction ID
boundexprs	pg_node_tree	Partition boundary expression. <ul style="list-style-type: none"> For range partitioning, it is the upper boundary expression of a partition. For list partitioning, it is a collection of partition boundary enumeration values. <p>The pg_node_tree data is not readable. You can use the expression pg_get_expr to translate the current column into readable information.</p> <pre>SELECT pg_get_expr(boundexprs, 0) FROM pg_partition WHERE relname = 'country_202201'; pg_get_expr ----- ROW(202201, 'city1'::text), ROW(202201, 'city2'::text) (1 row)</pre>

6.6 PG_REWRITE

PG_REWRITE records rewrite rules defined for tables and views.

Table 6-7 PG_REWRITE columns

Name	Type	Description
rulename	name	Rule name
ev_class	oid	Name of the table that uses the rule
ev_attr	smallint	Column this rule is for (always 0 to indicate the entire table)
ev_type	"char"	Event type for this rule: <ul style="list-style-type: none"> • 1 = SELECT • 2 = UPDATE • 3 = INSERT • 4 = DELETE
ev_enabled	"char"	Controls in which mode the rule fires <ul style="list-style-type: none"> • O: The rule fires in "origin" and "local" modes. • D: The rule is disabled. • R: The rule fires in "replica" mode. • A: The rule always fires.
is_instead	boolean	Its value is true if the rule is an INSTEAD rule.
ev_qual	pg_node_tree	Expression tree (in the form of a nodeToString() representation) for the rule's qualifying condition
ev_action	pg_node_tree	Query tree (in the form of a nodeToString() representation) for the rule's action

6.7 PG_TRIGGER

PG_TRIGGER records the trigger information.

Name	Type	Description
tgrelid	oid	OID of the table where the trigger is located.
tgname	name	Trigger name.
tgfoid	oid	Trigger OID.
tgtype	smallint	Trigger type

Name	Type	Description
tgenabled	"char"	O : The trigger fires in "origin" or "local" mode. D : The trigger is disabled. R : The trigger fires in "replica" mode. A : The trigger always fires.
tgisinternal	boolean	Internal trigger ID. If the value is true, it indicates an internal trigger.
tgconstrelid	oid	The table referenced by the integrity constraint
tgconstrindid	oid	Index of the integrity constraint
tgconstraint	oid	OID of the constraint trigger in the pg_constraint
tgdeferrable	boolean	The constraint trigger is of the DEFERRABLE type.
tginitdeferred	boolean	whether the trigger is of the INITIALLY DEFERRED type
tgnargs	smallint	Input parameters number of the trigger function
tgattr	int2vector	Column ID specified by the trigger. If no column is specified, an empty array is used.
tgargs	bytea	Parameter transferred to the trigger
tgqual	pg_node_tree	Indicates the WHEN condition of the trigger. If the WHEN condition does not exist, the value is null.

6.8 PGXC_GROUP

PGXC_GROUP records node group information. In DWS 3.0, each node group in a logical cluster is called a VW. At the storage KV layer, each VW corresponds to a vgroup.

Table 6-8 PGXC_GROUP columns

Column	Type	Description
group_name	name	Specifies the name of a node group.

Column	Type	Description
in_redistribution	"char"	Whether redistribution is required. <ul style="list-style-type: none">• n indicates that the Node Group is not redistributed.• y indicates the source Node Group in redistribution.• t indicates the destination Node Group in redistribution.
group_members	oidvector_extend	DN node OID list of in a node group
group_buckets	text	Distributed data bucket group
is_installation	boolean	Indicates whether the node group is an installation node group.
group_acl	aclitem[]	Access permissions
group_kind	"char"	Node group type. <ul style="list-style-type: none">• i indicates the installation node group, which contains all DNs.• n indicates a common non-logical cluster node group.• v indicates a logical cluster node group.• e indicates the elastic cluster node group.• r indicates a replication table node group, which can only be used to create replication tables and can contain one or more logical cluster node groups.
group_ckpt_csn	xid	CSN of the last incremental extraction performed on a node group.
vgroup_id	xid	ID of the vgroup corresponding to the node group.
vgroup_bucket_count	oid	Number of buckets in the vgroup corresponding to the node group.
group_ckpt_time	timestamp with time zone	Physical time when the last incremental extraction is performed on a node group.
apply_kv_duration	integer	Duration of incremental scanning in the last incremental extraction of a node group, in seconds.

Column	Type	Description
ckpt_duration	integer	Checkpoint duration in the last incremental extraction of a node group, in seconds.
group_flags	integer	Node group flag. Currently, only the first flag is valid. Other flags are not used in the current version. <ul style="list-style-type: none">Flag 1: If the value is 1, the node group is a read-only logical cluster. If the value is 0, the node group is a read-write logical cluster.

6.9 PGXC_NODE

PGXC_NODE records information about cluster nodes.

Table 6-9 PGXC_NODE columns

Name	Type	Description
node_name	name	Node name
node_type	"char"	Node type C: CN D: DN
node_port	integer	Port ID of the node
node_host	name	Host name or IP address of a node. (If a virtual IP address is configured, its value is a virtual IP address.)
node_port1	integer	Port number of a replication node
node_host1	name	Host name or IP address of a replication node. (If a virtual IP address is configured, its value is a virtual IP address.)
hostis_primary	boolean	Whether a switchover occurs between the primary and the standby server on the current node
nodeis_primary	boolean	Whether the current node is preferred to execute non-query operations in the replication table
nodeis_preferred	boolean	Whether the current node is preferred to execute queries in the replication table
node_id	integer	Node identifier

Name	Type	Description
sctp_port	integer	Specifies the port used by the TCP proxy communication library or SCTP communication library of the primary node to listen to the data channel.
control_port	integer	Specifies the port used by the TCP proxy communication library or SCTP communication library of the primary node to listen to the control channel.
sctp_port1	integer	Specifies the port used by the TCP proxy communication library or SCTP communication library of the standby node to listen to the data channel.
control_port1	integer	Specifies the port used by the TCP proxy communication library or SCTP communication library of the standby node to listen to the control channel.
nodeis_central	boolean	Indicates that the current node is the central node.

Examples

Query the CN and DN information of the cluster.

```
select * from pgxc_node;
 node_name | node_type | node_port | node_host | node_port1 | node_host1 | hostis_primary |
 nodeis_primary | nodeis_preferred
 | node_id | sctp_port | control_port | sctp_port1 | control_port1 | nodeis_central
-----+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+-----+-----+
dn_6001_6002 | D | 40000 | 192.**.**.*1 | 45000 | 192.**.**.*2 | t | f | f
 | 1644780306 | 40002 | 40003 | 45002 | 45003 | f
dn_6003_6004 | D | 40000 | 192.**.**.*2 | 45000 | 192.**.**.*3 | t | f | f
 | -966646068 | 40002 | 40003 | 45002 | 45003 | f
dn_6005_6006 | D | 40000 | 192.**.**.*3 | 45000 | 192.**.**.*1 | t | f | f
 | 868850011 | 40002 | 40003 | 45002 | 45003 | f
cn_5001 | C | 8000 | 192.**.**.*1 | 8000 | 192.**.**.*1 | t | f | f
 | 1120683504 | 8002 | 8003 | 0 | 0 | f
cn_5002 | C | 8000 | 192.**.**.*2 | 8000 | 192.**.**.*2 | t | f | f
 | -1736975100 | 8002 | 8003 | 0 | 0 | f
cn_5003 | C | 8000 | localhost | 8000 | localhost | t | f | f
 | -125853378 | 8002 | 8003 | 0 | 0 | t
(6 rows)
```

7 System Views

7.1 PGXC_DISK_CACHE_STATS

Records the usage of the file cache. This system view is supported only by DWS 3.0.

Table 7-1 PGXC_DISK_CACHE_STATS columns

Column	Type	Description
node_name	text	Node name
total_read	bigint	Total number of times that the disk cache is accessed
local_read	bigint	Total number of times that the disk cache reads the local disk
remote_read	bigint	Total number of times that the disk cache reads data from the remote storage.
hit_rate	numeric(5,2)	Hit ratio of the disk cache
cache_size	bigint	Total size of data stored in the disk cache (KB)
fill_rate	numeric(5,2)	Disk cache filling rate

Example

Run the following command to query the hit ratio of the disk cache on each node:

```
SELECT hit_rate FROM pgxc_disk_cache_stats;  
hit_rate
```

```
-----
56.91
56.85
NaN
NaN
NaN
NaN
(6 rows)
```

7.2 PGXC_DISK_CACHE_PATH_INFO

Records the information about the hard disk where the file cache is located. This system view is supported only by DWS 3.0.

Table 7-2 PGXC_DISK_CACHE_PATH_INFO columns

Column	Type	Description
path_name	text	Path name.
node_name	text	Name of the node to which the hard disk belongs
cache_size	bigint	Total size of cache files on the hard disk (bytes)
disk_available	bigint	Available space of the hard disk (bytes)
disk_size	bigint	Total capacity of the hard disk (bytes)
disk_use_ratio	double precision	Disk space usage

Example

Query information about the hard disk used by the file cache:

```
select * from pgxc_disk_cache_path_info order by 1;
 path_name | node_name | cache_size | disk_available | disk_size | disk_use_ratio
-----+-----+-----+-----+-----+-----
dn_6001_6002_0 | dn_6001_6002 | 19619 | 137401716736 | 160982630400 | .146481105479564
dn_6001_6002_1 | dn_6001_6002 | 35968 | 137401716736 | 160982630400 | .146481105479564
dn_6003_6004_0 | dn_6003_6004 | 27794 | 121600655360 | 160982630400 | .244634933235629
dn_6003_6004_1 | dn_6003_6004 | 26158 | 121600655360 | 160982630400 | .244634933235629
dn_6005_6006_0 | dn_6005_6006 | 24533 | 134394839040 | 160982630400 | .165159379579873
dn_6005_6006_1 | dn_6005_6006 | 31065 | 134394839040 | 160982630400 | .165159379579873
```

7.3 PGXC_DISK_CACHE_ALL_STATS

Records the usage of the file cache. This system view is supported only in cloud native 3.0.

Table 7-3 PGXC_DISK_CACHE_ALL_STATS columns

Column	Type	Description
node_name	text	Node name
total_read	bigint	Total number of times that the disk cache is accessed
local_read	bigint	Total number of times that the disk cache accesses the local disk.
remote_read	bigint	Total number of times that the disk cache accesses the remote storage.
hit_rate	numeric(5,2)	Hit ratio of the disk cache
cache_size	bigint	Total size of data stored in the disk cache (KB)
fill_rate	numeric(5,2)	Disk cache filling rate
temp_file_size	bigint	Total size of temporary/cold cache files (KB)
a1in_size	bigint	Total size of data stored in the a1in queue in the disk cache (KB)
a1out_size	bigint	Total size of data stored in the a1out queue in the disk cache (KB)
am_size	bigint	Total size of data stored in the am queue in the disk cache (KB)
a1in_fill_rate	numeric(5,2)	Filling rate of the a1in queue in the disk cache
a1out_fill_rate	numeric(5,2)	Filling rate of the a1out queue in the disk cache
am_fill_rate	numeric(5,2)	Filling rate of the am queue in the disk cache
fd	integer	Number of file descriptors that are being used by the disk cache

Example

Query the number of file descriptors used by the disk cache on each node:

```
SELECT fd FROM pgxc_disk_cache_all_stats;  
fd
```

```
-----  
1000  
1000  
0  
0  
0  
0  
(6 rows)
```

7.4 PGXC_OBS_IO_SCHEDULER_STATS

Queries the latest real-time statistics about read/write requests of the OBS I/O Scheduler. This system view is supported only by DWS 3.0.

Table 7-4 PGXC_OBS_IO_SCHEDULER_STATS columns

Column	Type	Description
node_name	text	Node
io_type	char	I/O type. <ul style="list-style-type: none">• r indicates read.• w indicates write.• s indicates file operations.
current_bps	int8	Current bandwidth rate (KB/s)
best_bps	int8	Optimal bandwidth rate reached recently (KB/s)
waiting_request_num	int	Number of queuing requests
mean_request_size	int8	Average length of recently processed requests (KB)
total_token_num	int	Total number of I/O tokens
available_token_num	int	Number of available I/O tokens
total_worker_num	int	Total number of working threads
idle_worker_num	int	Number of idle working threads

Example

Step 1 Query statistics about read requests of OBS I/O Scheduler on each node:

According to the result, this is a snapshot of the statistics at a certain time point when the current I/O scheduler reads I/Os. At this time, the bandwidth is increasing, and **current_bps** is equal to **best_bps**. Take dn_6003_6004 as an example. You can see that there are queuing requests on the current DN. The value of **total_token_num** is the same as that of **available_token_num**,

indicating that the I/O scheduler has not started to process these requests when the view is queried.

```
SELECT * FROM pgxc_obs_io_scheduler_stats WHERE io_type = 'r' ORDER BY node_name;
```

node_name	io_type	current_bps	best_bps	waiting_request_num	mean_request_size	total_token_num	available_token_num	total_worker_num	idle_worker_num
dn_6001_6002	r	26990	26990	0	215	18		16	
dn_6003_6004	r	21475	21475	10	190	30		30	
dn_6005_6006	r	12384	12384	36	133	30		27	

Step 2 Wait for a while and initiate the query again.

At this time, there is no queuing request in the queue, and **available_token_num** is equal to **total_token_num**, indicating that the IO Scheduler has processed all requests and no new request needs to be processed. However, the value of **current_bps** is not 0 because the period for collecting bps statistics is 3 seconds, and the result was generated 3 seconds ago.

```
SELECT * FROM pgxc_obs_io_scheduler_stats WHERE io_type = 'r' ORDER BY node_name;
```

node_name	io_type	current_bps	best_bps	waiting_request_num	mean_request_size	total_token_num	available_token_num	total_worker_num	idle_worker_num
dn_6001_6002	r	13228	26990	0	609	18		18	
dn_6003_6004	r	15717	21475	0	622	30		30	
dn_6005_6006	r	18041	21767	0	609	30		30	

Step 3 After a short period of time, the query result is as follows. The value of **current_bps** changes to 0.

```
SELECT * FROM pgxc_obs_io_scheduler_stats WHERE io_type = 'r' ORDER BY node_name;
```

node_name	io_type	current_bps	best_bps	waiting_request_num	mean_request_size	total_token_num	available_token_num	total_worker_num	idle_worker_num
dn_6001_6002	r	0	26990	0	609	18		18	
dn_6003_6004	r	0	21475	0	622	30		30	
dn_6005_6006	r	0	21767	0	609	30		30	

----End

7.5 PGXC_OBS_IO_SCHEDULER_PERIODIC_STATS

Collects statistics on the number of requests and flow control information of different request types (including read, write, and file operations) of OBS I/O Scheduler. This view is supported only by the cloud native data warehouse 3.0.

The first query result displays the statistics from the time when the cluster is started to the time when the query is performed. For details about the columns, see the following table.

Table 1 PGXC_OBS_IO_SCHEDULER_PERIODIC_STATS table columns

Column	Type	Description
node_name	name	Name of a CN or DN, for example, dn_6001_6002.
io_type	char	I/O type. The options are as follows: <ul style="list-style-type: none">• R (Read)• W (Write)• S (File operation)
recent_throttled_req_num	int	Number of traffic limiting times between two queries to the view
total_throttled_req_num	int	Total number of traffic limiting times
last_throttled_dur(s)	int8	Time since the last traffic limiting
waiting_req_num	int	Number of queuing requests
mean_tps	numeric(7,2)	Average TPS of the two queries to the view. TPS indicates the number of requests processed per second.
mean_req_size(KB)	int8	Average length of requests between two queries to the view. The unit is KB.
mean_req_latency(ms)	int8	Average latency of requests between two queries to the view. The unit is ms.
max_req_latency(ms)	int8	Maximum latency of requests before two queries to the view. The unit is ms.
mean_bps(KB/s)	int8	Average speed of read or write requests between two queries to the view. The unit is KB/s.
duration(s)	int	Interval between two queries to the view. The unit is seconds.

Example

Run the **SELECT * FROM pgxc_obs_io_scheduler_periodic_stats** statement to query the view content. The following is an example of the query result:

```
SELECT * FROM pgxc_obs_io_scheduler_periodic_stats;
```

node_name	io_type	recent_throttled_req_num	total_throttled_req_num	last_throttled_dur(s)	waiting_req_num	mean_tps	mean_req_size(KB)	mean_req_latency(ms)	max_req_latency(ms)	mean_bps(KB/s)	duration(s)
dn_6001_6002	S	0	0	0.00	0	0	155	0	0	0	0.00
dn_6001_6002	R	0	0	0.00	0	0	155	0	0	0	0.00
dn_6001_6002	W	0	0	0.00	0	0	155	0	0	0	0.00
cn_5001	S	0	0	.03	207	519	155	0	0	0	0.00
cn_5001	R	0	0	0.00	0	0	155	0	0	0	0.00
cn_5001	W	0	0	.01	288	288	155	0	0	0	0.00

(6 rows)

To display **0** before the decimal point in the value of **mean_tps**, execute **set behavior_compat_options='display_leading_zero'**.

Run the **select * from pgxc_obs_io_scheduler_periodic_stats** statement. The following information is displayed:

```
SELECT * FROM pgxc_obs_io_scheduler_periodic_stats;
```

node_name	io_type	recent_throttled_req_num	total_throttled_req_num	last_throttled_dur(s)	waiting_req_num	mean_tps	mean_req_size(KB)	mean_req_latency(ms)	max_req_latency(ms)	mean_bps(KB/s)	duration(s)
dn_6001_6002	S	0	0	0.36	0	132	326	0	177	0	0.00
dn_6001_6002	R	0	0	0.00	0	0	177	0	0	0	0.00
dn_6001_6002	W	0	0	0.00	0	0	177	0	0	0	0.00
cn_5001	S	0	0	0.00	0	0	177	0	0	0	0.00
cn_5001	R	0	0	0.00	0	0	177	0	0	0	0.00
cn_5001	W	0	0	0.00	0	0	177	0	0	0	0.00

8 GUC Parameters

force_read_from_rw

Parameter description: Forcibly reads data from another logical cluster (reads data from the logical cluster where the table resides).

Parameter type: user

Value range: Boolean

Default value: off

Configurable or not: Configuration is not recommended.

kv_sync_up_timeout

Parameter description: Specifies the waiting timeout period for KV synchronization.

Parameter type: user

Value range: an integer ranging from 0 to 2147483647

Default value: 10min

Configurable or not: configurable

enable_cudesc_streaming

Parameter description: Specifies whether to enable cross-logical cluster access through the cudesc streaming path (obtaining cudesc and delta table data from the logical cluster where the table resides).

Parameter type: superuser

Value range: enumerated values

- **off:** disables cudesc streaming.
- **on:** enables cudesc streaming, including read and write.
- **only_read_on:** enables cudesc streaming read.

Default value: on

Configurable or not: configurable

enable_cu_align_8k

Parameter description: Specifies whether to forcibly align CUs to 8 KB.

Parameter type: user

Value range: Boolean

Default value: off

Configurable or not: configurable

enable_cu_batch_insert

Parameter description: Specifies whether to enable batch insert for column storage.

Parameter type: user

Value range: Boolean

Default value: on

Configurable or not: configurable

enable_disk_cache

Parameter description: Specifies whether to enable file cache.

Parameter type: user

Value range: Boolean

Default value: on

Configurable or not: configurable

enable_disk_cache_recovery

Parameter description: Specifies whether the file cache can be restored when the cluster is restarted.

Parameter type: user

Value range: Boolean

Default value: on

Configurable or not: Configuration is not recommended.

disk_cache_block_size

Parameter description: Specifies the size of a single block cached in the file system, in KB.

Parameter type: postmaster

Value range: an integer ranging from 8 KB to 1 TB

Default value: 1MB

Configurable or not: configurable

disk_cache_max_size

Parameter description: Specifies the total size limit of the file system cache, in KB.

Parameter type: sighup

Value range: an integer ranging from 1 MB to 1 PB

Default value: 5GB

Configurable or not: configurable

disk_cache_max_open_fd

Parameter description: Specifies the maximum number of files that can be concurrently opened in the file system cache.

Parameter type: postmaster

Value range: an integer ranging from 0 to INT_MAX

Default value: 1000

Configurable or not: configurable

dfs_max_memory

Parameter description: Specifies the upper limit (unit: KB) of the memory used for reading and writing a foreign table.

Parameter type: user

Value range: an integer ranging from 131072 to 10485760

Default value: 256MB

Configurable: Yes

enable_aio_scheduler

Parameter description: Enables the user-mode I/O control framework. After this function is enabled, all OBS I/O requests are taken over by the user-mode I/O control framework. Also, enables asynchronous reads/writes.

Parameter type: sighup

Value range: Boolean

Default value: on

Configurable: Yes

obs_worker_pool_size

Parameter description: Specifies the maximum number of threads for the agent to perform OBS read/write operations when the user-mode I/O management and control framework is enabled.

Parameter type: postmaster

Value range: an integer ranging from 4 to 2048

Default value: 128

Configurable or not: Configuration is not recommended.

async_io_acc_max_memory

Parameter description: Queries the maximum memory (unit: KB) that can be used by the asynchronous read/write acceleration feature in a single task thread.

Parameter type: user

Value range: an integer ranging from 4096 KB to INT_MAX/2 KB

Default value: 128MB

Configurable: Yes

enable_metaversion

Parameter description: Specifies whether to enable the DN global metadata cache. After metadata is enabled on DNs, extra memory space is occupied. The memory space is controlled by **local_metacache_size** and **global_metacache_size**.

Parameter type: superuser

Value range: Boolean

Default value: off

Configurable or not: Configuration is not recommended.

local_metacache_size

Parameter description: Specifies the size of metadata cached in a local session on a DN. In extreme scenarios, if the metadata memory used by an SQL statement exceeds the value of this parameter, the SQL statement does not report an error. After the SQL statement is executed, LRU elimination is performed until the memory usage is less than the value of this parameter.

Parameter type: superuser

Value range: an integer ranging from 1024 KB to INT_MAX KB

Default value: 128MB

Configurable or not: Configuration is not recommended.

global_metacache_size

Parameter description: Specifies the size of the DN global metadata cache.

Parameter type: superuser

Value range: an integer ranging from 1024 KB to INT_MAX KB

Default value: 256MB

Configurable or not: Configuration is not recommended.

enable_metadata_partprune

Parameter description: Specifies whether to enable the metadata partition pruning function. If this parameter is enabled, DNs do not cache pruned metadata.

Parameter type: superuser

Value range: Boolean

Default value: on

Configurable or not: Configuration is not recommended.

fast_tablesize

Parameter description: Enables fast table size calculation, which may cause errors.

Parameter type: user

Value range: Boolean

Default value: off

Configurable: Yes

analyze_sample_multiplier

Parameter description: Specifies the multiplier of the ratio of the foreign table stripe sampled using **ANALYZE**. The value **0** indicates that the stripe ratio is 100%.

Parameter type: superuser

Value range: an integer ranging from 0 to 100

Default value: 3

Configurable or not: Configuration is not recommended.

enable_parallel_analyze

Parameter description: Specifies whether to use the parallel sampling mode when sampling internal and foreign tables using **ANALYZE**.

Parameter type: user

Value range: Boolean

Default value: on

Configurable: Yes

parallel_analyze_workers

Parameter description: Specifies the number of concurrent threads when the parallel **ANALYZE** sampling mode is used.

Parameter type: user

Value range: an integer ranging from 0 to 64

Default value: 10

Configurable: Yes

pgxc_node_readonly

Parameter description: Specifies whether a CN or DN is read-only.

Parameter type: superuser

Value range: Boolean

Default value: off

Configurable: No

enable_foreign_meta_shipping

Parameter description: Specifies whether to enable the delivery of foreign table metadata. If this parameter is enabled, the read cluster can read and write foreign tables.

Parameter type: user

Value range: Boolean

Default value: on

Configurable: Yes

enable_batchsort_heapsort_opt

Parameter description: Specifies whether to enable heap sorting optimization, which optimizes the **Order By...Limit...** queries.

Parameter type: user

Value range: Boolean

Default value: on

Configurable: Not recommended.

enable_batchsort_ips4o

Parameter description: Specifies whether to enable the IPS4O sorting algorithm for **Batchsortstate**.

Parameter type: user

Value range: Boolean

Default value: off

Configurable: not recommended.

enable_batchsort_new_sorting

Parameter description: Specifies whether to enable new sort optimization for **Batchsortstate**.

Parameter type: user

Value range: Boolean

Default value: on

Configurable: Not recommended.

enable_batchsort_specializations

Parameter description: Specifies whether to enable new professional optimization for **Batchsortstate**. If **enable_batchsort_new_sorting** is disabled, this parameter is invalid.

Parameter type: user

Value range: Boolean

Default value: on

Configurable: Not recommended,

force_disable_text_abbrev

Parameter description: Specifies whether to forcibly disable the Prefix Key sorting optimization.

Parameter type: user

Value range: Boolean

Default value: off

Configurable: Not recommended.

enable_insert_dop

Parameter description: Specifies whether to enable DOP during data import. If DOP is enabled, data import performance is high, but more CPU and memory resources are consumed.

Parameter type: user

Value range: Boolean

Default value: off

Configurable or not: configurable

enable_insert_ft_dop

Parameter description: Specifies whether DOP is enabled when data is exported to an OBS foreign table. If DOP is enabled, data export performance is high, but more CPU and memory resources are consumed.

Parameter type: user

Value range: Boolean

Default value: off

Configurable or not: configurable

enable_insert_ft_dop_performance

Parameter description: This parameter takes effect only when [enable_insert_ft_dop](#) is set to ON. Specifies whether to enable the high performance mode when exporting data to an OBS partitioned foreign table. If this parameter is enabled, the data export performance is high, but the memory usage increases significantly. If you can evaluate that the number of partitions in the partitioned foreign table is small and the memory resources are sufficient, you can enable this parameter. Otherwise, you are advised to disable this parameter.

Parameter type: user

Value range: Boolean

Default value: off

Configurable or not: configurable

parquet_timestamp_skip_conversion

Parameter description: Specifies whether to convert the time to the local time zone when a foreign table reads data in Parquet format and the timestamp in int96 format.

- When this parameter is set to **off**:
When an int96 timestamp is read in the parquet file, it is converted from the UTC time zone to the local time zone.
- When this parameter is set to **on**:
When an int96 timestamp is read in the parquet file, it is not converted from the UTC time zone to the local time zone.

Parameter type: user

Value range: Boolean

Default value: off

Configurable or not: configurable

parquet_enable_integer_decimal

Parameter description: Specifies the conversion rule of the decimal/numeric types when data is written to a parquet foreign table. In the decimal/numeric type definition, if the range of **precision** is specified, the parameter semantics are as follows:

- When this parameter is set to **off**:
 - $1 \leq \text{precision} < 39$: Data is written to the fixed-length array **FIXED_LEN_BYTE_ARRAY**. The format is the same as decimal in Apache Hive and Apache Impala.
 - $\text{precision} \geq 39$: Data is written to the variable-length array **BYTE_ARRAY**.
- When this parameter is set to **on**:
 - $1 \leq \text{precision} < 39$: Data is written to the Int64 type.
 - $19 \leq \text{precision} < 39$: Data is written to the fixed-length array **FIXED_LEN_BYTE_ARRAY**.
 - $\text{precision} \geq 39$: Data is written to the variable-length array **BYTE_ARRAY**.

Parameter type: user

Value range: Boolean

Default value: on

Configurable: Yes

9 Development Practices

9.1 Data Reading/Writing Across Logical Clusters

After an associated logical cluster user is created, the query or modification (including Insert, Delete, and Update) submitted by the user is calculated and executed in the associated logical cluster. If the user submits a query or modification request to a table in a different logical cluster, the optimizer generates a cross-logical cluster query or modification plan to enable the user to query the table.

Figure 9-1 Querying data across logical clusters

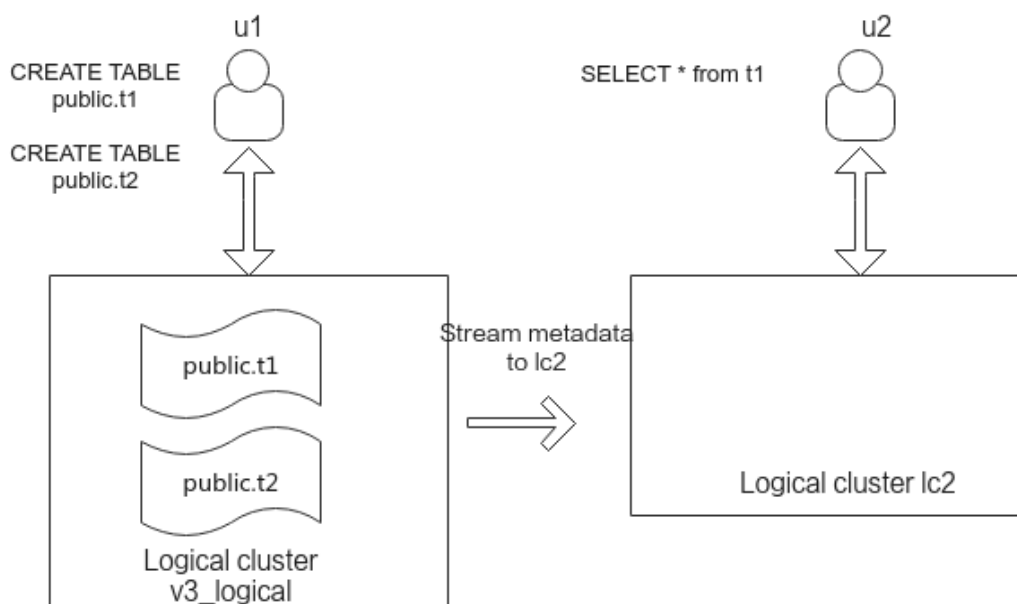
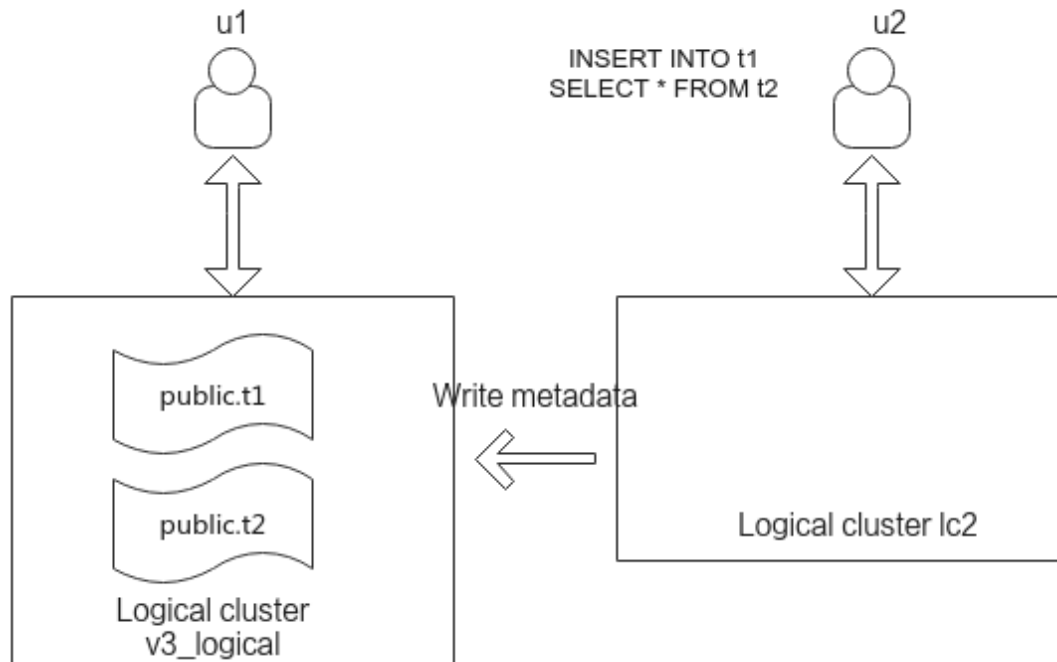


Figure 9-2 Data writing across logical clusters



Step 1 Create a GaussDB(DWS) 3.0 cluster. After a cluster is created, it is converted to a logical cluster **v3_logical** by default.

Step 2 Add three nodes to the elastic cluster, and then add the logical cluster **lc2**.

Step 3 Create user **u1** and associate it with logical cluster **v3_logical**.

```
CREATE USER u1 with SYSADMIN NODE GROUP "v3_logical" password "Password@123";
```

Step 4 Create user **u2** and associate it with logical cluster **lc2**.

```
CREATE USER u2 with SYSADMIN NODE GROUP "lc2" password "Password@123";
```

Step 5 Log in to the database as user **u1**, create tables **t1** and **t2**, and insert test data into the tables.

```
CREATE TABLE public.t1
(
  id integer not null,
  data integer,
  age integer
)
WITH (ORIENTATION = COLUMN, COLVERSION =3.0)
DISTRIBUTE BY ROUNDROBIN;

CREATE TABLE public.t2
(
  id integer not null,
  data integer,
  age integer
)
WITH (ORIENTATION = COLUMN, COLVERSION =3.0)
DISTRIBUTE BY ROUNDROBIN;

INSERT INTO public.t1 VALUES (1,2,10),(2,3,11);
INSERT INTO public.t2 VALUES (1,2,10),(2,3,11);
```

Step 6 Log in to the database as user **u2** and run the following commands to query **t1** and write data.

According to the result, user **u2** can query and write data across logical clusters.

```
SELECT * FROM t1;
INSERT INTO t1 SELECT * FROM t2;
```

----End

9.2 Data Lakehouse

9.2.1 Accessing HiveMetaStore Across Clusters

To access data stored in MRS Hive (including the scenario where Hive interconnects with HDFS and the scenario where Hive interconnects with OBS), you can refer to this tutorial to create an external schema.

Preparing the Environment

- You have created a GaussDB(DWS) 3.0 cluster, and the MRS and GaussDB(DWS) clusters are in the same region, AZ, and VPC subnet and that the clusters can communicate with each other.
- You have obtained the AK and SK.

Constraints and Limitations

- Currently, only the SELECT, INSERT, and INSERT OVERWRITE operations can be performed on tables in the Hive database through external schemas.
- MRS supports two types of data sources. For details, see [Table 9-1](#).

Table 9-1 Operations supported by the two types of MRS data sources

Data Source	Table Type	Operation	TEXT	CSV	PARQUET	ORC
HDFS	Non-partitioned table	SELECT	√	√	√	√
		INSERT/INSERT OVERWRITE	x	x	x	√
	Partitioned table	SELECT	√	√	√	√
		INSERT/INSERT OVERWRITE	x	x	x	√

Data Source	Table Type	Operation	TEXT	CSV	PARQUET	ORC
OBS	Non-partitioned table	SELECT	√	√	√	√
		INSERT/INSERT OVERWRITE	x	x	x	√
	Partitioned table	SELECT	x	x	√	√
		INSERT/INSERT OVERWRITE	x	x	x	x

- Transaction atomicity is no longer ensured. If a transaction fails, data consistency cannot be ensured. Rollback is not supported.
- GRANT and REVOKE operations cannot be performed on tables created on Hive using external schemas.
- Concurrency support: Concurrent read and write operations on GaussDB(DWS), Hive, and Spark may cause dirty reads. Concurrent operations including INSERT OVERWRITE on the same non-partitioned table or the same partition of the same partitioned table may not ensure the expected result. Therefore, do not perform such operations.
- The HiveMetaStore features do not support the federation mechanism.

Procedure

This practice takes about 1 hour. The basic process is as follows:

1. Create an MRS analysis cluster. (Select the Hive component.)
2. Create a table on Hive.
3. Insert data on Hive, or upload a local TXT data file to an OBS bucket then import the file to Hive through the OBS bucket, and import the file from the TXT storage table to the ORC storage table.
4. Create an MRS data source connection.
5. Create a foreign server.
6. Create an external schema.
7. Use the external schema to import data to or read data from Hive tables.

Creating an MRS Analysis Cluster

- Step 1** Log in to the management console, choose **Analytics > MapReduce Service**, click **Buy Cluster**, select **Custom Config**, set the configuration parameters, and click **Next**.

Table 9-2 MRS configuration

Parameter	Value
Region	Dublin
Cluster Name	mrs_01
Database Type	Normal
Cluster Version	MRS 3.1.3 (recommended) NOTE <ul style="list-style-type: none">MRS clusters support 3.0.*, 3.1.*, and later versions (* indicates a number).
Cluster Type	Analysis cluster
Metadata	Local

Step 2 Click **Next** to configure hardware parameters.

Table 9-3 Hardware configuration

Parameter	Value
Billing Mode	Pay-per-use
AZ	AZ2
VPC	vpc-01
Subnet	subnet-01
Security Group	Auto create
EIP	10.x.x.x
Enterprise Project	default
Master	2
Analysis Core	3
Analysis Task	0

Step 3 Configure the advanced settings based on the following table, click **Buy Now**, and wait for about 15 minutes for the cluster creation to complete.

Table 9-4 Advanced settings

Parameter	Value
Tags	test01
Hostname Prefix	(Optional) Prefix for the name of an ECS or BMS in the cluster.

Parameter	Value
Auto scaling	Retain the default value.
Bootstrap Action	Retain the default value. MRS 3.x does not support this parameter.
Agency	Retain the default value.
Data Disk Encryption	This function is disabled by default. Retain the default value.
Alarms	Retain the default value.
Rule Name	Retain the default value.
Topic Name	Select a topic.
Kerberos Authentication	This parameter is enabled by default.
Username	admin
Password	This password is used to log in to the cluster management page.
Confirm Password	Enter the password of user admin again.
Login Mode	Password
User	root
Password	This password is used to remotely log in to the ECS.
Confirm Password	Enter the password of user root again.
Agency	In Advanced Settings , set Agency to the preset agency MRS_ECS_DEFAULT_AGENCY of MRS in IAM.
Secure Communications	Select Enable .

----End

Preparing the ORC Table Data Source of MRS

Step 1 Create a **product_info.txt** file on the local PC, copy the following data to the file, and save the file to the local PC.

```
100,XHDK-A-1293-#fJ3,2017-09-01,A,2017 Autumn New Shirt Women,red,M,328,2017-09-04,715,good
205,KDKE-B-9947-#kL5,2017-09-01,A,2017 Autumn New Knitwear Women,pink,L,584,2017-09-05,406,very
good!
300,JODL-X-1937-#pV7,2017-09-01,A,2017 autumn new T-shirt men,red,XL,1245,2017-09-03,502,Bad.
310,QQPX-R-3956-#aD8,2017-09-02,B,2017 autumn new jacket women,red,L,411,2017-09-05,436,It's really
super nice
150,ABEF-C-1820-#mC6,2017-09-03,B,2017 Autumn New Jeans Women,blue,M,1223,2017-09-06,1200,The
seller's packaging is exquisite
200,BCQP-E-2365-#qE4,2017-09-04,B,2017 autumn new casual pants men,black,L,997,2017-09-10,301,The
clothes are of good quality.
250,EABE-D-1476-#oB1,2017-09-10,A,2017 autumn new dress women,black,S,841,2017-09-15,299,Follow
```


the store for a long time.
 108,CDXK-F-1527-#pL2,2017-09-11,A,2017 autumn new dress women,red,M,85,2017-09-14,22,It's really amazing to buy
 450,MMCE-H-4728-#nP9,2017-09-11,A,2017 autumn new jacket women,white,M,114,2017-09-14,22,Open the package and the clothes have no odor
 260,OCDA-G-2817-#bD3,2017-09-12,B,2017 autumn new woolen coat
 women,red,L,2004,2017-09-15,826,Very favorite clothes
 980,ZKDS-J-5490-#cW4,2017-09-13,B,2017 Autumn New Women's Cotton
 Clothing,red,M,112,2017-09-16,219,The clothes are small
 98,FKQB-I-2564-#dA5,2017-09-15,B,2017 autumn new shoes men,green,M,4345,2017-09-18,5473,The clothes are thick and it's better this winter.
 150,DMQY-K-6579-#eS6,2017-09-21,A,2017 autumn new underwear
 men,yellow,37,2840,2017-09-25,5831,This price is very cost effective
 200,GKLW-l-2897-#wQ7,2017-09-22,A,2017 Autumn New Jeans Men,blue,39,5879,2017-09-25,7200,The clothes are very comfortable to wear
 300,HWEC-L-2531-#xP8,2017-09-23,A,2017 autumn new shoes women,brown,M,403,2017-09-26,607,good
 100,IQPD-M-3214-#yQ1,2017-09-24,B,2017 Autumn New Wide Leg Pants
 Women,black,M,3045,2017-09-27,5021,very good.
 350,LPEC-N-4572-#zX2,2017-09-25,B,2017 Autumn New Underwear Women,red,M,239,2017-09-28,407,The seller's service is very good
 110,NQAB-O-3768-#sM3,2017-09-26,B,2017 autumn new underwear
 women,red,S,6089,2017-09-29,7021,The color is very good
 210,HWNB-P-7879-#tN4,2017-09-27,B,2017 autumn new underwear women,red,L,3201,2017-09-30,4059,I like it very much and the quality is good.
 230,JKHU-Q-8865-#uO5,2017-09-29,C,2017 Autumn New Clothes with Chiffon
 Shirt,black,M,2056,2017-10-02,3842,very good

Step 2 Log in to OBS Console, click **Create Bucket**, set the following parameters, and click **Create Now**.

Table 9-5 Bucket parameters

Parameter	Value
Region	Dublin
Data Redundancy Policy	Single-AZ storage
Bucket Name	mrs-datasource
Default Storage Class	Standard
Bucket Policy	Private
Default Encryption	Disable
Direct Reading	Disable
Enterprise Project	default
Tags	-

Step 3 Wait until the bucket is created.

Step 4 Switch back to the MRS console and click the name of the created MRS cluster. On the **Dashboard** page, click the Synchronize button next to **IAM User Sync**. The synchronization takes about 5 minutes.

Step 5 Click **Nodes** and click a master node. On the displayed page, switch to the **EIPs** tab, click **Bind EIP**, select an existing EIP, and click **OK**. If no EIP is available, create one. Record the EIP.

Step 6 (Optional) Connect Hive to OBS.

 **NOTE**

Perform this step when Hive interconnects with OBS. Skip this step when Hive interconnects with HDFS.

1. Go back to the MRS cluster page. Click the cluster name. On the **Dashboard** tab page of the cluster details page, click **Access Manager**. If a message is displayed indicating that EIP needs to be bound, bind an EIP first.
2. In the **Access MRS Manager** dialog box, click **OK**. You will be redirected to the MRS Manager login page. Enter the username **admin** and its password for logging in to MRS Manager. The password is the one you entered when creating the MRS cluster.
3. Interconnect Hive with OBS by referring to [Interconnecting Hive with OBS](#).

Step 7 Download the client.

1. Go back to the MRS cluster page. Click the cluster name. On the **Dashboard** tab page of the cluster details page, click **Access Manager**. If a message is displayed indicating that EIP needs to be bound, bind an EIP first.
2. In the **Access MRS Manager** dialog box, click **OK**. You will be redirected to the MRS Manager login page. Enter the username **admin** and its password for logging in to MRS Manager. The password is the one you entered when creating the MRS cluster.
3. Choose **Services > Download Client**. Set **Client Type** to **Only configuration files** and set **Download To** to **Server**. Click **OK**.

Download Cluster Client

Download the  client. The cluster client provides all services.

Select Client Type: **Complete Client** Configuration Files Only

Select Platform Type: x86_64 aarch64

Save to Path: 

OK

Cancel

Step 8 Log in to the active master node as user **root** and update the client configuration of the active management node.

```
cd /opt/client
```

```
sh refreshConfig.sh /opt/client Full_path_of_client_configuration_file_package
```

In this tutorial, run the following command:

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

Step 9 Switch to user **omm** and go to the directory where the Hive client is located.

```
su - omm
```

```
cd /opt/client
```

Step 10 Create the **product_info** table whose storage format is TEXTFILE on Hive.

1. Import environment variables to the **/opt/client** directory.

```
source bigdata_env
```

NOTE

If **find: 'opt/client/Hudi': Permission denied** is displayed, ignore it. This does not affect subsequent operations.

2. Log in to the Hive client.

- a. If Kerberos authentication is enabled for the current cluster, run the following command to authenticate the current user. The current user must have the permission for creating Hive tables. . Configure a role with the required permissions. . Bind a role to the user. If Kerberos authentication is not enabled for the current cluster, you do not need to run the following command:

```
kinit MRS cluster user
```

- b. Run the following command to start the Hive client:

```
beeline
```

3. Run the following SQL commands in sequence to create a demo database and the **product_info** table:

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

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

Step 11 Import the **product_info.txt** file to Hive.

- Hive is interconnected with OBS: Go back to OBS Console, click the name of the bucket, choose **Objects > Upload Object**, and upload the **product_info.txt** file to the path of the **product_info table** in the OBS bucket.

- Hive is interconnected with HDFS: Import the **product_info.txt** file to the HDFS path **/user/hive/warehouse/demo.db/product_info/**. For details about how to import data to an MRS cluster, see section **Managing Data Files** in the *MapReduce Service User Guide*.

Step 12 Create an ORC table and import data to the table.

1. Run the following SQL commands to create an ORC table:

```
DROP TABLE product_info_orc;

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

2. Insert data in the **product_info** table into the Hive ORC table **product_info_orc**.

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

3. Query whether the data import is successful.

```
select * from product_info_orc;
```

----End

Creating an MRS Data Source Connection

Step 1 Log in to the GaussDB(DWS) console and click the created data warehouse cluster. Ensure that the GaussDB(DWS) and MRS clusters are in the same region, AZ, and VPC subnet.

Step 2 Click the **MRS Data Source** tab and click **Create MRS Cluster Connection**.

Step 3 Set the following parameters and click **OK**.

- **Data Source:** mrs_server
- **Configuration Mode:** MRS Account
- **MRS Data Source:** Select the created **mrs_01** cluster.
- **MRS Account:** admin
- **Password:** Enter the password of the **admin** user created for the MRS data source.

Create MRS Cluster Connection

* Data Source: ?

* Configuration Mode: MRS Account File upload
 Configure the username and password of Manager of the MRS cluster, so that GaussDB(DWS) can automatically download the configuration and credential files.

* MRS Data Source: ? [View MRS Cluster](#)
 Kerberos Authentication: Disabled

* MRS Account: ?

* Password: ?

* Use a Machine-Machine Account:
 Creates a machine-machine account named dws in MRS and uses it for interaction with MRS. This account is in the supergroup group and has all permissions. If the switch is toggled off, the configured man-machine account will be used. Ensure this account has the permissions to access data.

* Database:

Description:

----End

Creating a Foreign Server

Perform this step only when Hive interconnects with OBS. Skip this step when Hive interconnects with HDFS.

- Step 1** Use Data Studio to connect to the created GaussDB(DWS) cluster.
- Step 2** Run the following statement to create a foreign server. {AK value} and {SK value} are obtained from [Preparing the Environment](#).

NOTICE

Hard-coded or plaintext AK and SK are risky. For security purposes, encrypt your AK and SK and store them in the configuration file or environment variables.

```
CREATE SERVER obs_server FOREIGN DATA WRAPPER DFS_FDW
OPTIONS
(
address'obs.xxx.com:5443', //Address for accessing OBS.
```

```
encrypt 'on',
access_key '{AK value}',
secret_access_key '{SK value}',
type 'obs'
);
```

Step 3 Check the foreign server.

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

The server is successfully created if information similar to the following is displayed:

srvname	srvowner	srvfdw	srvtype	srvversion	srvacl
obs_server	16476	14337			
{address=obs.xxx.com:5443,type=obs,encrypt=on,access_key=***,secret_access_key=***}					
(1 row)					

----End

Create an external schema.**Step 1** Obtain the internal IP address and port number of the Hive metastore service and the name of the Hive database to be accessed.

1. Log in to the MRS console.
2. Choose **Cluster > Active Cluster** and click the name of the cluster to be queried to enter the page displaying the cluster's basic information.
3. Click **Go to manager** on the O&M Management page and enter the username and password to log in to the FusionInsight management page.
4. Click **Cluster, Hive, Configuration, All Configurations, MetaStore**, and **Port** in sequence, and record the value of **hive.metastore.port**.
5. Click **Cluster, Hive**, and **Instance** in sequence, and record the MetaStore management IP address of the host whose name contains **master1**.

Step 2 Create an external schema.

//When interconnecting Hive with OBS: Set **Server** to the name of the external server created in [Step 2](#), **DATABASE** to the database created on Hive, **METAADDRESS** to the IP address and port number of the Hive metastore service recorded in [Step 1](#), and **CONFIGURATION** to the default configuration path of the MRS data source.

```
DROP SCHEMA IF EXISTS ex1;
```

```
CREATE EXTERNAL SCHEMA ex1
WITH SOURCE hive
DATABASE 'demo'
SERVER obs_server
METAADDRESS '***.***.***.***.***'
CONFIGURATION '/MRS/gaussdb/mrs_server'
```

//When interconnecting Hive with HDFS: Set **Server** to **mrs_server** (name of the data source created in [Creating an MRS Data Source Connection](#)), **METAADDRESS** to the IP address and port number of the Hive metastore service recorded in [Step 1](#), and **CONFIGURATION** to the default configuration path of the MRS data source.

```
DROP SCHEMA IF EXISTS ex1;
```

```
CREATE EXTERNAL SCHEMA ex1
WITH SOURCE hive
DATABASE 'demo'
SERVER mrs_server
METAADDRESS '***.***.***.***.***'
CONFIGURATION '/MRS/gaussdb/mrs_server'
```

Step 3 View the created external schema.

```
SELECT * FROM pg_namespace WHERE nspname='ex1';
SELECT * FROM pg_external_namespace WHERE nspid = (SELECT oid FROM pg_namespace WHERE
nspname = 'ex1');
```

nspid	srvname	source	address	database	confpth
		ensoptions	catalog		
16393	obs_server	hive	***.***.***.***	demo	***

(1 row)

----End

Importing Data

Step 1 Create a local table for data import.

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

Step 2 Import the target table from the Hive table.

```
INSERT INTO product_info SELECT * FROM ex1.product_info_orc;
```

Step 3 Query the import result.

```
SELECT * FROM product_info;
```

----End

Exporting Data

Step 1 Create a local source table.

```
DROP TABLE IF EXISTS product_info_export;
CREATE TABLE product_info_export
(
  product_price      integer      ,
  product_id         char(30)     ,
  product_time       date         ,
  product_level      char(10)     ,
  product_name       varchar(200) ,
  product_type1      varchar(20)  ,
  product_type2      char(10)     ,
  product_monthly_sales_cnt integer ,
  product_comment_time date       ,
  product_comment_num integer     ,
  product_comment_content varchar(200)
);
INSERT INTO product_info_export SELECT * FROM product_info;
```

Step 2 Create a target table on Hive.

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

Step 3 Import the local source table to the Hive table.

```
INSERT INTO ex1.product_info_orc_export SELECT * FROM product_info_export;
```

Step 4 Query the import result on Hive

```
SELECT * FROM product_info_orc_export;
```

----End